Recursive Processing of Structured Domains in Machine Learning

ALESSIO MICHELI

December 2003

Addr: Via Buonarroti 2, 56127 Pisa, Italy
Tel: +39 050 2212 798 — Fax: +39 050 2212 726
E-mail: micheli@di.unipi.it
Thesis Supervisors:
Prof. Alessandro Sperduti and Prof. Antonina Starita
Abstract

The adaptive processing of structured information (e.g. sequences, trees, graphs) is an emerging and critical topic in current Machine Learning research. In this thesis we propose the study of methods that are based on recursive functions for the adaptive transduction from structured domains, basically in the area of Neural Networks. Specifically, the neural computing realization of this approach, i.e. Recursive Neural Networks, deals with prediction tasks for patterns belonging to a structured domain, allowing a combination of the flexibility and the robustness of connectionist models with the representational power of a structured domain in a learning system.

Exploiting this basic idea, we propose new models, analysis of theoretical properties and applications aimed at extending the potentialities of the recursive neural computing approach in the Machine Learning framework.

A family of models, based on the Recursive Cascade Correlation methodology, is introduced to deal with contextual information in structured domains, allowing the extension of the traditional causality assumption. The proposed models allow us to realize contextual transductions and to extend, with respect to the causal models, the class of functions that can be computed for direct acyclic positional graphs. The computational power of the new contextual models is assessed both theoretically and experimentally.

The extension of the recursive dynamics to the unsupervised learning paradigm allows the formulation of a general framework for self-organizing maps able to cover previous approaches in literature. We offer a uniform formalism for studying the training mechanisms, the theoretical properties and the extensions to alternative new recursive-based unsupervised methods.

Finally, the adaptive recursive approach is exploited to develop a novel methodology for scientific applications in the area of computational Chemistry, offering a new perspective to the research in drug design and in the discovery of new compounds. The aim is to develop a general and flexible method for QSAR/QSAR (Quantitative Structure-Property Relationship and Quantitative Structure-Activity Relationship) analysis and to assess the recursive approach in real-world applications. Experimental results show the efficacy of the proposed approach for coping with heterogeneous data and problems.
Acknowledgments

I wish to thank first of all my supervisors, Antonina Starita, who kindly supported me during these years, providing constant encouragement for my research, and Alessandro Sperduti, for supporting this work with his competence, contributing ideas and constructive criticism. Tonina and Alessandro, it has been, and it is still continuously, an honor to work with you.

I am grateful to the reviewers of this thesis, Pierre Baldi and Ah Chung Tsoi, for their valuable comments, which undoubtedly derive from their distinguished scientific expertise. I also thank my thesis committee members, Dino Pedreschi and Paolo Ferragina.

I wish to thank all the coauthors of the works that were the basis for the thesis. Their collaboration has made possible both my theoretical and applied research.

In particular, besides the mentioned supervisors, I would like to thank Diego Sona, who shared with me the “fun” of working with neural networks. Actually, more than a colleague, he is a friend. I’m especially grateful to Barbara Hammer. I enjoyed working with her during her visits in Italy and I’m indebted to her for the great mathematical work on the developments deriving from the unsupervised framework, which are just sketched in Chapter 7 of this work. (Barbara, I simply couldn’t improve the published materials any more; overall, working and publishing with you has been an honor).

Most of the applications in QSPR/QSAR analysis have been developed in collaboration with researchers in other scientific fields. This gave me a formidable opportunity to taste the excitement of an interdisciplinary research, and to appreciate the relevance of real-world problems in fields such as computational Chemistry and drug design, which are characterized by a strong humanitarian and scientific value.

Hence, I wish to thank people of the pharmaceutical-chemistry research group (at the Department of Pharmaceutical Sciences - University of Pisa): first of all, the leader, Anna Maria Bianucci, who was the first with us to have faith in the exploiting of recursive neural network methodology for drug design purposes. Works on QSAR analysis have been inspired by her collaboration and support. Her enthusiastic approach has been shared by a great number of collaborators, in the past and in the present, in the dynamical environment of her lab. Thanks to Michele, Antonella, Ilaria, Alice, Daniele, Marilena, Pietro, Alessio, Anna, and Lucilla (in random order)!

Also, more recently, joint works have been activated with Luca Bernazzani, Celia Duce, Vincenzo Mollica, Roberto Solaro and Maria Rosaria Tiné (in alphabetic order), of the Department of Chemistry and Industrial Chemistry - University of Pisa. They
successful joined our research through projects aimed at studying basic chemical properties of a constantly increasing set of molecules by our methods. I really appreciated their confidence in the current recursive methodology.

I wish to thank people who offered me the opportunity to visit research structures during these years. In particular, Ah Chung Tsoi (Wollongong University), who offered me the opportunity to join his research group giving me support and encouragement; Pierre Baldi (UCI), who gave me the opportunity to present preliminary application results of this thesis at University of California, Irvine; Peter Tino, for a really exciting collaboration at the NCRG - Aston University (that we have to complete, Peter!). Peter taught me a lot and I very much appreciate his friendship.

Thanks also to all the fellow researchers I met during these years, for pleasant discussions on learning systems and applications topics. I would like to thank Marco Gori, Paolo Frasconi, Francesco Masulli, and their collaborators, with whom we have shared various national research projects. Also thanks to Gianluca Pollastri, to Markus Hagenbuchner (for our joint work, a few years ago, on comparing different architectures of the models), to Filippo Portera (for the collaboration just begun on kernel based methods), to Roberto Grossi (for useful suggestions on the notation for structured data).

I wish to thank all my colleagues and friends at the Department of Computer Science in Pisa, a very inspiring place for research: among them, the past and current roommates Diego, Emilio, Hernan, Paolo, Roberto, Stefano, and the staff and all the PhD students who enjoyed spending their time with me. It has been a pleasure to join the Medialab, where I met Alessandro and Antonio, and the Neurolab, where I met Daria, Elena, and Katuscia. In particular, some of my collaborators are in the “Computational Intelligence and Machine Learning” (CI&ML) research group and our site (http://dendrite.di.unipi.it) lists all them.

**Personal thanks**

Vorrei in queste pagine ricordare persone a me molte care come Nadia e Francesco, Mila e i miei amici Simone e Giorgio.

È una grande gioia per me poter qui ringraziare i miei genitori Anna e Lello, che hanno sempre creduto in me senza condizioni, e Veronica, che mi regala ogni giorno la sua spontanea gioia di vita.

Questo lavoro è dedicato a loro.
## Contents

### I Basics

1 Introduction

1.1 Motivations

1.1.1 Research Area in the SD Processing

1.2 Objectives of the Thesis

1.3 Main Contributions of the Thesis

1.4 Plan of the Thesis

1.5 Origin of the Chapters

### II Models

3 A General Framework for the Recursive Processing of Structured Domains

3.1 Structured Domains

3.2 Recursive Structure Transductions

3.2.1 Generalized Shift Operator and Graphical Representations of \( T_G \)

3.3 Learning Systems

4 Neural Realizations in Supervised Learning: Recursive Neural Networks

4.1 Recurrent Neural Networks

4.2 Recursive Neural Networks

4.3 Recursive Cascade Correlation
4.4 Null Model ..................................................... 65

5 Properties and Analysis .......................... 67
  5.1 Computational Properties of the Recursive Neural Networks .... 67
  5.2 Analysis .................................................... 69
    5.2.1 The Stationarity Assumption .................................. 72
    5.2.2 The Causality Assumption ................................... 72
    5.2.3 Learning Tasks ............................................ 74
    5.2.4 Other Approaches ....................................... 75
    5.2.5 Assessments .............................................. 75
  5.3 Some Notes on the Comparison between Neural Recursive Models ... 76
  5.4 Avoiding Overfitting in RCC .......................... 79

6 Contextual Processing of Structured Domains by RCC .......... 81
  6.1 Preliminaries ............................................. 84
  6.2 Contextual Recursive Model ........................... 85
  6.3 Computational Properties of CRCC .......................... 88
  6.4 Formal Determination of Context .......................... 96
  6.5 Contextual Recursive Cascade Correlation .................... 105
  6.6 Special Cases and Extensions ............................ 106
  6.7 Experimental Results .................................... 107
    6.7.1 Learning Contextual Mappings for Sequences ............... 108
    6.7.2 QSPR Analysis of Alkanes .................................. 112
    6.7.3 Analysis of Internal Representations ...................... 115
  6.8 Conclusions .............................................. 117

7 A General Framework for Unsupervised Processing of Structured Data 119
  7.1 Introduction .............................................. 119
  7.2 Structure Processing Self-Organizing Maps .................... 122
  7.3 A General Framework for the Dynamic ........................ 127
  7.4 Hebbian Learning ........................................ 136
  7.5 Learning as Cost Minimization ................................ 139
  7.6 Properties of rep ...................................... 148
  7.7 Discussion .............................................. 151

8 Discussion and Related Approaches .......................... 153
  8.1 Syntactic Approach ...................................... 154
  8.2 Symbolic Approach ...................................... 156
    8.2.1 Inductive Logic Programming .......................... 156
    8.2.2 Discussion and Comparison: Advantages and Problems ........ 159
  8.3 Propositional Approaches ................................ 163
Part I

Basics
Chapter 1

Introduction

1.1 Motivations

The study of information organization in combinatorial data structures has been one of main purposes of Computer Science since its origin. The use of well-suited data organization is often the key to simplifying the representation of a problem and its solution. Specifically, structured information, representing aggregates of elements and the relationships among them, in the form of mathematical graphs, are frequently used in Computer Science theory and applications as a very useful abstraction for real data. Since graphs are a highly general form of data, their subclasses also are frequently applied to represent the analyzed domain. For instance, we will also consider sequences and trees respectively suited to represent sequential and hierarchical relationships. The main source of our study is the idea that no system or technique for data analysis can be considered complete without including methods for dealing with, at least some kind of structured information.

The disciplines that enable flexible data analysis by finding a generic model of empirical data are jointly referred to as machine learning (ML). In this light the development of techniques for processing of structured domains (SDs, i.e. domains of structured data) in the framework of machine learning theory is the natural core of our objectives. In our view extending the generality of a methodology (adaptive algorithms in this case) in order to deal with classes of data structures, is one of the most promising ways of extending the possibility of successful new applications. Moreover the generalization of the problems induced by the use of structured data may in itself be an interesting way to study in depth the basis of ML in a new light.

ML is the field of artificial intelligence (AI) concerned with programs that improve their performance through experience with respect to a class of tasks. Specifically ML aims at finding a hypothesis (or model) that fits the data, learning from known sets of examples and accounting for the performance of the inferred model to unseen or future data. The utility of the ML approach is clear in applications where it is difficult to specify (e.g. due to the lack of a clear theory) or is inefficient to use a mathematical model that solves the specific problems, especially where the traditional programming approach omits deal-
ing with uncertain and noisy data. Major requests are the availability of data that represent
the problem well and the admissibility of tolerance in the precision of results. Examples
of this type of situation range from hand-written character recognition to the modeling of
complex biochemical ligand-receptor interactions. ML has a high potentiality in terms of
extracting knowledge from data, automatic model design, width of applications range and
appeal of the philosophical and neuro-physiological interpretations. However ”learning”
is inherently a complex task and the recent advances in the formal study of computational
and statistical learning theory allow a careful and objective analysis of the difficulties in
the area (see Chapter 2).

Unfortunately most of known ML methods are limited to using flat and fixed-width
data forms, resorting to the termed attribute-value language to represent examples of the
domain. On the basis of the assumption that predetermined attributes (or features) are not
able to completely capture the complexity of structured data, as further explained through-
out the present thesis (see for instance Section 2.3), it is understandable that standard ML
models cannot handle SD-problems in an efficient and general manner. This makes the
study of approaches to the processing of structured information in a more general form
relevant.

In order to solve predictive problems, the methods we deal with must be able to learn
a mapping between a structured information domain (SD) and a discrete or continuous
space. This can be viewed as a particular type of structural input-output transduction with
an unstructured output space [52]. Prediction tasks, such as regression and classification,
can be addressed using supervised methods to fit the transduction function to the given
data. Clustering problems can be dealt with in the framework of Unsupervised learning
methods.

Structural domains characterize a rich set of problems in Computer Science. Ex-
amples are knowledge representation, graph-based data mining, geometrical and spatial
reasoning (robotics, structured representation of objects in space), speech recognition,
natural language processing, text and World Wide Web document processing, problem
solving in artificial intelligence. Frequently in such areas the goal is to extract knowledge
from data or to model the problem solution by a predictive method. Unfortunately, of-
ten the absence of general theories or the uncertainty of a priori knowledge hamper the
coding of an analytical relationship of data by algorithms or symbolic paradigms. The
following problems are examples for specific fields, of the need to combine the use of SD
and ML:

- problem solving techniques in artificial intelligence, which often require perform-
ing a search in a tree. The aim is to control the inefficiency of an exhaustive search
by heuristics. Heuristics are aimed at identifying the most promising paths of the
search tree. A significant example is given in theorem proving, since it would be
desirable to drive the search process of a theorem prover. This is done through a
ranking of the single steps, which tell us whether they are promising. Adaptive
mapping of the state of a proof coded by set of formulas, representable by a tree
structure, have been studied in [60].
• natural language processing wherever it is necessary to evaluate parser trees or any form of tree that is used to represent the structure of a sentence.

• picture processing, where a structure can represent the mutual position of relevant image components, that allows a quite uniform and roto-translations invariant representation. Applications to logo recognition were developed using adaptive methods in [49].

• speech recognition processes, that are often affected by sources of noise.

• document recognition processes, especially web documents, where a nice structure can be extracted to represent various levels of semantics; when a fuzzy evaluation of objects is required, the problem can be naturally tackled by ML techniques.

In all the contexts described above learning often affords the solution in a flexible and general way. Moreover, where traditional statistic techniques rely on models allowing simple mathematical analysis often making specific assumptions on the underlying distribution of the data, machine learning methodologies can use sophisticated models to learn complex nonlinear dependencies in data for such problems.

Structured data domains are not peculiar only to Computer Science but also in many branches of Natural Sciences. Prominent applications are emerging for the treatment of medical, biological and chemical data. Probably one of the most elucidating example of structured data in such fields, is given, for instance, by chemical structure formulas that determine primary structure of molecules, because of their natural graphical description. One of the principal goals in Chemistry is to correlate the chemical structure of molecules with their properties (e.g. boiling point, chemical reactivity, biological and pharmaceutical properties, etc.). In the field of computational chemistry the development of models for the analysis of the quantitative relationships between structure and properties or biological activities are named by the acronyms QSPR and QSAR respectively. The ML approach allows us to deal with QSPR/QSAR problems when knowledge of the mathematical relationships between data is poor or absent and when data are affected by noise and errors, as is typical when they are experimentally acquired. In this area, the use of ML methods to deal with SD provides an opportunity to obtain a direct and adaptive relationship between molecular structures and their properties, avoiding the use of any fixed coding scheme of the structure in preestablished features. This constitutes a means of introducing new methodologies in the computational analysis of chemical and biochemical data (cheminformatics). For instance, the pressure to reduce costs and accelerate drug discovery cycles provides a strong demand, of high humanitarian and scientific value, for applications of the proposed methodology to bio-medical domains. Also, interesting challenges involve the development of novel compounds and materials. In this respect, the applications presented throughout this thesis are mainly related to explore real-world problems in the field of pharmacology and chemistry.
1.1.1 Research Area in the SD Processing

The research field of our interest concerns the adaptive processing of structured information (e.g. sequences, trees, graphs) addressing the task termed “Structured domain learning”.

This study constitutes a challenge to extend machine learning capabilities and generality, opening fertile branches of theoretical research and offering a new basis for obtaining innovative solutions to real problems.

We can characterize three different fields open to research for SD processing in ML: the first concerns analyzing the general basic properties of the methods; the second is related to the formulation of innovative models and learning algorithms for the processing of SD; the third is the study of specific novel solutions to scientific application problems exploiting the properties of the methods developed.

Various methodologies approach the topic starting from different paradigms in the machine learning area. Most of them provide only a partial solution to the problem, transforming the original structured data into a flat representation enabling the application of standard attribute-value models. Alternatively, ad hoc solutions can be devised in distance-based and kernel-based methods defining, a priori, a similarity measure in the data domain guided by a background knowledge of the task. More complex and general solutions can be studied by syntactic and symbolic approaches. The latter extends the class of languages used to express the knowledge, e.g. by using first-order logic, as for the inductive logic programming (ILP) and other related approaches in the relational data mining area. Basically, the main problem unresolved is to find an appropriate balance between the expressive power of the model and the computational complexity of the learning algorithm. This is particularly critical when coping with the high complexity of a structural domain.

Currently, in most ML and data mining branches, there is an effort to upgrade and extend the traditional approaches to learning from structured data, with a growing interest for both theoretical and applied research. An overview of such studies is given in the “related approaches” sections (Chapter 8).

In this thesis we propose the study of methods that are based on a recursive realization of the (adaptive) transduction functions from structured domains. According to the ML paradigm, we give a parametric general recursive form of the hypothesis and then we use training examples of the SD to learn or estimate the unknown parameters of the adaptive model.

At the current state of the art, one of the most significant approaches based on recursive transduction functions has been developed in the context of neural computing (or connectionism). The relative importance we reserve to neural networks is related to their relevance and amplitude of application-range for real-world ML problems. In the light of SD processing, neural networks offer an example of a complete and progressive development of a ML system from flat-input models to the models for structures of increasing complexity through the extension of the recursive encoding concept. In fact, the connectionist paradigm was first adapted to deal with (temporal) sequences (recurrent neural
networks (see [161] and [101] for, respectively, a comprehensive survey of the models and methods in the field) and has recently been extended to the processing of structured domains [155] [52]. The present work follows this new research line devoted to developing neural networks for structured data.

Throughout the dissertation we intend to show how the approach based on hypothesis space obtained by a recursive processing can be considered a suitable method to naturally represent hierarchical relationships, allowing efficient handling of structure variability. Specifically, the recursive neural network models provide a possible solution to the balancing dilemma cited above, allowing us to extend the input domain to structured data while retaining the relative efficiency of the neural networks learning strategies.

Although the proposed method can be efficiently mainly applied to hierarchical relationships data (e.g. tree) than to arbitrary graphs, the flexibility of the connectionist models opens this approach to a wide range of possible applications. The study of the connectionist approach is viewed also as an opportunity to focus on some issue in the processing for hierarchical structures.

However, the recursive assumption introduces peculiar aspects, constraints and open problems in the processing of SD that require further research and have to be carefully evaluated. In this thesis we propose an analysis specifically aimed at studying such problems.

A restriction that characterizes current recursive neural network models is due to the causality assumption. This assumption does not allow us to deal with contextual transduction that may be needed for some application domains. Solutions have been devised to relax the causality assumption for sequence learning. However, new models should be introduced to cope with this limitation in more complex domains to allow dealing with contextual processing of SD.

Another emerging topic is the study of unsupervised learning in SD. So far, this research line is characterized by the presence of self-organizing map methods for sequence and structure learning which exploit a recursive dynamic but which results unrelated between them. Hence, a general formalism is required to characterize the properties of these methods in a uniform setting based on recursive dynamics.

While some preliminary applications have been pursued in literature to assure experimental validation of the recursive neural networks approach, other are being developed involving both benchmark data sets and real-world data. However, this is still a critical issue requiring further discussion concerning relative benefits and limitations with respect to specific application fields.

In respect of the arguments discussed above, most of our interest will be devoted to the neural computing approach to SD, both in dealing with its intrinsic problems and, in perspective, in using it as a basis for integration with other methodologies. We propose to follow a vertical approach through the area covering analysis of the basic properties of the method, introducing new models, and finally assessing the methodology as new tool in real-world applications.
1.2 Objectives of the Thesis

The main objective of the thesis is a theoretical and practical investigation of predictive models for dealing with structured domains in the area of neural computing. The distinctive and common hypothesis of all the proposed models is related to the introduction of a recursive dynamic in the mapping between the input structured information domain (SD) and the output space. First, we focus on the evaluation of the major benefits and possible limitations of using the recursive processing approach in machine learning. After examining the questions open in the current approach we proceed to the design and analysis of new neural network methods that refine and extend the approach to cover topics in supervised and unsupervised learning frameworks. Specifically, we aim at providing a unifying formulation to characterize the recursive unsupervised approach to SD. Furthermore, another relevant topic concerns the contextual processing of structured data that will be addressed by introducing a new family of supervised models. We formally characterize interesting properties of the new methods considering both the computational and learning aspects, and we experimentally assess the efficacy of these new models. Finally, a considerable effort is devoted to exploiting the proposed approach for developing a new methodology for QSPR/QSAR analysis (Computational Chemistry). The experiments presented in this thesis are designed to be simultaneously proof-of-the-principle and real-world applications. In fact, besides investigating the methodology and developing new models, one of the main aims is to investigate the impact of the approach on methods developed for real-world problems, i.e., to assess the reliability of the proposed hypotheses. Solutions to specific problems are provided in the frame of bio/cheminformatics since this is a very interesting field characterized by quite complex domains where the managing of relationships and structure is relevant to achieving suitable modeling of the solutions.

1.3 Main Contributions of the Thesis

Following the research line devoted to developing neural approaches to structured domain learning (see Section 1.1.1 for refs.), we propose further analysis in the thesis further analysis and new models for coping with limitations in the current solutions and for extending and exploiting the potentialities of the approach.

Unified Review and Analysis Simply stated, the key idea of the approach is to introduce a family of recursive functions for processing structured data.

We review the main concepts of the research line in a general fashion allowing us to view the neural network models as realizations of such a basic formulation. The key idea of the approach is introduced by the definition of a general adaptive SD-Recursive processing system. Current recurrent and recursive neural networks are reviewed in such context. We then propose an analysis aimed at investigating the characteristics of the approach and at discussing its major benefits and limitations. To be more specific, first we provide evidence for the suitability of the recursive neural computing method for dealing
with SD and then we present the critical aspects of the approach. Introducing the open
questions provides the ground motivating the study of the innovative solutions proposed
throughout the thesis.

A secondary result concerns the practical comparison of the recursive neural network
model introduced so far: fully-connected Recursive Neural Network (RNN) and Recursive
Cascade Correlation (RCC).

A further result concerns the improvement and the refinement of the RCC realization.
Specifically, we introduce an incremental strategy in the training algorithm that allows
adaptive recursive models able to mitigate the overfitting problem to be built and therefore
to improve the generalization response.

**Contextual Models** We introduce the first approach for dealing with contextual infor-
mation in structured domains by recursive neural models (RNN). By generalizing the Re-
cursive Cascade Correlation (RCC) technique we are able to partially remove the causality
assumption underpinning the original approach. The contextual model introduced
(CRCC) is able to compute a code for the substructure accounting for the position of the
substructure belonging to a larger structure. This enables the CRCC to consider the mean-
ing of the substructures with respect to their context in the graph. The properties of the
innovative contextual model are proved both theoretically, by formally proving computa-
tional properties of the model, and experimentally by comparison with causal models.

We propose a formal definition and determination of the context. Theorems are intro-
duced to prove the computational power of the model. Specifically, by a compact expres-
sion we characterize the “shape” of the context exploited by the model and we show its
evolution within the frame of constructive models. We then prove the capability of the
model to capture contextual information and therefore to compute contextual functions
that are not computable by RCC. Examples of such functions can be found in tasks where
the target cannot retain the causal assumption and the output for each vertex (of the in-
put structure) is a function of the whole structure. Moreover we demonstrate that some
causal supersource transduction which cannot be computed by RCC can also be computed
by CRCC. This property allows us to extend the class of discriminated structure beyond
sequences and trees, considering as well DPAG (Direct Positional Acyclic Graph) and
DOAG (Direct Ordered Acyclic Graph), i.e. CRCC allows the computational power to
extend to contextual transductions involving DPAGs. We then show that the set of func-
tions which CRCC can in principle compute is a superset of the functions computable by
the RCC causal model.

It is worth noting that the approach used in this study is independent from the spe-
cific neural implementation and can therefore be applied to other recurrent and recursive
models to compare them from a computational point of view.

Experiments confirm these results. Moreover further experiments also show an in-
teresting result in a task where theory cannot provide an answer to the problem since
no information about the soundness of the causality assumption is available. In fact, the
CRCC is shown to be able to improve performance results of RCC for a supersource trans-
1. INTRODUCTION

ductions task on a domain of trees that RCC can in principle compute. Further analysis carried out by projections of the internal representation developed by CRCC accounts for the specificity of the CRCC contextual encoding.

As a practical result, we show that the new model preserves the prediction capability of RCC on strict causal tasks and extends prediction capability to tasks that involve the context, both for sequences and structured data. Hence, as a general outcome, we suggest that the new family of models can be adopted as a basic tool whenever information on the validity of the causality assumption is not known.

Unsupervised Learning for SD The recursive approach is also exploited also for unsupervised learning and related to the previous supervised methods. In fact, we develop a unifying general framework for the extension of various unsupervised methods to the recursive processing of structured domains. In particular, we propose a uniform way of combining different existing approaches based on self-organizing maps, e.g. recursive SOM, temporal Kohonen map, SOM-SD. This general framework constitutes a base to describe, compare and implement new models and various learning strategies based on a recursive dynamics. The unified formulation of the model is also exploited to study in a unified view the theoretical properties of such unsupervised models for SD.

Some specific results are the following: we realize that the key point to characterize the different approaches is the internal formal representation of the structures on the neural map. Exploiting this concept, the dynamics of the different models is shown in terms of similarity measure functions (defining the proposed GSOM-SD frameworks); the appropriate choice of the internal representation allows the different mechanisms of the existing approaches to be instanced. In particular we discuss how the properties of such internal representation allows us to compare the different methods with respect to noise tolerance and to the in-principle capability of representation.

Moreover, the general framework allows training mechanisms to be formulated in a uniform manner. First we show how Hebbian learning for SOM based models can be formulated within the general framework for SD. We also study the extension of Hebbian learning to various topologies such as Vector Quantization and Neural Gas topologies. Another result concerns the study of learning methods based on energy functions. Hence, we propose a general formulation of the objectives of the learning process for SD in terms of an error function which must be optimized. In such a case, training rules can be derived by gradient methods. It turns out that unlike the case of unsupervised vector processing, Hebbian learning is only an approximation of gradient dynamics of appropriate cost functions for SD. Again, the study is extended to transfer alternatives such as Neural Gas and Vector Quantization to the recursive processing of SD.

Related Approaches Finally, the overview of complementary approaches allows us to evaluate the proposed recursive approach with respect to the broad area of relational data mining and related approaches. Specifically, we propose a seminal comparison of the relative advantages and drawbacks of the recursive neural networks and inductive sym-
bolic approaches to the processing of SD. This provides an alternative way to discuss the characteristics of the proposed methodology in a wider frame. The final goal is to give some suggestions for future work. Hence, this discussion can motivate the need to bridge the various approaches aimed at delineating a general framework and developing hybrid models for the processing of SD.

**Novel Methodologies in QSPR/QSAR Analysis and Applications**  The application part is devoted to introducing a novel approach to QSPR/QSAR analysis and to deal with scientific discovery in this area. Moreover, the presented experiments allow us to evaluate the introduction of the new ML approach in a relevant set of scientific problems.

Basing on the idea of functional transductions, we develop a general framework that allows us to present in a unified fashion both the traditional and the innovative approach to the QSPR/QSAR analysis. In this formalism we can prove that RNNs can be a suitable tool for the tasks pursued in the QSPR/QSAR analysis.

The results achieved are at the state of the art and some of them allow us to outperform the results obtained by using traditional methods on the same tasks and data. We show the generality and flexibility of the new approach when coping with different tasks and heterogeneous data (chemical compounds). To show this we deal with problems both in the area of QSPR and QSAR analysis. We exploit the approach specifically for the following tasks:

- QSPR analysis of *alkanes*, i.e. the prediction of boiling point temperature for a group of acyclic hydrocarbons; the aim is the assessment of our novel method by comparison with a multilayer feedforward network approach based on an ‘ad hoc’ molecular representation that yields “state-of-the-art” results.

- QSPR analysis concerned with predicting *thermodynamical properties* of the solvation process, to analyze the solvation free energy ($\Delta_{\text{sol}} \mathcal{G}$ in water) of a set of different linear molecules for classes of chemical compounds selected by the Chemical Department of the University of Pisa.

- QSAR analysis for the prediction of the non-specific activity (affinity) toward the Bz/GABA$_A$ receptor of a set of benzodiazepines. The experiments, besides their peculiar interest due to the strong therapeutic relevance of such molecules, also allow the assessment of our method by comparison against the traditional “Hansch” QSAR equation-based approach.

- QSAR analysis and design of novel molecules belonging to the class of adenine analogues (8-azaadenine derivatives), i.e. the analysis and prediction of A1 adenosine receptor ligands affinity toward the receptor. We analyze a set of molecules, selected by the Chemical Pharmaceutical Department of Pisa University, characterized by a widespread potential therapeutic interest as they may be exploited as potassium sparing diuretics, with kidney-protective properties, or in therapy of degenerative diseases such as Alzheimer. After the assessment steps carried out in the
applications presented so far, this problem allows our model to be applied as a prediction tool for new molecules. Hence, this experiment opens the field to applying the model in order to reduce the cost of the drugs discovery process.

Interesting results are also obtained in the extraction of knowledge from the trained recursive models. Specifically, the analysis of internal representation developed by the recursive neural networks applied to the “benzodiazepines” task allows us to show the capacity of the RNN model in discovering relevant structural features, e.g., the characteristics of many substituents affecting the activity of benzodiazepines, on the basis of the association between the molecular morphology and the target property. This is a further step toward the assessment of the model as a new tool for the rational design of new molecules and it offers new strategies to discover new qualitative aspects of the QSAR problem.

The general result achieved, considering the full set of applications, is that the proposed approach based on recursive processing of structured domain by neural networks fits the analyzed problems and it introduces benefits in the current methodology for QSPR and QSAR studies.

Finally, the experience gained in the QSPR/QSAR applications has been recently partially transferred onto a software system aimed to support the designer of the applications in the pre-processing and the post-processing phases of the data mining cycle for the specific case of SD problems. The system is composed by an integrated set of tools especially dedicated to process chemical structures for a prediction system based on RNN.

1.4 Plan of the Thesis

The thesis is organized in three parts.

In the first part (BASICS) (Chapter 2) we first briefly review the basic concepts of the Machine Learning area; we then provide an elementary introduction to the basic neural network models for supervised and unsupervised learning (clearly, the reader expert in ML and neural networks can skip Section 2.1 and Section 2.2). In Section 2.3 we introduce our approach to SD learning informally defining the peculiarities that characterize the adaptive processing of SD from both the data domain and computational learning model aspects. We then introduce the main structured domains application problems addressed in this thesis. The motivations of the application aim are explained in the context of the computational Chemistry research for the development of new materials and new drugs.

In the second part (MODELS) we first define the recursive method; then we exploit it to present the neural network for structures and to frame the original contributions, analysis and models of the thesis. Some experiments involving artificial and real-world data sets are presented as “proof-of-the-principle” for the new models and to confirm the
analysis. However, most of the experiments addressed to real-world problems and not directly related to assess single models are postponed to the application chapters (PART III).

In Chapter 3 we introduce the technical background that makes up the general framework to present all the models proposed as original contribution. We present the notation used for the main class of graphs that constitutes the structured domains considered. We then outline the functional graph transduction based on recursive functions and related notions, that allow a general adaptive recursive processing system for SD to be defined.

Chapter 4 briefly reviews the neural networks for SD in the framework introduced. Detailed descriptions of the model exploited in the applications, i.e. the Recursive Cascade Correlation, are also given.

In Chapter 5 we discuss the properties of the recursive approach and models beginning with a short review of the main previous theoretical results. This is followed by a more detailed presentation of the advantages and of the open problems arising in the method. Hence, this part is central to uniformly motivate the study of new ideas and models that will be introduced in the other chapters of PART II. Problems related to designing the basic recursive neural network architectures are studied in the final Section of the Chapter.

The major original contributions of PART II are presented in Chapters 6 and 7.

In Chapter 6 we introduce an original model (CRCC) to deal with contextual processing of structured domains. We formally characterize the computational properties in comparison to the previous causal models and we present experiments aimed at assessing the efficiency and the computational power of the new model.

In Chapter 7 we focus on the unsupervised methods based on self-organizing maps characterized by a recursive dynamic. We propose an original general framework (called GSOMSD) which transfers the idea of recursive processing of structured data for supervised recurrent and recursive networks to the unsupervised scenario. We show how known, and also fresh, models for recursive unsupervised learning are instances of the GSOMSD framework. Investigation of learning algorithms, both Hebbian and energy function learning, and some of their theoretical properties are provided in the developed unified framework.

In Chapter 8 we discuss the recursive neural networks method with respect to related approaches. The aim is to sketch future directions of research within a wider framework considering emerging branches of research in the Structure Domain Learning and Relational Data Mining.

The THIRD PART (APPLICATIONS) is devoted to developing innovative solutions to real-world problems in computational chemistry and to assessing the proposed methodology within that frame. The aim is to think in terms of the problem that needs to be solved. Hence, firstly we review the principles of the traditional approaches to the area of QSAR/QSPR analysis. Then, the main concepts underpinning the basic models introduced in PART II are briefly resumed and explained in a more intuitive form, i.e. with a disseminative aim. Although this may cause some redundancy in the presentation, the
intention is that PART III can be read by itself. This is highly important for introducing the work in a interdisciplinary setting that involves researchers in Pharmacology/Chemistry and Biology, with potentially low background knowledge in Computer Science, aimed at using new computational tools to solve Cheminformatic problems encountered in QSPR, QSAR and drug design applications.

In Chapter 9 we present the methods for QSPR and QSAR analysis. First we give the details of the major traditional approaches. We then generalize such approaches proposing a new view of the area through a functional transduction framework. We use this framework to present the novel approach proposed (that is based on recursive neural networks) and to investigate the relative benefits in a unified perspective. Finally we outline general representational issues for chemical compounds: the representation used for specific classes of compounds are presented in Chapters 10 and 11.

Specific examples and the experimental results, including performance and qualitative comparisons with traditional QSPR/QSAR, covering a heterogeneous set of tasks and compounds, are presented in Chapters 10 and 11.

In Chapter 10 we deal with QSPR analysis of alkanes (prediction of the boiling temperature) to evaluate our method with respect to traditional approaches, including an “ad hoc” method for the problem based on neural networks. A second QSPR application constitutes preliminary work aimed at building a novel model for the prediction of thermodynamical properties of various classes of chemical compounds. We consider the solvation free energy ($\Delta_{\text{sol}}, G^\circ$ in water) of a set of various linear molecules. This problem allows the comparison with traditional group contribution methods.

In Chapter 11 we deal with a well know problem for QSAR analysis, i.e. the prediction of the non-specific activity (affinity) toward the benzodiazepine/GABA$_A$ receptor by a group of benzodiazepines. We compare our approach to the traditional QSAR “Hansch” approach. Qualitative aspects (structure-activity relationship) of the analysis are addressed through the studies on the internal representation developed by the RNN and compared with prior knowledge in the field.

The second QSAR application is a first step in testing an approach, by our method, to the challenge of discovering new drugs by predictive models. The analysis of a group of adenosine and 8-azaadenine derivatives is completed by the prediction step for a theoretical library of new compounds.

Finally, we describe the characteristics of an integrated software tool projected to support the development of applications of RNN in SD.

Conclusions and a summary of possible future directions of research are drawn in Chapter 12.
1.5 Origin of the Chapters

Many results of this thesis have already been presented at conferences and/or have been already been published, jointly with various coauthors. In order to exploit the interdisciplinary potentially of the application topics, an effort has been made to divulge the results both in Computer Science/Machine Learning/Computational Intelligence community and in the computational Chemistry community.

In particular:

- The model for contextual processing (Chapter 6), as it appears in [124], was first developed as a Bi-causal model applied to sequence domain. The basic idea of the CRCC model for SD was introduced in [125]. An extended journal version of the CRCC methods has been submitted for publication [126]. Some basic concepts of the theoretical results on the formal determination of context have been discussed in [127].

- Chapter 7 describes and develops the ideas introduced in [75]. A journal version on these topics has been accepted for publication [76]. An extended version of these results, including some further developments, is reported in [74].

- For PART III, the first QSPR/QSAR applications appeared as a journal version in [21]. A preliminary version restricted to the benzodiazepine problem can be found in [20].

  - The framework presented in Chapter 9 and the results on the analysis of the internal representation developed by recursive neural networks appear in [128].
  - A preliminary work for predicting thermodynamical properties was summarized in [15, 16].
  - The design of new biologically active adenosine and 8-azaadenine derivatives was introduced in [19] and in [129].
  - The RCC models with the idea for the i-strategy were summarized and used in [20, 128, 21].

A summary of the approach to QSPR/QSAR appears also as a book version in [22].
1. INTRODUCTION
Chapter 2

Background

In this Chapter we review some basic notions that introduce to the research area considered in the thesis. A basic overview of machine learning (ML) is given in Section 2.1. In Section 2.2 we recall the basic notions of neural-computing.

Finally, before introducing the models for the processing of SD in Part II, we delineate the attributes that characterize the learning of data structures. We begin by introducing the description and the peculiarity of a SD-problem for domains and models (Sections 2.3.1 and 2.3.2) and finally we give a concrete example of applications problems characterized by structured domain in Sections 2.3.3 and 2.3.4.

2.1 Machine Learning

In the following we present a brief introduction to some basic concepts and terms of Machine Learning (ML) theory focusing on the approaches that are studied in the thesis.

ML deals with the problem of inferring general functions from known data. The ML methods are particularly useful for poorly understood domains where the lack of a clear theory describing the underlying phenomena or the presence of uncertain and noisy data, hampers the development of effective algorithms that solves the specific problems. The techniques developed in the field of ML can be exploited for a variety of applications and they are playing a central role also in other (related) disciplines, such as the “Knowledge Discovery in Databases” (KDD) and “Data Mining” (DM) that deal with the overall cycle process of extracting unknown and useful information from the data.

For the sake of introduction, we find it useful to firstly describe the ML topics of our interest in terms of few key design choice concerning the data, the tasks, the models, and the learning algorithms:

- **Data** is the set of facts available for the problem at hand. Examples from the application domain are represented in a formalism that should be able to capture the structure of the analyzed objects.
- The **task** defines the aim of the application. It implicitly defines what is to be considered the useful nature of the results. The original problem is moved into the
problem of learning of some designed target function. The form of task is related to the type of feedback information available from the system that we are modeling. Tasks that we consider in the following can be, for example, roughly partitioned into predictive (classification and regression) or descriptive (cluster analysis, association analysis, summarization, etc.).

- The model is characterized by the type of hypothesis, i.e., the function fitting the data of interest, used to describe the solution for the task. The representation of such functions defines the space of hypotheses. In ML the hypothesis (model) is used to approximate the target function. More in general the hypothesis can be an expression in a given language that describes the relationships among the data.

- An algorithm is used to learn the best hypothesis according to the data, the task and the class of considered models.

The data that we consider can be of different types according to the representation formalism, e.g.,:

Single flat table: all the objects in the data set are represented with a matrix of data, where for each example there is a fixed-size vector of properties (or features) of the object, according to an attribute-value language. The features can have real values (continuous attribute) or nominal values (discrete or categorical values).

Structured domain: the objects in the domain are sequences (lists), hierarchical structures (e.g., trees), graphs, relational data, multi-table representations, and so on. In Data Mining, structured objects correspond to objects of different types (multiple relations in a relational database), e.g., database with more than one table.

In the ML area the tasks are mainly distinguished according to the type of information available as data. The task can be supervised, when information on the desired model response is given for each example (or sample), or unsupervised when this information is not available.

In the framework of supervised learning we distinguish the following terms, according to the two different computational tasks:

- Classification: means to approximate a discrete-valued function. The function maps a pattern into a \( M \)-dimensional decision space, where \( M \) is the number of categories or classes \((M \geq 2)\). For \( M > 2 \) we speak of multi-class problems. A special case is given for binary classification, with \( M = 2 \), called also ”concept learning” within the pattern recognition framework.

- Regression: means to approximate a real-valued target function. The function maps a pattern into a continuous space. The problem may be also viewed as ”curve fitting” in multidimensional space.

In the following we will consider also an unsupervised task:

- Clustering: means to determine useful subsets of a unclassified set of data sample.
To build a **model** able to capture the underlying relations among the data, according to the goal defined in the task, is the main aim of the machine learning methods. To specify a model we need to delineate the class of functions that the learning machine can implement (hypothesis space). The panorama of ML models is quite large. The classes of hypotheses that can be considered include: equations, classification and regression trees, predictive rules (for association or prediction), distance-based models, probabilistic models (Bayesian networks), neural networks and kernel-based learning machines (e.g. SVM).

Most of the known ML methods use flat representation of data (fixed-size vectors of features in one table). However an attribute-value language is not able to capture in a natural way the complexity of a structured domain. ML models have been specifically investigated to handle structured data. The new and of increasing interest area of ML dealing with structured domains can be referred to as “relational data mining” or “structured domain learning”. Inductive Logic Programming has been proved successful for relational data mining tasks involving concept learning. Recurrent neural networks have been applied to model sequences for several types of classification and regression tasks, while Recursive neural networks extend the input domain to hierarchical structures.

Learning **algorithms** perform a (heuristic) search through a space of hypotheses that are valid in the given data. Each model listed above, that uses a parameterized space of hypotheses, has a corresponding learning algorithm allowing to adapt the free parameters of the model to the task at hand. For example, we can mention multiple linear regression as a learning algorithm for equation models or the covering algorithm as a learning algorithm for rule induction. Some other ML methods do not rely on the construction (and “learning”) of a global hypothesis approximating the target function. We can mention, in the class of distance-based methods, the **nearest neighbor** (and its variants, e.g. \( k \)-NN) and other more complex instance-based approaches, such as case-based reasoning.

Beyond the specific characteristics of the different models there are some common concepts valid for every predictive machine learning model that we will use in the following.

A ML model is supposed to be used in two different states (the two phases could be interleaved): the **learning phase**, and the **prediction phase**. The learning phase corresponds to the building of the model. A hypothesis (a function \( h \) that fits the data of interest) is constructed on the basis of a set of known data, the **training data set**. The training data are the “experience” that the model tries to learn from. Therefore, this data set should be a representative sample of the real distribution of the problem at hand. The prediction phase is the operative one. The model is used to compute an evaluation of the learned function over novel samples of data. The knowledge acquired in the learning phase should allow the model to predict with reasonable accuracy the correct response for previously unseen data. The estimation of this accuracy is the critical aspect of each ML application and the rational basis for the appropriate measures of the model performance. The **holdout** method is the most used approach for this estimation: a set of known data independent of the training set, called **test data set** is used to perform prediction. An appropriate mea-
sure of the performance over the test set can estimate the accuracy (or the generalization capacity) of the model. Different statistical techniques can be used to improve the estimation accuracy: e.g. *k-fold-cross validation, bootstrapping*, etc. [23] [81]. If the performance on the test set is used to choose between different models or different configurations of the current model (i.e. it is used to drive the building of the predictive model), another set of data called *validation set* is used to assess the final prediction accuracy of the system. Committee and ensemble techniques can be used to improve the performance of the single models (e.g. [23]).

More formally in the ML framework the learning problem is viewed as a function approximation problem (approximation of a unknown target function). Different situations concerning the type of feedback or prior knowledge available to the learning system share the same background principles of ML. We present these elements mainly in the setting of the learning problem viewed as the problem of multivariate function approximation from sparse data.

In supervised learning, the data set $\mathcal{D}$ is a set of pairs $(x, d)$, called examples, where $x$ is the input value or vector (called instance) and $d$ the desired value given by a teacher to represent the observed values of a function $f$ that models the real source of data. The intrinsic error $(d - f(x))$ is the noise term carried by the teacher.

The learning machine is capable of implementing a set of input-output mappings $\mathcal{H}$ (the space of hypotheses). The function $h$ is called a *hypothesis*. The problem is that of selecting the particular function $h \in \mathcal{H}$ that approximates the desired response $d$ over the data set in an optimum fashion, e.g. minimizing some risk functions. In particular, we want the function $h$ to be a reasonable estimate of the functional relation between previously unseen pairs $(x, f(x))$ (prediction or *generalization* property). The learner can hypothesize, or estimate, the target function that produces $d$, also called *target value*. The error of a hypothesis $h$, i.e. the risk associated to it, is given by a loss or cost function $L()$ that measures the distance between $d$ and $h(x)$. A common example is given by the quadratic error function:

$$L(h(x), d) = (d - h(x))^2$$ (2.1)

The average error over all $\mathcal{D}$ is the so called *expected risk* $R$. If we assume that there exist and that is known the probability distribution $P(x, d)$ which governs the data generated and the underlying functional dependences, $R$ can be expressed as

$$R = \int L(d, h(x))dP(x, d)$$ (2.2)

In this framework the goal of supervised learning is to find a hypothesis $h$ that minimizes the expected risk $R$:

$$h = \arg\min_{f \in \mathcal{H}} R$$ (2.3)

where $\mathcal{H}$ is the hypothesis space.
Examples of hypothesis spaces are the sets of polynomials of degree \(n\)
\[
\mathcal{H}_n = \{ h \ | \ h(x) = a_0 + a_1 x + ... a_n x^n \} \tag{2.4}
\]
or any non-linear basis function expansion
\[
\mathcal{H}_n = \{ h \ | \ h(x) = \sum_{i=1}^{n} c_i K(x, W_i) \} \tag{2.5}
\]
where the \(c_i\) and \(W_i\) are the free parameters and \(K\) a given function. We will show in Section 2.2 the specific form assumed by equation 2.5 for the neural network model. In the above examples, the hypothesis spaces are ordered by increasing complexity given by the value of \(n\). Note that selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypothesis \(\mathcal{H}\) that the program can ever represent and therefore can ever learn.

The only available information for the learner is contained in a finite set of \(l\) examples called the training set. The training set is a set of pairs \((x_i, d_i)\) with \(i = 1...l\). We therefore cannot minimize directly \(R\) but we can approximate it using the inductive principle of empirical risk minimization [164]. It says to search a solution minimizing the functional
\[
R_{emp} = 1/l \sum_{i=1}^{l} (d_i - h(x_i))^2 \tag{2.6}
\]
where \(R_{emp}\) is the empirical risk, or training error. \(R_{emp}\) depending on the values of free parameters (omitted to simplify the notation).

The search is performed by the training algorithms finding the best values for the free parameters of the model, i.e. searching through \(\mathcal{H}\) the hypothesis that minimizes \(R_{emp}\) (fitting of the training examples). The goodness of the resulting hypothesis \(h\) is measured by the generalization error, as given by the distance of \(h(x)\) from \(f(x)\) for any possible \(x\), according to the underlying probability distribution. An estimation of the generalization error (accuracy) is given sampling \(D\) to extract a finite set, called test set, using the same probability distribution used to generate the training set.

The statistical learning theory allows to formally study the condition under which the empirical risk \(R_{emp}\) can be used to drive an approximation of \(R\) and allows to derive some bound on the generalization error to be expected. Without including any details, in the framework of concept learning, we can say that, with probability \(1 - \delta\), the following inequality holds (VC-bonds):
\[
R \leq R_{emp} + \epsilon(l, VC, \delta) \tag{2.7}
\]
where \(\epsilon\) is a function of the number of training examples \(l\) and of \(\delta\), and \(VC\) is a non-negative integer called the Vapnik Chervonenkis dimension (VC-dimension) [164]. The VC-dimension measures the capacity, or complexity, or expressive power of \(\mathcal{H}\) by the number of distinct instances of data set that can be completely discriminated using \(\mathcal{H}\).
2. BACKGROUND

$\mathcal{H}$ of type seen above, including neural networks, shown a VC-dimension that is proportional to the number of parameters; however this is not true in general. The intuition behind the VC bound is that the deviation between expected risk and empirical risk decreases with the number of data points $l$ (dimension of the training set) but increases with the VC-dimension. On the other hand the training error decreases as the capacity or VC-dimension is increased. Fixed the number of training data, the general result is that, the increase of the model capacity can reduce $R_{emp}$ at the expenses of the generalization performance (overfitting). Therefore, the challenge in solving a supervised learning problem is to realize the best generalization performance by matching the machine capacity to the available amount of training data for the problem at hand. The method of structural risk minimization [164] provides an inductive procedure for achieving this goal by making the VC-dimension of the model a control variable. The goal, in structural risk minimization, is to find a model such that decreasing the VC-dimension occurs at the expense of the smallest possible increase in training error. This minimizes the bound of $R$ (equation 2.7).

Bounds on the number of training examples can be found using the probably approximated correct (PAC) framework [162] [163].

It is worth to note that VC-dimension is a quantitative capacity measure. It measures the richness of the class of functions but it cannot delineate the class of functions implemented by a model in terms of representation of the input domain. Clearly, if the target function cannot be described in the hypothesis space due to the initial language choice, the model cannot yield a consistent solution. Two classes of functions can show the same VC-dimension however they may be able to properly represent different objects. In this view we cannot use the above concept to fully describe the computational properties of the proposed models.

Another related discussion regards a crucial trade-off concerning the hypothesis space expressivity. The choice of the hypothesis space involves some basic question entailing what the model can compute and what the model can learn. If the model cannot compute (represent) the target function it also cannot learn it. On the other hand it is often the case that models that may be shown to be capable of a certain kind of computation cannot easily be trained from examples to perform it. Since the model defines the class of hypothesis considered in the learning procedure, it specifies the computational power of the method. Anyway, too much power ‘expressive’ model lead also to problem of efficiency in the search of the solutions. The search of a balancing between expressive power of the model and the computational complexity of the learning algorithm is a critical issue of many ML methods.

To further characterize the search of the best hypothesis, it is possible to describe for each method the constrains imposed to the learner, referred as inductive bias (or simply bias). The representation of the hypotheses made selecting a specific ML model defines the hypothesis space that the learner can explore to find $h$ (language bias). The strategy adopted by the algorithm to search $h$ over the hypothesis space is the search bias.
Furthermore another decision concerns adding a strategy to control the overfitting for the particular training data set (overfitting-avoidance bias). More formally the inductive bias is the set of assumptions made by the learner in order to constrain the problem: the addition of such assumptions allows to obtained deductively the inductive result of the learner on the training set. It must be noted that these assumptions are necessary to extract regularities, i.e. a learner without bias cannot extract any regularities from data and reduces itself to a lookup-table system with loss of generalization properties [130].

As seen above, there are benefits by limiting the complexity of the model. Practical approaches can be characterized by the bias that constrains the problem. The reduction of the hypothesis space, imposing language bias, can be well-motivated by a knowledge-driven process. If there is a priori knowledge on the expected solution, it is possible to reduce efficaciously the search space. For example “regularization theory” is a nice framework in which the search of hypothesis is subject to a smoothness constraint. The heuristics induced by the search bias are the basis to deal with the problem of a complete search of the hypothesis space, which is in general intractable.

On the other side, different ML methods (models and algorithms) correspond to different inductive bias. There is no universal “best” learning method. The lack of inherent superiority of any ML methods is formally expressed in the so called “no free lunch theorem” [38]. Each method is more or less suitable according the specific task at hand. In particular, the language can be more or less appropriate to describe the relationships among the data of the application. Moreover, comparison parameters, such as predictive accuracy, speed (generating and using the model), robustness, scalability and interpretability can be considered to evaluate the quality of the method.

In the following section, we will focus on the class of hypothesis space and learning algorithms that characterize the neural computing area. We will show the properties of this method that make it suitable for many real-world problems. Alternative ML methods to NN, which can deal with SD, are reviewed in Chapter 8 with comparison aims.

### 2.2 Neural Networks

Since the original contributions of this thesis mainly concern neural networks (NN) models, this section provides an informal overview of the basic concepts of the neural computing area. Two basic models, multi-layer feedforward neural networks for supervised tasks, and self-organizing maps (SOMs) for unsupervised tasks are reviewed.

The relevance of the neural approach in ML is due to its flexibility and to the generality of the type of problems that the model can afford. Actually, NN are tools for mathematical modeling that allow to capture underlying functional relationships in the data within regression, classification, and pattern recognition problems. Moreover, they can simultaneously treat multivariate data with mixed domains entailing numerical and categorical values. Due to this flexibility many real-world problems can be naturally faced using neural networks. Actually, many tasks have been solved using such models, with performances often at “the state of the art” in the machine learning area.
These models have many properties that make them very general, and give us a clear idea of their importance in the field of ML, such as:

- NN are universal approximators (Theorem of Cybenko, 1988) [81];
- NN can learn from examples (automatic inference);
- NN can deal with noise and incomplete data;

General advantages of neural networks approach, over the other ML methods, rely on its properties of tolerance to the lack of data integrity (dealing with missing data, noise, and uncertainty), i.e. the robustness property. The regression or classification process is relatively fast once the training of the model is completed. Moreover, since many neural networks can be trained with stochastic gradient descent, also the learning algorithms have good scalability (in the number of examples) properties.

Thankful to their properties on the treatment of numerical information, both regression and classification task can be modeled using neural networks, even if the computational task requires combination of discrete and continuous values. Dealing with classification or regression often requires only slight modifications of the model architecture and of the learning algorithms.

Finally, the accuracy and generalization performance is quite high and tested for a wide set of application problems. Due to their flexibility, neural networks approach is often the first model used when dealing with new application problems characterized by poor background information.

In fact, the hypothesis space considered by the neural model learning algorithms is the continuous space of all functions that can be represented by assigning the parameters values, i.e. the weight values of the given NN architecture. This allows the model to represent a rich space of non-linear functions, making neural networks a good choice for learning discrete and continuous functions whose general form is unknown in advance.

Since the hypothesis space is a continuously parametrized space and error function is differentiable, the search of the best hypothesis, i.e. the learning algorithm, can be based on a gradient descent technique. The learning algorithm for neural models searches a class of smooth functions able to perform smooth interpolation between data points. The smoothness property characterizes the inductive bias of NN models.

The main disadvantages associated to the neural network models are:

1. poor interpretability of learned models, i.e. the black box problem;
2. empirical choice of the network topology;
3. training process dependency: training methods, training parameters values and the initial conditions (weights values assigned in the first step of the learning algorithm), have effects on the final solution (accuracy).

The first point involves both the extraction of the knowledge learned by a neural network and its representation in a form understandable by a human expert. It is known that the investigation of the meaning of the internal representation developed by a NN is not
2.2. NEURAL NETWORKS

The knowledge acquired and abstracted by the model from training data set is represented in a compactly distributed form as numerical weights across the connections of the network and the activity of the internal units. While this form of knowledge representation enables the model to good prediction performance, it is responsible for the inherent explanation difficulty. This problem, due to its importance, although not completely solved, has been already faced in various ways, e.g.:

- projecting data, typically the internal representation values, to a lower-dimensional space in which visualization is easier, e.g. by principal component analysis;
- discovering underlying clusters of internal representation developed by the neural network;
- discovering symbolic rules or an automaton associated to the model;
- computing sensitivity analysis.

The second point concerns the relationship between the model performances in particular applications and the hand-designed neural architecture. The problem of optimizing the architecture has been addressed in satisfactory manner for practical problems by the use of constructive or pruning techniques. Various models have been proposed to dynamically modify the network topology and its dimension as part of the learning algorithm. Constructive methods, also called growing methods, permit to start with a minimal network configuration and to add units and connections progressively, allowing automatically adapting the architecture to the task. Examples are given by Cascade Correlation family of algorithms [47], tiling [122], upstart [53] algorithms, etc. An example of comparison between Cascade Correlation and standard networks is in [67]. On the opposite, pruning techniques are based on a gradual reduction of model complexity, allowing to select the useful part of large initial network. A third way to approach the design of network topology is to combine different simple networks to form a committee of models [23].

2.2.1 Standard Neural Networks

Relating to ML theory (Section 2.1) a neural network realizes a mathematical function

$$ h(x) = \Phi(x, W), $$

where $x \in \mathbb{R}^n$ is the vectorial input pattern, $W \in \mathcal{W}$ is the set of real-value (multi-dimensional) vector of adaptive free-parameters (the weights) (e.g. $\mathcal{W} \subset \mathbb{R}^{m \times 2}$), and $\Phi : \mathbb{R}^m \times \mathcal{W} \to \mathbb{R}^2$. Thankful to the learning algorithms that adapt the free-parameters $W$, neural networks allow to consider an hypothesis space spanning over the continuous space of all the functions that can be represented by assigning the weight values of the given architecture.

For instance, the two-layer feedforward neural network, the most known type of connectionist model, computes a function of the type:

$$ g(x) = \sigma_p \left( \sum_j w_{pj} \sigma_j \left( \sum_i w_{ji} x_i \right) \right) = \sigma_p(W_Z \sigma(W_I x)) $$

where $x \in \mathbb{R}^n$ is the vectorial input pattern, $\sigma_i(v) = \sigma_i(v_i)$ are the activation functions, $W_I \in \mathbb{R}^{m \times n}$ is the weight matrix associated with the input space, $W_Z \in \mathbb{R}^m$ is the
weight vector used to compute the output value (in case of \( z \) output values we have \( W_Z \in \mathbb{R}^{m \times z} \); \( W_I \) and \( W_Z \) are the free-parameters of the neural model.

The computation of the function in Equation 2.8 is traditionally described in terms of a data-flow process performed by an interconnected set of computing units in a form of network. Specifically, NN models can be characterized by the design of the neural units, the topology of the network of units, i.e. the layout of the architecture, and the type of learning algorithm used to adapt the computed function to the computational task.

**Units** In Equation 2.8 each function of the type

\[
out_j = \sigma_j \left( \sum_i w_{ji} x_i \right) = \sigma_j(W_j x) \tag{2.9}
\]

can be seen as computed by an independent processing element \( j \), with output \( out_j \), called artificial neuron or simply unit. In order to not clutter the notation too much, we include the bias parameters, i.e. a linear offset \( \theta \), in the weight vector of each unit, rather than denoting it explicitly. Hence, we assume that the input vector is extended by a constant value, so that \( x_0 = 1 \) and \( \theta_j = w_{j0} \).

The computation of each unit can be further decomposed as in the following:

\[
net_j = \sum_i w_{ji} x_{ji} \tag{2.10}
\]

\[
out_j = \sigma(net_j) \tag{2.11}
\]

A graphical representation of the above equations is shown in Figure 2.1.

Adopting different designs of the neural units we obtain different neural models. For example, Equation 2.10 can be modified yielding higher order processing units. Moreover, the activation function \( \sigma \) can be of different types, e.g. it can be a binary threshold function, a linear function, or, often, a non-linear squashing function like the sigmoidal-logistic function. The sigmoidal-logistic function has the property to be a smoothed differentiable threshold function. Two examples are the asymmetric and symmetric functions, respectively:

\[
\sigma(u) = \frac{1}{1 + e^{-\alpha u}} \in [0, +1] \tag{2.12}
\]

\[
\sigma(u) = \frac{2}{1 + e^{-\alpha u}} - 1 \in [-1, +1] \tag{2.13}
\]
2.2. NEURAL NETWORKS

Figure 2.2: An example of two-layers fully connected feedforward NN with an input layer, a hidden layer, and one unit in the output layer.

Note that, depending on the class of values produced by the network output units, discrete or continuous, the model can deal, respectively, with classification or regression tasks.

Other functions can be used, such as the Gaussian function used by radial basin function (RBF) neural networks and a stochastic function used by models such as the Boltzmann machines.

**Architecture** The architecture of a NN defines the topology of the connections among the units. The two-layer feedforward neural network described in Equation 2.8 corresponds to the well-know MLP (multi layer perceptron) architecture: the units are connected by weighted links and they are organized in the form of layers. The input layer is simply the source of the input $\mathbf{x}$ that projects onto the hidden layer of units. The hidden layer projects onto the output layer (feedforward computation of a two-layer network). In equation 2.8, the connection from the input $x_i$ to the hidden unit $j$ is represented by $w_{ji}$, while the connection from the hidden unit $j$ to the output unit $p$ is represented by $w_{pj}$. A graphical representation of this network, with two layers and one output unit (see Equation 2.8) is given in Figure 2.2.

The type of function expressed by this network (i.e. Equation 2.8) has universal approximation properties for continuous functions (see for example [81] [130]). The representational capacity of the model is related to the presence of a hidden layer of units, with the use of non-linear activation function, that transforms the input pattern into the internal representation of the network. The learning process can define a suitable internal representation, also visible as new hidden features of data, allowing the model to extract from data the higher-order statistic that are relevant to approximate the target function.

It is known that the expressive power of NN is strongly influenced by two aspects: the number of units and their configuration (architecture). The number of units can be related to the discussion of the VC-dimension of the model. Specifically, the network capabilities are influenced by the number of parameters, that is proportional to the number of units, and recent studies report also the dependencies on their value sizes [10].

In the connectionist frame a variety of different architecture have been studied (see e.g. [81]). In the following we restrict our attention to few classes of NN models that are
relevant for our purposes. For instance, a well-know model based on a lattice framework of connections is the self-organizing maps (SOMs). The SOM models will be further extensively described in Section 2.2.2.

A different category of architecture, based on the addition of feedback loops connections in the network topology, is the class of recurrent neural networks. The presence of self-loop connections provides the network with dynamical properties, letting a sort of memory of the past computations in the model. This allows us to extend the representation capability of the model to the processing of sequences (and structured data). Recurrent neural networks will be the subject of the PART II of the thesis.

**Learning Algorithm** The learning algorithm allows adapting the free-parameters of the model, i.e. the values of the connection-weight, in order to obtain the best approximation of the target function. In the ML framework this is often realized in terms of minimization of an error (or loss) function on the training data set.

The fact that the hypothesis space of neural network is continuous, together with the fact that the loss or error function is differentiable with respect to the continuous parameters of the hypothesis function, allows us to use an error gradient descent technique. The use of quadratic loss function (Equation 2.1) rises to a least-mean-square (LMS) algorithm. For multi-layer neural networks (MLP) back-propagation techniques are the most popular among the supervised training algorithms [146]. In particular, the values of the error are back propagated from the output neural units to the input units, changing the weights proportionally to the influence they had on the total error ($R_{emp}$, Equation 2.6). The back-propagation training algorithms is an iterative procedure that up-dates the weight values $\mathbf{W}$ according to

$$\Delta \mathbf{W} = -\eta \frac{\partial R_{emp}}{\partial \mathbf{W}}$$

(2.14)

where $\eta > 0$ is the step size, or learning rate. Despite of its implementation simplicity and usefulness, back-propagation has the classical disadvantages of any stochastic gradient descent method (local minimum, dependency on starting points, etc.).

For different neural architectures, such as the recurrent neural networks, specific variations of the gradient based learning methods have been devised (see PART II).

### 2.2.2 SOM: Self-Organizing Map

In this section we present a specific neural computing method, based on the SOM model, as proposed by Kohonen [99], able to cope both with clustering, vector quantization and vector projection tasks.

The architecture of self-organizing map (SOM) neural networks, also called Kohonen maps, from the author name [99], consists of $N$ neurons located on a regular low-dimensional, usually 2-D, grid. The SOM is an unsupervised-learning neural network method that produces a similarity diagram of input data. In particular, it approximates
an unlimited number of input data items, by a finite set of models (neurons in the grid). The learning process specializes the models to the input data and automatically orders them on the grid along with their mutual similarity (topology-preserving map). Large-dimensional input vectors are projected down on the grid, by a non-linear projection, in a way that maintains the similarity relationships among the data.

Summarizing, the SOM can be used for:

- pattern recognition and vector quantization: the input is transformed into the closest neuron weights, i.e. the reference or “codebook” vector;
- data compression purpose: the input is transformed into the indexes, or digital codes, of the winner unit;
- projection and exploration purpose: the distribution of multidimensional data is visualized in a lower dimensional space.

For the sake of presentation, we will assume in the following that the reader has already knowledge of the terminology of the vector quantization and clustering theory. Therefore we will not define concepts like reference vector, Voronoi set, reconstruction or distortion error, $K$-means algorithms, etc. In particular, depending on the context, we call the reference vector also as cluster center, codebook, or prototype vector. In order to introduce the arguments, the notation used in this section is chosen to be close to the standard literature for SOM models. A slight different and shorter notation, apt for our purpose, will be introduced in Chapter 7.

The SOM Learning Algorithm

Formally speaking, the aim of SOM algorithm is to learn a map from vectors to a discrete output space:

$$\mathcal{M} : \mathcal{I} \to \mathcal{R}$$  \hspace{1cm} (2.15)

where $\mathcal{I}$ is an input domain (vectorial), discrete or continuous; $\mathcal{R}$ is an output space which represents the activity of SOM neural units. In particular, the $N$ neurons are organized in a grid (lattice) of units represented in $\mathcal{R}$ by their spatial coordinates. Each unit is equipped with a weight $w = [w_1, ..., w_n]$. Typically the grid is a two dimensional lattice of units and $\mathcal{I} \equiv \mathbb{R}^n$.

The specific characteristic of the SOM learning algorithm is to create a mapping aimed at preserving, as much as possible, the topological relationships among the input vectors, i.e. data points lying near each other in the input space should be mapped onto nearby map units. Note that each unit receives the same input vector $x \in \mathcal{I}$, while the output of the SOM is given by the activity of the units.

In the basic training algorithm, the SOM map is initialized choosing random values for the weight vectors. The training algorithm allows creating a SOM by an iterative modification of the free parameters of the model (weight values). At each training step, a sample vector $x$ is randomly chosen from the data set, and given to the SOM as input. Two
phases (or stages) can be distinguished in the learning process: first, in the “competitive” phase the distances between the current input and all the unit weights are computed to select the winner unit, which is the one with the weight vector most similar to the given input. Next, the “cooperative” phase allows upgrading the weight vectors of the units that have topological relationships with the winner unit.

Once all the training data have been presented for a sufficient number of cycles, the trained map can be used for new data, just applying the competitive step. In this way the SOM can be used for the purposes seen above (pattern recognition, vector quantization, data compression, projection, etc.). However the cooperative step can be applied also for the new data obtaining a continuous on-line learning version of the SOM. Full details of the algorithm are explained in the following, on the basis of the two main stages:

**Competitive stage** The current input vector $\mathbf{x}$ is compared with all the unit weights using an Euclidean distance criteria. The unit which best matches $\mathbf{x}$ is chosen as the winner unit. Standard SOM adopts a “winner-take-all” strategy according to the following equation:

$$\text{Winner } i^* = \arg \min_i \{ ||\mathbf{w}_i - \mathbf{x}|| \}$$

(2.16)

where $i$ is an index of the map units and $i^*$ is the index of the winner unit (for the sake of simplicity we omit from the notation the dependencies of $i^*$ on the current data sample $\mathbf{x}$).

**Cooperative stage** For each step of the training algorithm, the weight of the winning unit and the weight of the units in its neighborhood are moved closer to the input vector. Therefore, given an input vector $\mathbf{x}$, not only the “winning” prototype vector $\mathbf{w}_{i^*}$ is adjusted, but also vectors that are assigned to lattice sites $i$ adjacent to $i^*$ are updated, with a step size that decreases with the lattice distance between $i$ and $i^*$. The update of the prototype vector of the unit $i$, for the algorithm iteration $t$, is driven by the following equation

$$\mathbf{w}_i(t + 1) = \mathbf{w}_i(t) + \eta(t) h_{i,i^*}(t) [\mathbf{x} - \mathbf{w}_i(t)]$$

(2.17)

where $\eta(t)$ is the learning rate and $h(t)$ is the neighborhood function (or neighborhood kernel), i.e. $h_{i,j}(t)$ is a function that decreases monotonically for increasing $||i - j||$. Moreover, learning rate and neighborhood radius values are also decreased as function of the iteration counter $t$. In fact, the neighborhood kernel is chosen to be wide in the beginning of the learning process to guarantee global ordering of the map, and both its width and height decrease slowly during learning. Note that the function $h()$ is used to model the geometrical relationships between the units (reference vectors) on the SOM grid, and not in the input data spaces. A typical choice for $h()$ is the Gaussian function:

$$h_{i,i^*}(t) = \exp \left(-\frac{||\mathbf{r}_i - \mathbf{r}_{i^*}||^2}{2\sigma_{nh}^2(t)} \right)$$

(2.18)
where $\sigma_{nh}(t)$ is the range (width of the neighborhood function) and $r_i$ and $r_{i^*}$ are the vectorial locations (position of units $i$ e $i^*$) on the SOM discrete lattice. In practice, the shape of the neighborhood is chosen from standard geometrical form (rectangular, hexagonal etc. on the SOM grid) to include a finite set of the map units.

It is important to stress that the update rule of the cooperative stage is fundamental to the formation of topographically ordered maps. In fact, the weights are not modified independently of each other but as topologically related subsets. For each subset similar kinds of weight updates are performed. For each step a subset is selected on the basis of the neighborhood of the current winner unit. Hence, topological information is supplied to the map because both the winning unit and its lattice neighbors receive similar weight updates that allow them, after learning, to respond to similar inputs. Although this seems quite natural and evident, a formal proofs of ordering is difficult and out of our scope [99].

Discussion

The SOM can be regarded as an unsupervised neural computational model. It inherits some of the characteristics described in the previous section for the neural models, including the strengths and the pitfalls. We can mention, for example, the robustness to data error or data missing, and the relationship between the SOM and the topological map of the brain motor cortex (homunculus) that yields a biological interpretation of the model. However interpretability of SOMs, thankful to their visualization properties, is much higher than classical neural models. As unsupervised model in the neural computation area, the SOM is one of the most referred models, with a large amount of results coping with theoretical properties and applications.

Moreover, further advantages can be considered in comparison with other vector projection algorithms: unlike in multidimensional scaling, for example, the SOM can first be computed using any representative subset of input data; new input items can be mapped straight into the most similar models without recomputing the whole mapping. These characteristics can be efficiently combined with pre-sampling techniques in the KDD process. The convergence of the SOM algorithm takes further advantage from a proper initialization of the map.

Finally, it must be always considered that to obtain good results with SOM algorithms as a vector quantizer, the topology of the lattice of units has to match the topology of the space $\mathcal{I}$ which is to be represented. To overcome this pitfall different algorithms have been proposed. For instances, in Neural Gas [120] the neighborhood set is ranked on the basis of the distance with the input data instead of the distance over the lattice of units. However, often, the benefits of these alternative methods have to be paid with the loss of the SOM visualization properties. The Neural Gas family of algorithms will be presented in Chapter 7.
2.3 Moving to Structured Domains

The neural network and other ML methods described so far are restricted to deal with flat representation of the input domain. The aim of this section is to informally introduce the innovative parts of the thesis delineating the elements that characterize the processing of structured information, i.e. the domain and the model, with respect to the traditional approaches. Finally we can introduce an example of real-world problem characterized by structured domain and an instance that will be used as benchmark problem to test and compare the new models (PART II).

2.3.1 Data Domains

The first design choice for a learning system is the collection and representation of the problems data. The set of represented data constitutes the training experience from which the system will learn. The type of information available can have a significant impact on the qualitative response of the learner.

The instance of data entry for the system is a quantitative or structural description of an object referred as pattern (in ML terminology). The most frequent, and often convenient, way to represent the data is the form of a pattern vector with fixed dimension (or tuple of constants). The dimension \( n \) of the vector corresponds to the number of measurements, or features, or attributes. This form is referred here as flat because there is no hierarchical arrangement of the elements of the vector. The language used to describe patterns is termed attribute-value language. Primitive elements occurring in variable number of instances, with a sequential ordered structure, are in principle representable in a flat form. However, the characteristics of sequential components are best described by a string (or list) of symbols, i.e. the simplest case of structure relationships.

Unlike the flat description, this type of representation yields patterns composed by primitive elements whose meaning is related to the basic structure of the input data. A more powerful approach is realized through the use of tree representations. Tree structure is the common form to convey hierarchical ordered information. More complete ways to represent relationships between primitives lead to the general concept of graph. Labeled graphs may represent in the same structure both vector patterns and their relationships.

The choice of the appropriate representation, considering both the selection of the representational schemes, e.g. among the ones presented above, and the selection of the set of distinguishing features that convey information, is strongly problem oriented. In the case of structured representations, besides the domains where the patterns are naturally described by resorting to structural relationships, like in the examples given in Chapter 1, there are problems in which structured nature of the data is not immediately recognizable but a structural representation is convenient. The reason may be of efficiency: for example, an improved representation can be obtained reducing the amount of data required when the structure allows to consider the repetitive nature inherent in data or when it is possible to extract invariant elements, like in image analysis applications. For instance, in the example of a geometrical image we can represent an object as structured composition of
symbols instead of resorting to a digitalization scheme. Other reasons to adopt structured representations are related to the possibility to increase the information contents by using relationships.

A basic problem is however to define a suitable strategy to extract the structure from data. Even in the case of domains that are naturally describable by structures we have to decide the information level of details, for example which type of information can influence the topology or can be included in the graph vertexes, which information is represented by the label of the node, and so on. The case of Chemistry gives us a natural example of problem characterized by set of structured objects, i.e. the 2D graph of the chemical molecules. In that case the graph represents a suitable model to represent in a very general and abstract form, in the same graphical paradigm, both the information at atom and bond level and, potentially, the background information of expert that allows us to identify functional groups in the selected compounds.

Moving to consider structured data as input pattern of learning systems we have to extend the learning process to take into account:

1. Information type and contents:

   • Structured information expressed in the form of various classes and subclasses of graphs. Instances of subclasses that will be considered are lists, trees, directed ordered acyclic graphs (DOAGs) or directed positional acyclic graph (DPAGs), DOGs and DAGs. In the following we will refer alternatively to structured information or data structures meaning in both cases a combinatorial data structure represented by the couple of sets \((V,E)\), where \(V\) is the finite set of vertexes (or nodes) and \(E\) the set of edges, i.e. a graph in the general case, or its subclasses. \(E\) is constituted by unordered or ordered pairs of vertexes for undirected or directed graphs respectively.

   The information conveyed by structures involves:

   - information related to each vertex (typically encoded by a fixed-size label);
   - single connections between vertexes of the structure;
   - local information, i.e. the relevant information can be carried out by specific part of the whole structure. To give an example, we may consider the occurrence of a specific vertex or specific substructures occurring in the pattern, or even specific substructure occurring in specific positions, etc.
   - global information, i.e. the information that inherently depends (also) on the whole topology of the structure;
   - target information, i.e. the information that collect the desired response of the learning system. Such information can be available (supervised learning) or not explicitly available (unsupervised learning); moreover it can be defined at different levels, i.e. the target can be used to label the global structure, or some substructures or each vertex of the structure. Discrete or continuous values can characterize different tasks.
2. Variability:

- size of the structure, that depends upon the number of vertexes, the out-degree, the depth, etc.
- relative arrangements of single vertexes inside the structure topology.

In order to achieve a suitable processing of structure information, a carefully evaluation of the previous aspects should be considered. The main consequences can briefly sketch in the following points:

1. The representation of structures using a fixed-size attribute-values language is generally inadequate: it is unnatural, difficult, incomplete or, otherwise, inefficient. Typically, the used representation scheme is decided a priori and cannot change automatically according to the task, i.e. it is not adaptive.

2. The learning is influenced by the growth of the proportion between high combinatorial number of possible data examples (large hypothesis space dimension) and the presence of a finite and small training set.

First attempts to deal with SD used a predefined mapping, performed in the preprocessing phase by the extraction of features, to static attribute-value representations. This procedure often compromises the explicit information carried by the relationships among the components of data described in the structure. Moreover the design of a static encoding of structure has drawbacks for the automatization of the processing cycle and for the learning task. In fact, since a specific heuristic for the problem is necessary to extract suitable features, the encoding scheme is dependent on the specific research area or even on the specific task.

In computational chemistry, for example, topological indexes (see [72] and Chapter 9) are mathematically defined for chemical graphs. This approach has disadvantages because the following problems have to be solved: the uniqueness of representation, the relevance of the selected attributes, and the efficiency of the procedure to select and to compute such features. Moreover the selection of features is strongly task-dependent and possible only recurring to an expert in the field and using an expensive trial and error approach. More in general, any a priori definition of the encoding process is constrained to be: or specific and knowledge-dependent and therefore not extendible to different problems, where the relevance of the established features may change dramatically; or as general as possible and therefore requires the code not to be the same for different graphs. Unfortunately the property of uniqueness is not a trivial problem, for example it is related to the isomorphism problem of graphs, and may lead to representations with poor relationships with the relevant properties, making complex the learning task, especially for the generalization aspects.

Other types of representations have drawbacks related to the information contents and variability of SD. In principle the encoding of a complex object by an object of an inferior degree of complexity can preserve all the information but at the expenses of a remarkable increase in the size of representation: a fixed-size vectorial representation, if it is
not incomplete, should be dimensioned to the maximum size allowed to the graph, and therefore it is potentially wasteful. We can also mention that the vectorial representation should preserve the information on the edges, for example by using an incidence matrix, which requires a quadratic increase in the dimension of representation. Another representation scheme may be obtained by encoding graphs through sequences: for instance, in the case of binary-trees, a representation based on nested parenthesis is a way of creating a sequential representation that makes possible to reconstruct the tree. Since the number of vertexes explodes exponentially with the height of the trees, even small trees give rise to long sequences. Furthermore, the sequential mapping of data structures is likely to break some nice regularities inherently associated with the data structure (that can be relevant for the learning task). In ML context, these inefficiencies of the representation exacerbate learning problems connected with input dimension, especially for the models whose input dimension has relationships with the number of free-parameters, such as the neural network models.

Point 2 concerns an issue of the ML theory. The richness of information allowed by relationships is added to the normal trade-off between the expressiveness of hypothesis space and the efficiency, i.e. computation time and number of examples required to learn. The high combinatorial number of possible data examples cannot always easily be compensated by a high cardinality of the training set. This induces a stronger need to impose conditions and constraints in the ill-posed problem to find the correct mapping in a potentially large hypothesis space. This demand of parsimony is solved in different ways by the different approaches to SD.

### 2.3.2 Computational Learning Model

The computational learning tasks in which we are involved are:

- Function approximations by supervised learning: classification and regression;
- Clustering by unsupervised learning;

The core of the differences between flat and structured processing is that while flat processing requires a set of attribute values to represent structural features, or a metric defined a priori on the SD, methods for SD processing view data not only as aggregate of elements but they are also able to capture their relationships and extract the topological information directly from data by adaptive processing.

We can summarize in the following the characteristics that make, by our point of view, a ML model suitable for the SD processing, i.e. what we mean for adaptive processing of structured information:

**Representation of SD:** the hypothesis space has to be able to represent hierarchical relationships;

**Adaptive transduction:** (implicit) measure of similarity on structures developed by the model has to be adaptive, i.e. inferred by the set of training examples and according
to the specific task. The transduction of the structure (mapping of the structures to a unstructured domain) must depend on contextual information, i.e. it has to be not-algebraic, and adaptive. An apt learning rule must exist in order to develop a (internal) knowledge representation of structured information;

**Handling of variability:** the method has to be able to deal efficiently with structure variability.

A nice but not essential requirement is that SD processing should be viewed as generalization of processing of flat or attribute-value information. This is achieved when SD processing applied to flat data reduces to the processing carried out by standard model. Moreover the model should support both the possibilities (structured and unstructured data). In these cases, the efficiency should properly scale from simple to complex data representations.

Other requirements correspond to general requirements for ML models:

- evaluation of the suitableness of bias of the model in relation with the problem;
- efficient learning;
- good generalization performance;
- knowledge extraction capabilities.

The description of the specific characterization of the models in the light of the above frame will be done in Section 5 of PART II for the described neural computing approaches.

### 2.3.3 Applications: Cheminformatics

Let us consider a concrete example of real-world problem characterized by structured domain.

The term *Cheminformatics* is emerging to delineate the integration of the studies in Chemistry with the advancements of biological screening and informatics models techniques used to guide drug discovery, and more in general aimed at developing new chemical compounds. The objective of the different approaches in the area is to provide methods to discover and to design promising compounds with desired properties (e.g. biorespons). A specific interest is on the methods that use computational tools allowing the prediction of the behavior of the compounds before they are concretely synthesized.

The relevance of the method relies on the possibility to expedite new compounds discovery creating enormous scientific, humanitarian and economic benefits. Clearly, in this frame, the research in drug design and discovery is prominent. In fact, the development of pharmaceutics that safely and effectively treat diseases and disorders is a very time-consuming and expensive process. Various studies estimate that traditional development of a new prescription drug takes between 10 and 15 years and costs an average of $500-$800 million. The pressure to reduce costs and accelerate the drug discovery cycle is therefore high, especially considering the humanitarian value of the aim and the constant necessity to cope with new diseases. Besides the efficiency we can also consider the need
to improve the percentage of compounds with real therapeutic value and to reduce the side effects of the drugs.

The studies involve technology and problems from traditional methods and new tool for data analysis, including library design and virtual screening (or \textit{in silico} screening), Data Mining, molecular graphics and simulation technology.

Other disciplines, such as genomics, proteomics, and the application of predictive ADME-Tox (adsorption, distribution, metabolism, elimination, and toxicity) are clearly related to this field. In particular, the ADME-Tox analysis allows the study of optimizing efficacy, drug safety, and bio-availability of a lead compound. This should introduce an important manner to accelerate the steps toward the project of new drugs, allowing the researcher to test hypotheses before the molecules are synthesized. Toxicity prediction represents one of the most relevant examples, where the aim can range from the assessment of the potential genotoxicity, carcinogenicity, or other toxicity of pharmaceutical to anticipate adverse health effects. The task is considered very hard due to the broad range of potential biological targets and mechanisms.

The advantages of the computational methods extend to the possibility of treatment of such problem in the same frame. Specifically, we focus here one of the primary and earlier stage of the complex drug discovery process (drug lead identification) aimed at finding a relatively small compound (ligand) that binds with a receptor of certain proteins or enzymes.

**QSPR/QSAR** In Cheminformatics area, the major emphasis is given to methods aimed at predicting biological activity and other properties from the structure of the molecules. QSPR/QSAR analysis, i.e. Quantitative Structure-Property Relationship and Quantitative Structure-Activity Relationship, respectively, are paradigms useful in such context. They are especially useful when there is a known collection of molecules of known property or activity, while micro-mechanism of interaction between compounds or between active molecules and the bio-receptor that determined the properties of interest, are not known in advance or very difficult to devise.

Basically, these methods aim at developing a model of the structure- property or activity relationship from a collection of known samples. The learned model is then used to infer an estimation of the behavior of other similar compounds not yet measured. The studied properties range in a wide range of physiochemical measurable quantity, while the \textit{activity} in general measures how well the molecule (drug) candidate binds to its target and generates the desired biological response.

Specifically QSAR analysis plays an important role in the rational drug design process. As for any \textit{predictive} method the relevance of the method depends upon the amount of information that is available: in the drug design it corresponds to information about the ligand and receptor complex. In fact, although the ideal case corresponds to the possibility to study directly the interaction between ligand and receptor/protein drug target structure, the method is not always applicable due to the unavailability of the three-dimensional structures details. Moreover, the complete behavior of the ligand, including the pharma-
cokinetics of the drug, is not tackled by the “ideal” approach. Hence, predictive methods able to consider and to infer the effective activity of the molecules are very useful to enlarge the region of tractability of the problem.

During the last twenty years quantitative structure-property relationships (QSPR) and quantitative structure-activity relationships (QSAR) have been applied to wide range of problems gaining an extensive recognition in physical, organic, analytical, pharmaceutical and medicinal chemistry, biochemistry, chemical engineering and technology, toxicology, and environmental sciences. Examples of the wide range of predicted properties include melting and boiling temperature, molar heat capacity, vapor pressure, solubility, viscosity and partition coefficients, standard Gibbs energy of formation, refractive index, density, solvation free energy, receptor binding affinities, pharmacological activities, and enzyme inhibition constants.

Various analytical tools from statistics and machine learning are used in QSPR/QSAR analysis including predictive modeling (classification and regression), visualization, and exploratory data analysis through principal components and cluster analysis.

The fact that the domain of QSPR/QSAR problem is naturally composed by structured data (molecular graph), that there is a lack of a clear theory to explain the molecular interactions, and the presence of noisy (due to experimental error and approximation) and incomplete data make the ML for SD an approach ideally suited to tackle the problem. Instead, most of the traditional approaches, based on the method developed by Hansch since 60’, are based on a case-dependent extraction of features, guided by the knowledge experts, to obtain a set of suitable structural descriptors of the molecules, i.e. a flat representation of the problem domain. Hence, despite the success in specific applications, general methods able to cope with heterogeneous molecules and problems are emergent issues.

Before extending the explanation of the methodology in Part III of the thesis, we introduce an example in the following involving a QSPR problem for a class of simple organic compounds. The problem can be considered both a clear example of the study of the relationships between the structure of a small molecule and their properties, and a benchmark in the QSPR studies.

### 2.3.4 A Benchmark Problem: the QSPR Analysis of Alkanes

The set of alkanes (saturated hydrocarbons) molecules provides a concrete example of real objects belonging to a structural domain. It is interesting to note that, as historical fact, the mathematical notion of tree structure have been introduced by the mathematician A. Cayley in 1857 when he was trying to model and to enumerate alkanes compounds.

The study of the Quantitative Structure-Property Relationship for alkanes aimed at predicting the boiling point property, provides a simply and clear example of the application goal of the structure-transduction: given an alkane compound, we would like to predict the value of the boiling point temperature just looking at its chemical structure. Using a set of compounds for which we already know the boiling point values (training
set), the model learns to map the compound structure to the real value that measure the boiling point property.

The boiling temperature of alkanes is frequently used as a benchmark property in testing Quantitative Structure-Property Relationship (QSPR) models. Due both to its high relevance in the design of industrial process and to its well known characteristics, many methods have been developed for the prediction of boiling point of organic compounds since the field origins and it is a currently studied problem, e.g. [89]. For these reasons we can consider this QSPR problem in the following to test the proposed model on a real-world problem characterized by a SD. The presence of previous approaches allows us also to compare our method with “ad hoc” methods for the problem (PART III).

The data set that composes our benchmark, which is taken from [30], is based on all the 150 alkanes with up to 10 carbon atoms ($\text{C}_n\text{H}_{2n+2}$). It is well known that for this class of compounds, the prediction of the boiling point can be performed by disregarding the information about the hydrogen atoms. Hydrogens suppressed graphs of alkane molecules are trees. Carbon-hydrogens groups are associated with vertexes, and bonds between carbon atoms are represented by edges. In order to represent them as rooted positional trees, we used the standard conventions used in chemistry, i.e. the I.U.P.A.C. nomenclature rules [87]. The details of the definition of appropriate rules, for the specific set of molecules studied, are discussed in PART III (e.g. Sections 9.4 and Section 10.1.1 for alkanes).

An example of alkane representation is shown in Figure 2.3. The vertexes in the trees have a maximum out-degree of 3, maximum in-degree 1, and the maximum tree depth is 10. There is a total of 1331 vertexes in the data set.

The prediction of the boiling point yields to a regression task with a target associated

---

**Figure 2.3:** Example of rooted-tree representation for an alkane (3-ethyl-3-methylpentane).
to the root vertex of each tree. The target is the boiling point expressed in Celsius degrees (°C) into the range $[-164, 174]$.

The *alkanes* data set will be used throughout the dissertation both as *proof-of-the-principle* aimed at comparing models on a SD-problem (in PART II) and as a true application, with further details, aimed at comparing versus traditional approaches to the problem (in PART III).
Part II
Models
Chapter 3

A General Framework for the Recursive Processing of Structured Domains

In this chapter we review and introduce some definitions and notions which will be fundamental for our work. In essence, in Chapter 3 and 4, we present the methodology and the basic models that are the background of the research line in recursive neural networks as originally introduced in [155] and [52]. A survey of the approach is also recently appeared in [101].

In this presentation the basic abstract structure of the proposed approach is introduced via the definition of a functional graph transduction based on recursive functions. This set of definitions makes up the general framework for the recursive processing of structured domains. In particular, we will show (in Chapter 4) how neural models for the processing of structured data can be realized specializing the proposed general framework. These notions allow us also to uniformly describe innovative solutions to the applicative problems in the area of Computational Chemistry (PART III).

The chapter is divided into three parts: in the first one we recall the basic notion on graphs introducing our notations for SD. In the second one we introduce a class of functional graph transduction which can be computed recursively, defining the basic framework for the recursive models. In the third part we introduce the basic elements to exploit the transduction as a learning system.

Besides presenting the core of the hypothesis, i.e. the recursive approach to SD, we introduce in the present chapter the mathematical notation used in this thesis. There is also few other notations that will be used later in the thesis. That specific material, however, will be introduced only when needed.

3.1 Structured Domains

We start with some preliminary definitions that allow to uniformly treat all the considered data domains.

In this thesis we make an extensive usage of structured data, i.e. data that can be
4. A GENERAL FRAMEWORK FOR THE RECURSIVE PROCESSING OF SD

represented in the form of classes of graphs, especially referring to hierarchical structures. Due to the relative variety of notations in the area of discrete structures we recall (only) some basic definitions to fix the working condition. Basically, for the most part we refer to notions common in literature with only slightly differences and some specific notations. Also, slight different notations can be adopted and presented by need in the chapters, relating them to the notations introduced in this section.

The structured domains (SD) considered here are sets of either labeled ordered or positional directed acyclic graphs, DOAG and DPAG respectively. For a DOAG we mean a DAG (directed acyclic graph) where for each vertex a total order on the edges leaving from it is defined.

A path of length $p$ from a vertex $v$ to a vertex $u$ in $G$, denoted by $\text{path}(v, u)$, is a sequence of vertexes $v = v_0, v_1, ..., v_p = u$ such that $(v_{i-1}, v_i) \in \text{vert}(G)$ for $i = 1, ..., p$. A path $\text{path}(v, u)$ is a cycle if $v = u$. The out-degree and the in-degree of a vertex are, respectively, the number of edges leaving and entering it. Bounded out-degree and in-degree are assumed for the considered DAGs.

Labels are tuples of variables (or attributes): they are attached to each vertex. The set of labels for graphs $G \in \mathcal{G}$ is denoted by $\mathcal{L}(\text{vert}(G))$, and the set of labels of the domain by $\mathcal{L}(\mathcal{G}) = \bigcup_{G \in \mathcal{G}} \mathcal{L}(\text{vert}(G))$, or more simply $\mathcal{L}$ when the context is clear (or very general). $l(v)$ denotes the attribute vector of the vertex $v$. Unless differently specified, the attribute vectors are assumed to be numeric and with constant dimension $n$. For symbolic labels, with symbol from a finite alphabet $\Sigma$, a label coding function $\phi_L : \Sigma \rightarrow \mathbb{R}^n$ can be defined to encode the symbolic labels into numerical vectors. For the sake of presentation, symbols can be retained in the graphical representation of structures to label the vertexes in a concise way.

The skeleton of a structured data $G$ is the unlabeled version of the graph $G$, denoted as $\text{skel}(G)$. The skeleton of a graph retains the topology of the graph. Hence, the term structure, in this thesis, refers to (and is characterized by) the node labels $\mathcal{L}(\text{vert}(G))$ and their relationships, expressed in $\text{skel}(G)$.

The class of directed positional acyclic graphs (DPAG) is a superclass of DOAGs where a position is assigned to each edge leaving from each vertex, i.e. besides ordering, a distinctive positive integer can be associated to each edge, allowing some position to be absent. In particular, we assume that for each vertex $v \in \text{vert}(G)$, an injective function $\text{egd}(v) : [1, 2, ..., K]$ is defined on the edges leaving from $v$.

We shall require the DPAGs either to be empty or to possess a supersource, i.e. a vertex $s$, with zero in-degree, such that every vertex in the graph can be reached by a directed path starting from $s$. Note that if a DPAG does not possess a supersource, it is still possible to define a convention for adding an extra vertex $s$ (with a minimal number of outgoing edges), such that $s$ is a supersource for the expanded DPAG [155]. The assumed structured input domain, denoted by $\mathcal{G}$, is a set of labeled DPAGs with supersource and bounded out-degree and in-degree. The function $\text{source}(G)$, with $G \in \mathcal{G}$, returns the (unique) supersource of $G$. The symbol $\zeta$ denotes the void graph.
For a DAG, if an edge \((u, v)\) is present in \(\text{egd}(G)\), then \(u\) is a parent (predecessor) of \(v\) and \(v\) is a child (or successor) of \(u\). If there exists a path from \(u\) to \(v\) then \(u\) is an ancestor of \(v\) and \(v\) is a descendant of \(u\).

The set of successor of vertex \(v\) is sorted according to the order defined on the edges and more generally, for a DPAG, the successors are “positioned” according to the function \(S_v\) defined above. Fixed size vectors can represent the set of children of a vertex allowing the presence of null components for absent (or missing) children.

Vertexes with zero out-degree are external vertexes or leaves. A non-leaf vertex is an internal vertex. The set of external vertexes is called frontier. The set of children of a DPAG leaf is a vector of null components. It is possible to conventionally obtain a full DPAG, with internal vertexes with constant out-degree \(K\), from a DPAG with maximum out-degree \(K\), representing each missing child, i.e. the null component of the children vectors, with a special unlabeled external vertex.

Given \(s \in \text{vert}(G)\), the subgraph \(G(s)\) with supersource \(s\) of a DPAG \(G\) is the DPAG induced by the descendants of \(s\).

The principal class of graphs (sub-class of DPAGs) considered in the following, i.e. sequences and rooted trees, are described in more details as examples:

**Example 3.1 (Sequences)** The class of sequences, or lists, is a simple example of structured data that are frequently adopted to model problem domains. In sequences each internal vertex has only one edge entering and one edge leaving in/from it (out-degree = in-degree = 1). For sequences it is possible to define a total order on the vertexes. For instance, dealing with discrete time series the time stamps of the temporal order are associated to the vertexes of the linear chain using the integer variable \(t\). The supersource is defined by the last vertex of the serial order.

As a very special case, when the number of vertexes of the sequences in the class \(\mathcal{G}\) is 1, \(\mathcal{G}\) reduces to the class of vectors composed by \(\mathcal{L}\), i.e. the set of labels of the domain.

**Example 3.2 (Trees)** Free trees are a class of connected undirected graphs characterized by the absence of cycles. Rooted ordered trees are free trees with a distinctive vertex (or node) called the root. Direction on trees can be naturally assigned to edges following the path from the root to each node of the tree. In the case of trees, the supersource is always defined by its root node. We then define the level of a vertex: vertexes at distance \(l\) from the root lie at level \(l + 1\); the root lies at level 1. The height of the rooted tree is defined as its maximum level.

In positional trees the set of successors (children) of a node are associated with distinctive positional numbers that allow representing the set as a vector where some components can be null. Rooted positional trees are a subclass of DPAG, with supersource, formed by prohibiting cycles in the undirected version of the graph. Positional trees with bounded out-degree \(K\) are also called \(K\)-ary trees. For \(K = 1\), \(K\)-ary tree reduce to lists (sequences). In the following we often use the term tree referring to the class of labeled \(K\)-ary trees.

Given a node \(v\) in the tree \(T \in \mathcal{G}\), we give the following definitions:
• out\_deg(v) is the number of successors (children) of v; the maximum out-degree over G is the constant value K.

• ch[v] is the K-dim vector of children of v, and ch_j[v] is the j-th child of v, with respect to the assigned order on the outgoing edges; in positional trees some positions ch_j[v] can be empty (nil).

• l(v) ∈ ℝ^n is the input label associated with v, and l_i(v) is the i-th element of the label;

• T(r) is a tree rooted at node r. The subtree T(v) rooted at v is the tree induced by the descendants of v.

Traversal of the tree allows to systematically visit (and process) all the nodes of the tree in some order. Processing the root and then recursively processing all subtrees we define a preorder traversal; recursively processing all subtrees and finally the root we define a postorder traversal.

The parentheses representation of trees is a useful string representation of trees. The hierarchy is expressed in the nesting of parentheses, and the labels are expressed by symbols. Given a tree T(r) rooted in the node r, its parentheses representation p-rep[T(r)] can be recursively wrote as

\[
\begin{align*}
\text{p-rep}[\zeta] & = \text{nil} \quad \text{(a void string)}; \\
\text{p-rep}[T(r)] & = l(r)\text{p-rep}[T(ch_1(r)), \text{p-rep}[T(ch_2(r)], ..., \text{p-rep}[T(ch_k(r))] 
\end{align*}
\]

where l(r) is the symbolic label attached to the root. Equations 3.1 and 3.2 correspond to the preorder traversal on the tree. This notation can be practically exploited to prepare concise symbolic representations of a tree data set.

For DPAG we adopt a notation similar to the notation described above for trees, adapted to DPAG or subgraphs G(s) with supersource s. Basically we extend the notation of trees to DPAGs without repeating the shared concepts. For example we extend the notion of children and parents, leaf, etc. to DPAG, where the main difference in this context is due to the number of parents of a vertex (in-degree) that is strictly 1 in a tree (0 for the root) but it can be greater than 1 for DPAG. In literature some terms can be distinguished for trees and DPAGs, such as for instance the case of the terms node and arc used in this context as synonyms of vertex and edges introduced for DPAG.

A postorder traversal on DPAG is a traversal, analogous to the postorder traversal on trees, which follows an inverted topological order on the DPAG.

Further specific extensions of the notations for DPAG will be introduced in Chapter 6.

Finally, a useful overview of the classes (set) of labeled graph as defined above is shown in the following chain of inclusions:

\[
\text{Vectors } \subset \text{Sequences } \subset (K\text{-ary})\text{Trees } \subset \text{DPAGs} \tag{3.3}
\]
3.2 Recursive Structure Transductions

In this section we define a class of hypotheses that map from a structured domain \( \mathcal{G} \) (in the class of DPAGs, DOAGs, trees or lists) to discrete or continuous (output) space \( \mathcal{O} \). We will show throughout the dissertation how this hypothesis space is able to represent hierarchical relationships, and to perform an efficient handling of structure variability. This general framework is the basis to develop all the proposed models for processing and learning in structured domains.

On the basis of the class of data outlined in Section 3.1, and characterizing the class of functions to be learned as functional transductions, we can characterize the class of processing systems proposed in the thesis according to the following general definition:

**Definition 3.1 (SD-Recursive processing system)** A SD-Recursive processing system is a pair \((\mathcal{G}, \mathcal{T}_\mathcal{G})\), where \( \mathcal{G} \) is the domain of labeled DPAG with supersource and \( \mathcal{T}_\mathcal{G} \) is a functional graph transduction.

We consider in our framework the class of functions that can be characterized as the class of functional graph transductions \( \mathcal{T}_\mathcal{G} : \mathcal{G} \rightarrow \mathcal{O} \) which can be represented in the following form

\[
\mathcal{T}_\mathcal{G} = g \circ \tau_E, \tag{3.4}
\]

where \( \tau_E : \mathcal{G} \rightarrow \mathcal{X} \) is the encoding function and \( g : \mathcal{X} \rightarrow \mathcal{O} \) is the output function. To encode a structured data, specifically a labeled DPAG \( G(s) \in \mathcal{G} \) with supersource \( s \), we use the following recursive definition of \( \tau_E \):

\[
\tau_E(G(s)) = \begin{cases} 
nil & \text{if } G = \zeta \\
\tau(l(s), \tau_E(G(ch_1[s])), \ldots, \tau_E(G(ch_K[s]))) & \text{otherwise}
\end{cases} \tag{3.5}
\]

where \( \tau \) is defined as

\[
\tau : \mathcal{L} \times \underbrace{\mathcal{X} \times \cdots \times \mathcal{X}}_{K \text{ times}} \rightarrow \mathcal{X} \tag{3.6}
\]

where \( \mathcal{L} \) denotes the label space, \( \mathcal{X} \) denotes the encoded subgraphs spaces (or simply code space), and it can be a discrete or continuous space, \( K \) is the maximum out-degree of the input domain \( \mathcal{G} \), \( l(r) \) is the label attached to the supersource of \( G \), and \( G(ch_1[s]), \ldots, G(ch_K[s]) \) are the \( K \) subgraphs of \( G(s) \) with supersource in the children of \( s \).

Note that, because of Equation 3.5, \( \mathcal{T}_\mathcal{G} \) is causal since \( \tau \) only depends on the current vertex and vertexes descending by it. Moreover, when \( \tau \) does not depend on any specific vertex, as in Equation 3.5, then \( \mathcal{T}_\mathcal{G} \) is also stationary; in this work we focus only on stationary transductions.

The recursive definition of \( \tau_E \) determines a systematic visit (postorder traversal) of the input DPAG. The computation of \( \tau_E \) is a progressive process which starts from the frontier
of the input DPAG and terminates at the supersource of the DPAG, where a numerical code for the whole structure is generated. Following a reverse topological sort in the DPAG, a code is generated step by step for each visited vertex and stored as state information for each corresponding vertex. For each currently visited vertex its numerical label and the codes already computed for its descendants, e.g., children in a tree, (stored in the state), are used to compute the code for the current vertex. Note that for frontier vertexes the process starts with a null state because there is no previous information from descendants. Also note that any different order in the traversal that respects the topological order is valid and produces the same final code.

We define the state $x(v) \in X$ associated with each vertex $v$ of the structured data $G$, as

$$x(v) = \tau_E(G(v))$$  \hspace{1cm} (3.7)

Thus, we can rewrite Equation 3.5 as a state transition system according to the following equations

$$x(v) = \tau(l(v), x(ch_1[v]), \ldots, x(ch_K[v]))$$
$$y(v) = g(x(v))$$  \hspace{1cm} (3.8)

where $\tau$ is the state transition function, $g$ is the output function, $y(v)$ is the output vector, and $x(ch_1[v]), \ldots, x(ch_K[v])$ is the $K$-dim vector of states for the vertexes children of $v$. The complete encoding $x(s)$ of a structured data $G$ is obtained at the vertex $s = source(G)$ (e.g., the root of a tree). The bottom-up (frontier-to-root for trees) process starts with null state $x_0$, typically coded by 0 values. Moreover we define $x(nil) = x_0$.

**Example 3.3 (Finite Automata)** Mealy and Moore machines (hence also Deterministic Finite State Automata) can be easily described, as special cases, in the present framework, limiting the input domain $G$ to strings and $X$ to be a finite space. Specifically, define a Mealy/Moore machine in our framework, where

- $X$ is a finite set of states;
- $L$ is a finite input alphabet;
- $O$ is a finite output alphabet;
- $\tau : L \times X \rightarrow X$ is the next-state function;
- $g$ is the output function, that in the case of a Mealy machine is $g : L \times X \rightarrow O$, and in the case of a Moore machine is $g : X \rightarrow O$;
- $x_0$ is the initial state from the set $X$. 
Extension of the input domain to trees can be faced by Tree automata, e.g. the frontier
to-root tree automaton (FRA) (or bottom-up tree automaton).

In general, we can describe also state machines (and we mainly deal with them in the
following) that are not restricted to be finite assuming $\mathcal{X}$ to be a continuous space, i.e.
realized by real values.

The class of transductions between structured spaces characterizes a very large set of
possible relations. In the following we restrict our attention to specific class of functional
transduction for which it is reasonably easy to build adaptive models.

**Supersource transductions** We first consider the case where the output space of the
transduction is unstructured (or flat). Specifically in *supersource transduction* the re-
sponse of the SD-recursive processing system is given when the encoding process termi-
nates, i.e. in correspondence of the supersource of the input structure. The response is
given in the form of a scalar value.

In terms of Equations 3.8 and 3.9 a supersource transduction is recursively defined as

$$x(v) = \tau(l(v), x(ch_1[v]), \ldots, x(ch_K[v]))$$

and

$$y = g(x(s))$$

where $s = source(G)$.

**IO-isomorphic transductions** The system specified in Equations 3.8 and 3.9 allows to
produce (emit) an output vector of values for each vertex of a given graph $G \in \mathcal{G}$. In
such cases, the defined transductions can be characterized by the *IO-isomorphic* (Input-
Output-isomorphic) property [155]: if we consider the transduction results as character-
ized by a structured shape, this structure has the same skeleton of the current transducted
input graph. The input and the output are both structured and they share the same skele-
ton. Frequently the IO-isomorphic concept is encountered for sequential transduction, i.e.
transduction applied to sequences: in such case the transduction is termed synchronous,
whenever it is required an output for each input step, i.e. to generate an output sequence
of the same length of the input sequences.

It is worth to observe that under the specific hypothesis of stationarity and causality
stated in Section 3.2, the IO-isomorphic transductions can be seen as a subclass of the
supersource transductions. In fact, we can obtain an output value for each vertex of graph
$G$ considering a supersource transduction applied to each subgraph $G(v)$. In such case,
the graph $G$ is viewed as the set of its subgraph $G(v)$ with $v \in \text{vert}(G)$. The supersource
transductions applied to each $G(v)$ can compute the same value emitted for each $v$ by the
IO-isomorphic transduction applied to $G$. 
### 3.2.1 Generalized Shift Operator and Graphical Representations of \( T_g \)

Graphical representations of \( T_g \) can be obtained using a symbolic transformation of the state variables called the \textit{generalized shift operator} \( q \). This operator will be used also in the following to obtain compact forms for the realizations of the models \( \tau \) and, used together with the description of transductions in Equation 3.8, it allows to describe architectural details of the new models.

For a temporal variable \( x(t) \) the unitary time delay is expressed by \( q^{-1} \), defined as \( q^{-1}x(t) = x(t - 1) \), i.e. the system looks one step backward in the serial order of the temporal values. Multiple delays, e.g. \( n \), can be represented by the power of the operator \( q \), obtained by the composition of \( n \) shift operators \( q^{-1} \), where \( q^n x(t) = x(t) \). Moreover, the shift operator can be inverted, considering positive exponents, to express non causal dependencies, i.e. to consider forward steps in the serial order. Therefore, for sequences the operator \( q \) allows to express shifts in the serial order of nodes.

To deal with graphs we generalize the shift operator to consider the set of children of each vertex of the structure. Specifically, given a state vector \( \mathbf{x}(v) \equiv [x_1(v), \ldots, x_m(v)]^t \), we define extended shift operators as follows:

**Definition 3.2 (Generalized shift operator)** If \( \text{ch}_j[v] = \text{nil} \) then \( q_j^{-1}x_i(v) \equiv x_0 \), and \( q_j^{-1}x_i(v) \equiv x_i(\text{ch}_j[v]) \) otherwise.

Hence, the delay expressed by \( q \) is extended to the graph topology, and it can be further extended to deal with vectorial notation, according to the following definitions:

\[
q^{-1}x_i(v) = [q_1^{-1}x_i(v), \ldots, q_K^{-1}x_i(v)], \quad (3.12)
\]

\[
q_j^{-1}\mathbf{x}(v) = [q_j^{-1}x_1(v), \ldots, q_j^{-1}x_m(v)] \quad (3.13)
\]

and,

\[
q^{-1}\mathbf{x}(v) = [q_1^{-1}\mathbf{x}(v), \ldots, q_K^{-1}\mathbf{x}(v)]. \quad (3.14)
\]

On the basis of these definitions, we can compactly describe Equation 3.8, using the generalized shift operator, as

\[
\mathbf{x}(v) = \tau(\mathbf{l}(v), q^{-1}\mathbf{x}(v)) \quad (3.15)
\]

Moreover, we can give a graphical and general representation of the function \( \tau \) (Figure 3.1).

Figure 3.1 allows to express the recurrent nature of the \( \tau_E \) function in a graphical form combining feedback connections and the delay operator \( q \). The different ways to connect and to combine the feedback information give rise to different type of models representable as different realizations of \( \tau \).
3.3. LEARNING SYSTEMS

The main aim of the thesis is to deal with Learning Systems, i.e. systems able to realize inference of data models from set of examples, as outlined in Chapter 2. Hence, in the
A General Framework for the Recursive Processing of SD

Figure 3.2: Unrolling of the encoding process for a sequence (left side) and for a tree (right side). For each input structure an encoding network is generated by replicating the recursive model \( \tau \) for each vertex of the structured data. It is shown the code \( x_\alpha \) computed for each node, where \( x_\alpha \) is a short hand for the code \( x(v) \) of the subtree rooted in the node \( v \) with label \( \alpha \) in the example of structured data shown, and \( x_0 \) is the initial state. The labels, here represented as symbols, are supposed to be encoded into suitable numerical vectors \( l_\alpha \) (through a label coding function \( \phi_L \)).

Following, we first need to extend to structured domains the concepts of data sets and target function, and then to introduce the equipment that can characterize the SD-recursive processing system as learning tool.

As mentioned in Section 2.1, for supervised computational tasks we consider regression or classification tasks. In regression tasks for structured domains we mainly consider target functions \( f() \) over \( \mathcal{G} \) defined as \( f : \mathcal{G} \rightarrow \mathbb{R}^z \) where \( z \) is the output dimension, while in classification tasks we have \( f : \mathcal{G} \rightarrow \{0, 1\}^z \) (or \( f : \mathcal{G} \rightarrow \{-1, 1\}^z \)). Binary classification is specified by \( z = 1 \). For such tasks, a training set \( \mathcal{D} \), or more generally a data set \( \mathcal{D} \), on a domain \( \mathcal{G} \), is defined as a set of couples \((G, f(G))\), where \( G \in \mathcal{G} \) and \( f(G) \) is the target value associated to the instance \( G \) of data.

In unsupervised learning tasks the training set \( \mathcal{D} \) is defined as a set of instances \( G \) belonging to \( \mathcal{G} \).

The function \( T_{\mathcal{G}} \) as defined in Section 3.2 can be used to map from a structured domain \( \mathcal{G} \) to an output domain \( \mathcal{O} \), that can be discrete or continuous, and therefore can be used in approximation tasks. We consider in particular two different cases:

### Supersource transductions

When dealing with the prediction of scalar quantity as a function of the input structure, we use supersource transductions. In fact, the output of the function \( T_{\mathcal{G}} \) is meaningful, and can be compared with the desired target values \( f(G) \), when the whole input graph \( G \) has been encoded, i.e. when the supersource of \( G \) has been processed. Most of the tasks presented in the application part can be described in the supersource transduction framework.
**IO-isomorphic transductions** IO-isomorphic transductions characterize systems that produce an output vector of values for each vertex of a given graph $G \in \mathcal{G}$. In such cases we can consider tasks where the target values are associated to each vertex of the input structure, i.e. the target function $f$ is defined for each vertex of the graphs $(f(v), \ v \in \text{vert}(G), \ G \in \mathcal{G})$.

The basic idea of the approach is to make adaptive the mapping performed by $\mathcal{T}_G$ incorporating free parameters in the model that realizes $\mathcal{T}_G$ and using a learning algorithm that allows constructing an hypothesis that fits the data adapting the parameters to the task at hand.

Specifically, the functions that compose $\mathcal{T}_G$ will be assumed to be dependent on trainable parameters. In particular we use the following definitions for characterizing the learning models in structured domains.

**Definition 3.3 (Parametric SD-Recursive processing system)** A SD-Recursive processing system $(\mathcal{G}, \mathcal{T}_G)$ is parametric if $\mathcal{T}_G$ is a function of tunable parameters $W$.

For the sake of simplicity, we can omit in the notation the explicit dependences of $\mathcal{T}_G$ from parameters $W$ and we can assume that dependencies in the following. The special case of the absence of free parameters, can be easily described in this framework by a set of $W$ constant or by an empty set $\emptyset$.

A realization of a Parametric SD-Recursive processing system can be achieved choosing parametrized functions $\tau$ and $g$ for the state transition function and for the output function (see Equation 3.5, or 3.8 and 3.9).

Learning can be seen as parameter optimization with respect to an error measure or risk function computed for the given model on the given task and data. A learning algorithm $\mathcal{A}$ is used to estimate the model parameters $W$, i.e. to minimize the risk function tuning the model parameters for the training data $\mathcal{D}$. Hence, we can state the following definition:

**Definition 3.4 (Adaptive SD-Recursive processing system)** An Adaptive SD-Recursive processing system is a triple $(\mathcal{G}, \mathcal{T}_G, \mathcal{A}_W)$ where $(\mathcal{G}, \mathcal{T}_G)$ is a Parametric SD-Recursive processing system and $\mathcal{A}$ represents the learning algorithm used to fit the parameters $W$ to the data in the set $\mathcal{D}$ defined on $\mathcal{G}$.

We have so far outlined the class of hypotheses used to deal with structured domains, defining the form as a SD-Recursive processing system and introducing parameters in the transduction function to adapt the behavior of the system according to the given task and data. In terms of the notation in Section 2.1 we have delineated the following hypothesis space:

$$\mathcal{H} = \{ h \mid h(G) \equiv \mathcal{T}_G(G), \ G \in \mathcal{G} \} \quad (3.16)$$
The main advantage of the approach we propose is strictly related to the learning capability of the models that realize the processing system. In fact, by using the proposed adaptive processing scheme, the specific encoding procedure is learned on the basis of the training data. Specifically, the proposed approach allows an adaptive processing of structured domains since it allows the extraction of the topological information of structures directly from the examples of the computational problem at hand.

Note that the graphical representation of the function $\tau$ can be used also to represent parametrized $\tau$ functions, for instance, by associating the arrows in Figure 3.1 to the parameters. Different models can be specified for various possible realizations of the parametrized $\tau$. In the following chapters we present neural realizations of Adaptive SD-Recursive processing systems, based also on different risk functions, that allow a flexible and robustness implementation of a learning system for structured domains.
Chapter 4

Neural Realizations in Supervised Learning: Recursive Neural Networks

For supervised learning tasks in SD, Neural Structure Transductions are obtained realizing the function $\tau$ and $g$ of the framework introduced in Section 3.2 with connectionist models.

Recursive Neural Networks (RNN) are *Adaptive SD-Recursive processing systems* $(G, T_G, A_W)$, where the set of free-parameters $W$ are realized by the weights of the connectionist models $\Phi(\cdot, W)$, and the functions that compose $T_G$ (Section 3.2) are instantiated, in the cases of our interest, according to the following schema of definitions:

- $X = IR^m$ is a continuous state space, named *code space*, corresponding to the output domain of the $m$ state neural units of the RNN;

- $L = IR^n$ denotes the (input) label space, i.e. the labels are real valued vectors;

- $O = IR^z$, or $O = \{0, 1\}^z$ (or equivalently $O = \{-1, 1\}^z$), is, respectively, the continuous or discrete output space, with $z$ the number of output units of the RNN;

- $\tau : IR^m \times IR^m \times \cdots \times IR^m \rightarrow IR^m$

  is the transition function realized by a neural network that computes a new state $x(v)$ for state units of the network from the currently available input $(I(v))$ and the previous states $x(ch_1[v]), \ldots, x(ch_K[v])$, according to the input graph topology, and $K$ is the maximum out-degree of the input domain $G$;

- $g : IR^m \rightarrow O$ is the output function realized by a neural network;

- $x_0 \in IR^m$, the initial state, is assumed to be a null vector in the following ($nil = 0$).
A synthesis of the functions and domains for Neural Structure Transductions is given in the following schema:

\[ G \xrightarrow{\tau_E} \mathbb{R}^m \xrightarrow{g} \mathbb{R}^z \]  
(4.1)

where a RNN realizes \( G = g \circ \tau_E \) approximating the functions \( \tau \) and \( g \) by neural networks, i.e. two functions \( \Phi(\cdot, W) \) (Section 2.2). Different instances of the function \( \Phi() \) can be used to implement different realizations of the \( \tau \) and \( g \) functions. The main classes of our interest are described in the following using this framework, i.e. describing each model according to the functions \( \tau_E, \tau \) and \( g \).

We have defined the function \( g \) as a map \( g : \mathbb{R}^m \rightarrow \mathbb{R}^z \), where typically \( z = 1 \) for regression tasks or as a map \( g : \mathbb{R}^m \rightarrow \{0,1\}^z \) or \( g : \mathbb{R}^m \rightarrow \{-1,1\}^z \) for classification tasks. In order to realize the function \( g \) it is possible to choose any known mathematical model. In the neural networks area the function \( g \) may be realized using simple neural model, e.g. linear output neurons to realize a regression model: \( g(x) = m^T x + \beta \), where \( m \in \mathbb{R}^m \) and \( \beta \in \mathbb{R} \) is the output threshold, or using a multilayer network to perform regression or classification tasks.

Examples of different realizations which satisfy the above equations for \( \tau_E \) and \( \tau \) are given in the following. Specifically, we briefly review the recurrent neural networks and a general realization of recursive neural networks with one fully-connected hidden layer. Such models are illustrative of the wide class of possible neural network architectures (a review of various different dynamics for discrete time recurrent neural networks can be found in [161, 110]). However, we do not lose generality, since it is proven that most of the models can at least be simulated or approximated within the simple Elman-dynamic used in the following [73]. Historical remarks on the introduction of RNN can be found in [51]. Finally a fully description of a constructive model based on the Cascade Correlation family (Recursive Cascade Correlation) is presented. Other novel proposed realizations are introduced in the following chapters.

## 4.1 Recurrent Neural Networks

Recurrent framework can be specified as an Adaptive SD-Recursive processing system \((S, G, A_W)\), where \( S \) is a set of sequences as characterized in Example 3.1 (in Section 3.1).

Recurrent neural network models are a specific case of the presented framework. Elman 1st-order recurrent networks [42] can be described as an instance of the recursive model in Equations 3.8 and 3.9. According to Example 3.1, the out-degree and in-degree are 1 for internal vertexes of a sequence \((K = 1)\) and the unique successor of a vertex \( v \) with time index \( t \) (for time series), i.e. \( \text{ch}[v] \), can be identified by the index \( t - 1 \). Hence, Equation 3.8 can be reduced to standard state-transition equations of recurrent neural networks:

\[ x(t) = \tau(l(t), x(t-1)) = \tau(l(t), q^{-1} x(t)) \]  
(4.2)
4.1. RECURRENT NEURAL NETWORKS

where $\tau$ can be realized by a set of $m$ hidden recurrent units.

A recurrent neural network distinguishes itself from a feedforward network by having feedback loop connection in its topology, i.e. a weighted version of the output is fed back into the input. In presented framework, such feedbacks are expressed by the shift operator $q$. Different use of the operator $q$ yields different architecture layout and models. In fact, feedback may occur at unit level (local type) or in different parts of the architecture (global type), i.e. intra or between network layers. A uniform overview of the various models and learning algorithms for recurrent neural networks can be found in [161] and [101].

The presence of feedback has profound impact on the learning capability of the network. It provides the neural model with dynamic properties, by the use of contextual internal states (memory). This makes the network responsive to the temporal structure of the input signal. The recurrent neural model opens the connectionist models to a broad set of temporal processing applications, such as the cognition tasks of vision, speech and signal processing, the modeling of system control and digital filter and the time series prediction tasks.

Due to the causality assumption the recurrent neural network are able to store in an internal state past information from the sequence of inputs and to use it together with the current input. The internal state of the recurrent neural network encodes a representation sensible to the context (e.g. temporal-context for temporal signal processing). Moreover, due to the learning capabilities, the memory of a recurrent neural network is a dynamic memory that is dependent from the task. The state-units discover adaptive abstract representation of past data containing the information relevant to the prediction.

The learning algorithm must be adapted to face this contextual nature of the approximation task, in this case a map from set of sequences to output values. It must account the set of encode transitions developed by the model for each step of the inputs. BPTT (back-propagation through time) [37] and RTRL (real time recurrent learning) [173] supervised learning algorithms are designed for recurrent neural networks. They compute, in different style, (the same) gradient values of the output errors across an unfolded over time network that is equivalent to the recurrent one (encoding networks).

Here, our interest resides in this model for historical reasons: the recurrent model represents the first step toward a SD processing with connectionist model, as it allows to extend the representational power of neural networks to sequence of input patterns in the sense explained in Section 2.3. Basically it contrasts with traditional methods that use a static map of the input sequence in order to adapt them to the vectorial nature of the input pattern. A typical solution made use of spatial representation of time employing a buffer that stores a finite set (window) of sequence vertexes, e.g. the class of FIR filter models and of IDNN (input delay neural networks). This raises the model with a limited form of temporal processing: processing of variable-dimension sequence is constrained by the use of finite memory. Two drawbacks are relevant: the size of the input buffer must be chosen a priori; and, there is a linearity relationship between the number of units of the model and the memory capacity. Hence, when more memory size is required the system must be redesigned. Moreover, the dimension of input buffer must grow with the number of model parameters. So, large input memory is paid in terms of efficiency and complexity.
of the model. As seen in Section 2.1 this may lead to low generalization performance (overfitting problem). In contrast the recurrent model can in principle memorize variable period of the sequence. Moreover the use of numeric values to code the features of the set of past inputs allows a non-linear memory capacity in the number of network units.

The recursive neural network is the generalization of the recurrent model to deal with more complex structures, e.g. trees and DOAGs. Hence, recurrent neural networks can be implicitly included in the following presentation of recursive networks.

4.2 Recursive Neural Networks

The use of a stationary and causal model for allows to choose a uniform and quite simple neural realization for through the definition of a recursive neural network model. Let consider, for example, a realization for that use a recursive neural network with hidden neurons, i.e. a fully connected recursive neural network with one hidden layer. The output of the hidden units for the current vertex , is computed as follows:

\[ x(v) = \tau(l(v), q^{-1}x(v)) = \sigma(Wl(v)) + \sum_{j=1}^{K} W^{j} q_{j}^{-1}x(v) \]  \hspace{1cm} (4.3)

where \( \sigma(u) = \sigma(u_i) \) (sigmoidal function), \( l(v) \in \mathbb{R}^{m} \) is a label, \( W \in \mathbb{R}^{m \times m} \) is the weight matrix associated with the label space, \( K \) is the maximum out-degree in \( G \), \( q_{j}^{-1}x(v) \in \mathbb{R}^{m} \) are the vectorial codes obtained by the application of the encoding function \( \tau_{E} \) to the subgraphs of \( v \), and \( W^{j} \in \mathbb{R}^{m \times m} \) is the weight matrix associated with the jth subgraph space. Note that the bias vector is included in the weight matrix \( W \) (see Section 2.2). The graphical representation of \( \tau \) in Figure 3.1 can be seen as a single recursive neuron or a more complex recursive neural networks depending on the values of \( m \).

At this level, the realization of the function \( \tau \) can be expressed by looking inside the \( \tau \) box of Figure 3.1. An example of realization of \( \tau \) with \( m = 1 \), i.e. a single recursive neuron unit, is graphically shown in Figure 4.1. Specifically this is the simplest non-linear neural realization for \( \tau \) according to the following instance of Equation 4.3 for \( m = 1 \):

\[ x(v) = \sigma(\text{net}(v)) \]  \hspace{1cm} (4.4)

\[ \text{net}(v) = \sum_{i=0}^{n} w_{i} l_{i}(v) + \sum_{j=1}^{K} \hat{w}^{j} q_{j}^{-1}x(v) \]  \hspace{1cm} (4.5)

where \( w_{i} \) are the weights associated to the label space, \( \hat{w}^{j} \) are the weights associated to the subgraphs spaces. Note that the vector \( q^{-1}x = q_{1}^{-1}x, \ldots, q_{K}^{-1}x \) in Figure 4.1 (where the dependences on the current vertex \( v \) are omitted for short) can be considered an extension of the inputs to the standard neuron (Section 2.2) that store the information from previous outputs of the model. The extended inputs represents “context” information about the subgraphs of the current processed input vertex. The current information is expressed by the label field \( l \) of the vertex. The weights \( \hat{w}^{j}, j = 1, ..., K \) are specific of the recursive
4.2. RECURSIVE NEURAL NETWORKS

Figure 4.1: A graphical representation of a recursive neuron, i.e. a simple neural realization of the function $\tau$ with $m = 1$.

Figure 4.2: Architectural elements in a Recursive Neural Network (RNN), i.e. an example of neural realization of the function $\tau$ with $m = 3$ and $K = 2$.

neuron (with respect to the standard one) and they are the free parameters associated to the extended inputs.

An example of neural realization of $\tau$ that involve $m = 3$ fully connected units in a single layer is shown in Figure 4.2. Several extension can be addresses considering different architectures of the neural network that realize $\tau$, e.g. multilayers networks, or different factorizations of $\varphi$. However in the following we mainly deal with the basic models (with universal approximation capability [73] as it will be shown in Section 5), while the architectural engineering of the recursive neural network is out of the aims of this thesis. Note that Elman style Recurrent Neural Networks can be obtained as a specific case imposing $K = 1$.

Examples of encoding networks can be easily included in the schema presented in Section 3.2 once the function $\tau$ is realized by the model represented in Figure 4.2. For the sake of clarity we can also make explicit the associations of the connections of the
Figure 4.3: Encoding network for a generic RNN and two instances of input structure (see caption of Figure 3.2 for details). The parameters $W$ of the neural realization of $\tau$ are explicitly shown associating them to the network connections.

graphical model $\tau$ with the parameters of the RNN, as we show in Figure 4.3. This representation allows us to show that the weights of the recursive units (parameters $W$ of the Adaptive SD-Recursive processing system) are repeated according to the topology of the considered input structure.

The network shown in Figure 4.3, given the input structure $G$, can be regarded as a (multi-layer) feedforward neural network with a topology constrained to the topology of the graph $G$ and that behaves the same as the RNN on $G$. The tuning of parameters of $\tau$ and $g$ performed through the training algorithm $A_{W}$ allows to adapt the computation result of the model according to the task. Looking to the encoding network is also useful considering the training strategy. Since each encoding network is a feedforward neural network, the training can be performed using the standard back-propagation class of algorithms (Section 2.2) with slight modifications due to the presence of multi-instances of the same weights through the network (weight sharing): the error signal is first computed for the output units and it is back-propagated through the encoding network. The individual gradient contributions of each weight are computed through the encoding network. The gradient contributions of corresponding copies of the same weight are collected for each structure. The total amount is then used to change all the copies of the same weight. The weights can than be updated according to batch or on-line upgrade strategy. The BPTS (Back-Propagation Through Structure) is based on this idea.

Actually, several supervised learning algorithms can be extended to train recursive neural networks. The Back-Propagation Through Structure and RTRL (Real Time-Real Time) are gradient descendant technique based algorithms [155] that extend the BPTT and RTRL algorithms developed for recurrent neural networks.

---

1Other examples will be provided in Section 9.3.1 showing more networks details and using an input instances composed by chemical structures.
4.3 Recursive Cascade Correlation

As introduced in Section 2.2, the problem of designing optimal neural network architecture for a given task is still theoretically unsolved. Moreover, the results strongly depend on the choice made using an expensive trial and error approach. For these reasons, we present in the following a specific neural realization of the functions $\tau$ and $g$ based on constructive algorithms, i.e., algorithms that build a network starting with a minimal configuration and adds new units and connections during training.

Recursive Cascade Correlation (RCC) [155] [154] is the extension of Cascade Correlation algorithms [47] [46] to deal with structured data. Cascade Correlation class of algorithms supply a method to solve the problem to establish a number of hidden units, that is typically found when dealing with fixed size architecture. RCC creates a recursive neural network using an incremental approach for the classification or regression tasks.

The main advantage of the RCC approach can be summarized in the following topics:

- automatic determination of network dimension and topology: RCC allows to deal with hypothesis spaces of flexible size, since the number of recursive hidden units is decided by the learning algorithm;
- training of a single unit for each step.

Specifically, this incremental learning approach has been found particularly useful in application settings (e.g. [21]) when no “a priori” information is known on the under analysis problem’s complexity. For this reason, most of the studies conducted in the following, both for analysis and applications aims, are based on the RCC class of models.

Specifically, we consider the following instance of Equation 3.8:

\[
x_1(v) = \tau_1(l(v), q^{-1}x_1(v)) \\
x_2(v) = \tau_2(l(v), q^{-1}[x_1(v), x_2(v)]) \\
\vdots \\
x_m(v) = \tau_m(l(v), q^{-1}[x_1(v), x_2(v), \ldots, x_m(v)])
\]  

(4.6)

where $x_i(v)$ is the $i$-th component of $\mathbf{x}(v)$, i.e., the output of the $i$-th hidden unit in the network. In this model, direct connections between hidden units are disregarded.

Each unit, i.e., the function $\tau$, is realized by a recursive neuron: the output $x_h$ of the $h$-th hidden unit over the current vertex $v$ can be computed as

\[
x_h(v) = \sigma(\text{net}_h(v)) \\
\text{net}_h(v) = \sum_{i=0}^{n} w_{hi} l_i(v) + \sum_{i=1}^{h} \sum_{j=1}^{K} \hat{w}_{ji} q_j^{-1} x_i(v)
\]  

(4.7)

where $\sigma(\cdot)$ is a sigmoidal function, $w_{hi}$ is the weight of the $i$-th input element to the $h$-th hidden unit, $\hat{w}_{ji}$ is the weight of the edge connecting the $i$-th unit to the $h$-th current unit, which brings the encoded information of the $j$-th child of the current input vertex.
Figure 4.4: Architectural elements in a Recursive Cascade Correlation (RCC), i.e. a realization of the function $\tau$ with $m = 2$ and $K = 2$.

Figure 4.5: The evolution of a RCC network with $K = 2$.

An example of a network implementing the above equations for the case $m = 2$ and $K = 2$ is illustrated in Figure 4.4 as an internal realization of the function $\tau$: for graphical convenience the (recursive) inputs vectors $q^{-1}x$ are grouped in vectors $q^{-1}x_h$ for each unit $h$.

The evolution of a network, obtained by adding a new hidden recursive neuron at each main iteration of the learning algorithm, is shown in Figure 4.5, where the output neural units that realize the function $g$ is also shown. Note that the output of the network is meaningful only when the activity of the hidden units represents the code for a complete structure or for a targeted vertex.

In the RCC for structures model applied to regression problem, to realize the function $g$, we use a single standard linear output neuron. The present formulation of the model is base on a reduced version of the architecture that can be considered. For instance, in order to reduce the number of parameters, the direct (non-recursive) connections between hidden units are not allowed in the model (see Section 5.4). Note also that the removed connections are not present in the class of RNN presented so far.
Figure 4.6: The encoding network (on the right side) obtained by replicating (unfolding) two recursive hidden neurons of RCC for each vertex in the input tree (on the left side). The black squares represent void pointers which are encoded as null vectors (in this case, the void pointer is equal to 0). The labels are associated to each vertex in the structure (L1, L2 and L3). The output of the encoding network is the code computed for the input structure.

The unfolding process can be shown also for RCC models, associating input data with the recursive neurons of RCC. An example that involves two units is in Figure 4.6.

**Learning in RCC** In the following we discuss how a neural graph transduction $\mathcal{T}_G$ can be learned using an extension of the Cascade Correlation algorithm [47]. The RCC learning algorithm creates a neural network using an incremental approach for the classification (or regression) of structured patterns. At the beginning, it is assumed that the starting network $\mathcal{N}_0$ is a network without hidden units and one output-layer yielding constant null outputs for each input pattern. In the first network $\mathcal{N}_1$ a hidden recursive unit $x_1$ is added such that the correlation between the output of the unit and the residual error of network $\mathcal{N}_0$ is maximized. Typically, since the maximization of the correlation is obtained using a gradient ascent technique on a surface with several maxima, a pool of hidden units is trained and the best one selected. The weights of $x_1$ are frozen (they cannot be retrained in the next steps) and the remaining weights (output layer) are retrained. If the obtained network $\mathcal{N}_1$ cannot solve the problem, new hidden recursive units are progressively added which are connected with all the inputs and previously installed hidden units. The process continues until the residual errors of the output layer satisfy a specified stopping criteria (e.g the errors are below a given threshold). Hence, the described method dynamically builds up a (not-fully connected) recursive neural network and terminates once a sufficient number of hidden units has been found to solve the given problem. The resulting network
is a cascade of units (see Figure 4.5).

Summarizing, learning is performed as in the Cascade Correlation model family according to the recursive nature of the hidden units. Specifically, the algorithm works interleaving the minimization of the total error function (LMS), e.g., by a simple back-propagation training of the output layer, and the maximization of the (non-normalized) correlation, i.e. the covariance, of the new inserted hidden unit with the residual error:

\[
S = \sum_u \left| \sum_v (x_h(v) - \bar{x}_h)(E_u(v) - \bar{E}_u) \right|
\]  

(4.8)

where \(u\) spans over the output units, \(v\) spans over the vertexes of all input structures for which a target is defined, \(\bar{x}_h\) is the mean output of the current unit, \(E_u(v)\) is the residual error of the output unit \(u\) for vertex \(v\), and \(\bar{E}_u\) is the mean residual error of the output unit \(u\).

The gradient of \(S\) is calculated for a generic weight \(w_{hi}\) (including the type \(\hat{w}_{hi}^j\)) deriving equation 4.8 with respect to the desired weight:

\[
\frac{\partial S}{\partial w_{hi}} = \sum_u sgn_u \sum_v (E_u(v) - \bar{E}_u) \frac{\partial x_h(v)}{\partial w_{hi}}
\]  

(4.9)

where \(sgn_u\) is the sign of the correlation between the output of the current hidden unit and the residual error of the output unit \(u\).

Applying the RTRL algorithm approach as described in [172] we can determine the derivative of the output of the current hidden unit as follows:

\[
\frac{\partial x_h(v)}{\partial w_{hi}} = \left( l_i(x) + \sum_{t=1}^{K} \hat{w}_{ph}^t \frac{\partial x_h(\text{ch}_t[v])}{\partial w_{hi}} \right) \sigma'
\]  

(4.10)

\[
\frac{\partial x_h(v)}{\partial \hat{w}_{hi}^j} = \left( x_i(\text{ch}_j[v]) + \sum_{t=1}^{K} \hat{w}_{hh}^t \frac{\partial x_h(\text{ch}_t[v])}{\partial \hat{w}_{hi}^j} \right) \sigma'
\]  

(4.11)

where \(\sigma'\) is the first derivative of \(\sigma(\cdot)\) sigmoidal function.

The above equations are recurrent on the structures and can be computed by observing that for all the leaves of the structured data (all vertexes with null out-degree) equation 4.10 becomes \(\frac{\partial x_h[v]}{\partial w_{hi}} = l_i(v)\sigma'\), and all remaining derivatives are null. Consequently, we only need to store the output values of the unit and its derivatives for each component of the structure.

Finally, we perform a gradient ascent to maximize \(S\), that in the simplest case correspond to compute for each weight \(w_{hi}\) (including the type \(\hat{w}_{hi}^j\)) of the candidate hidden unit the following update equation

\[
w_{hi} := w_{hi} + \eta \frac{\partial S}{\partial w_{hi}},
\]  

(4.12)

where \(\eta\) is the learning rate. Although the implementation details are out of our scope, it is worth to note that different methods can be adopted to update the weight values.
on the base of the computed gradient. For example we have considered in the current implementation of the algorithms, beyond the standard back-propagation rule, the quick-propagation [45] and the R-prop [140] methods.

Is also worth noting that the gradient values computed by the RTRL style does not change adopting a different strategy, e.g. the BPTS, that can be adopted in the learning algorithm of RCC, for instance to reduce the computational complexity of the training algorithm (see Section 5.3).

Further details concerning the RCC model and learning algorithm implementation will be introduced in Chapter 5 (e.g. computational capabilities and computational cost).

### 4.4 Null Model

The null model is introduced for informative reason to better evaluate the performance of the various models. The null model is obtained by computing the expected value for the target over the training set. It predicts these constant values for all the input patterns. In terms of the defined function it can be specified by considering

\[
\tau_E(G) = \bar{t}, \quad G \in \mathcal{G}
\]

\[
g = \text{identity function}
\]

where \( \bar{t} \) is computed as mean of the values \( t_i \) for \((G_i, t_i) \in \mathcal{D}\) (training set).
Chapter 5

Properties and Analysis

In this chapter we study the characteristics of the proposed approach to recursive processing of SD, focusing on the models presented so far. The material presented in the previous chapters allows us to delineate in a definite frame the open problems of our research area. Various research demands must be addressed to complete the analysis of the neural realization of Adaptive SD-Recursive processing systems in the wide framework of the Machine Learning theory. Moreover, even though neural computing models for SD yield to successful applications for specific class of problems, such as in the case of sequential domains, the general class of sequence and structure learning problem is still far from being satisfactorily solved. In particular, the current models are characterized by constraints that lead to various limitations for learning structure transductions. Presenting criticisms to the current approach provide a uniform ground for motivating our original developments of the method (that will be presented in the following chapters).

Hence, in the first section we briefly review the main known results on the computational and approximation capability of the models and the main open problems analyzed in the literature (that also accounts for the richness of the research activity in the area). In the second section we propose an original analysis of the proposed approach showing both how the models can match the requirement for a good tool to deal with SD (as introduced in Section 2.3) and the need of further study and extensions in the field. Finally, specific properties and some details on the implementation of the recursive neural models (RNN and RCC) are investigated with comparison aims in Section 5.3.

5.1 Computational Properties of the Recursive Neural Networks

Two main aspects concern the computational properties of a neural method: the first one is aimed at studying the computational power of a given model, the second one is aimed at studying if the training algorithm is able to effectively learn the desired behavior. Clearly, if a target function cannot be represented by the hypothesis space, it is also not possible to learn it. The opposite is not true: there exist functions that can be in principle computable
by the model but the model cannot easily be trained from examples to perform it.

Theoretical properties of recursive models have been pioneeredly investigated in [73]. The recursive models have proved to be \textit{universal approximators} for specific classes of graphs, with reference to a fully connected architecture (i.e., the generic architecture shown in Section 4.2): for each mapping from labeled \textit{trees} to real vector spaces there exists a recursive network (i.e., an assignment of the parameters values) which approximates the mapping arbitrarily well on a set with probability arbitrarily close to 1. Results for the theoretical bound on the number of units for the learning of finite data sets are also available [73].

The recursive neural network allows valid generalization in the following sense: for finite input trees, it is possible to bound the probability that the empirical error and the expected error of the recursive neural network deviate more than \(\epsilon\) by a term depending on the number of parameters in the architecture, the number of training examples and \(\epsilon\) [73].

Furthermore, concerning formal theories of language and automata, other interesting results are known. Specifically, for recurrent neural networks (i.e. domain restricted to sequences) computational power has been longer investigated for a longer time: recurrent neural networks can simulate any finite state automaton [2, 136] and possess the power of Turing machines [148]. Some type of architectures can present a lower computational capability. For example, Elman-style network can simulate any finite state automaton[63, 108], while not fully connected network, such as recurrent Cascade Correlation and locally recurrent neural networks, cannot [57, 109, 50].

Moving to recursive models we can refer to the class of finite state machines known as \textit{frontier-to-root tree automata} (see [158, 61]). Most of the results developed for the representation of finite-state machines by recurrent networks can be applied as well to represent frontier-to-root tree automata by RNN (see [152, 113, 62]): for instance, any frontier-to-root tree automaton can be simulated by RNN. More in general, as presented above, most of the models can at least be simulated or approximated within the simple Elman-dynamic presented in Chapter 4 ([73]). However, also for recursive models, arguments concerning the constraining of the network have effects on the computational power of the model. In particular Recursive Cascade Correlation inherits the limits of the recurrent cascade correlation method. It must be noted that this limitation holds when representing certain cycles in some finite-state automata (useful to recognize arbitrarily long input sequences). Because of that, occurring such cases, the number of added hidden units the Recurrent Cascade Correlation model grows linearly with the longest string length of the training set. For most of the practical tasks used in data structures processing, these limitations do not hold since there are only finite structures.

More in general, the causality assumption of recursive models limits the class of functions that can be computed. This relevant issue is treated extensively in the next section and further in Chapter 6 of the thesis.

As mentioned above, the study of the computational power of the model is not sufficient to assure a proper learning of the tasks solution. In fact, although the model has in principle the capability of storing past information, remote information (i.e. informa-
tion far away from current input vertex) cannot always be learnt effectively. For gradient based training method, such as BPTT and RTRL algorithms, a basic difficulty is known as the problem of vanishing gradients. In [14] it is shown that, regardless of the particular algorithm considered to perform the gradient computation, and more in general for parametrized dynamical systems, two situations are to be considered: either the system is unable to robustly store discrete state information for the long term, or gradient vanishes exponentially. In the last case, the risk is that the short-term influences dominate the long-term influences in the computation of the total gradient. A theoretical analysis of this problem is studied in [14] and reviewed in [83]. However, since the analysis is based on theoretical arguments, the practical importance of such limitation needs to be evaluated on the real afforded tasks.

Various alternative training methods and architectures have been suggested to improve learning in the presence of long-term dependencies. For example, the problem can be mitigated allowing the model to use shorter paths for the effective propagation of the error signal (gradient) throughout the network. The model called NARX [117] follows this idea realized through the introduction of “shortcut” connections of the type $q^{-h}$, with $h > 1$, among the units of the recurrent neural network (specifically, NARX use feedbacks from output units).

A different approach as been recently proposed in [31] studying a training procedure for recursive neural networks based on the termed “layer-by-layer” learning algorithm [32]. Others proposal concern no-gradient optimization approaches or methods specifically designed to cope with vanishing error problem, such as the long short-term memory (LSTM) [84]. The last proposal has been shown to be able to efficiently learn to solve many previously learning difficult tasks. For a review see [83].

5.2 Analysis

This section deals with the characterization of the connectionist approach to the SD processing. This allows to highlight the current open-problems of the approach and to explain our motivations for a deeper investigation of neural methods for SD aimed at achieving new advancements.

We can delineate the features of Recursive Neural Networks as Adaptive SD-Recursive processing systems (Definition 3.4). Essentially the recursive neural networks deal with SD using the following concepts:

1 - A structured form of computation. The information is processed in a structured driven manner in order to perform a progressive transduction of the input example. The code of a structure is obtained hierarchically as composition of the code of its substructure (recursivity and compositionality). This is what we have addressed as the concept of recursive encoding. The isomorphism between the input structure and the encoding network shows graphically this concept. In this way the transduction is not-algebraic, i.e. the computed code depends on context information related to
the nested substructures. Arbitrary dimension of the input structure can be tackled. Moreover this method allows processing the local (vertex label) and the topological information with a homogeneous overall view.

The main aspect related to this concept is the *causality* assumption. A transduction is causal if the computation of each code depends only on the current node and nodes descending by it. The use of this assumption allows using internal states to memorize the code of substructures and to configure each neuron in analogy to Mealy/Moore models.

2 - *Parsimony.* The demand for parsimony stated in Section 2.3.2 is afforded by the use of the *stationarity* assumption. Stationary means that the computation that produces the code is the same for all the vertexes, i.e. we have used the same realization of the function \( \tau \) to process each vertex of a given input DPAG. The *encoding network* graphically shows as the same weights are used to perform a coding for each vertex of the DPAG. This method is related to the already known technique of *weight sharing* used in connectionism as a way to reduce the effective complexity of a network with a potentially large number of free-parameters. Actually, stationarity can be seen as a technique that significantly reduces the degrees of freedom of the model. In case of SD with a maximum out-degree, the stationarity assumption allows to realize models with a number of parameters that are independent of the dimensionality of the structure, i.e. the number of vertexes of the processed sequences, trees or DOAGs/DPAGs.

3 - *Adaptive transduction.* The encoding function is based on neural network models, i.e. \( \tau \) (and \( g \)) functions are realized by neural networks. Consequently the transduction functions depend on trainable parameters and can be adapted to the given task. This is the main advantage of this approach over the static coding scheme required when input is represented by attribute-value language. Moreover the hypothesis space is still continuous and the loss function differentiable so that it is possible to learn from examples by gradient methods extending the known methods used for standard neural networks. The propagation scheme of the gradient through the structure follows the form stated in point 1.

The above concepts can justify the following considerations. In Section 2.3.2 we have placed some conditions to evaluate the suitableness of a ML model for SD. Such conditions concern the representation of SD capability, the adaptive transduction, the handling of variability capability, respectively. According to these arguments, in our view, the recursive neural network is a suitable model for SD processing, for the following reasons:

**Representation of SD:** the hypothesis space is able to represent hierarchic relationships. The compact recursive formulation of the encoding process \( \tau_E \), composed with \( g \), represents syntactically distinct hypotheses for every assignment of the free-parameters.
Adaptive transduction: an adaptive measure of similarity is implicitly learnt by the model on the space of structures because the free-parameters are trained on the basis of the input-output examples. Points 1 and 3 above (in the current section) explain the generation of an adaptive coding of the structure and the possibility to have a suitable and efficient learning scheme. In particular, the prediction phase is fast since the coding scheme is applied without any relaxation or training process.

Handling of variability: the variability of data structures is faced by the structured form of computation (point 1) because the defined recursive coding is able to treat structures of different dimensions using the same form. We can note that

- the code of the structure is “distributed”, i.e. represented by the distributed output values of the set of units that realize the code space $\mathcal{X}$;
- in the recursive neural networks the generated codes are constrained to be all of the same size. Hence, the dimension of the code is independent with respect to the position of the current input in the data hierarchy or the number of structure vertexes. The processing of structure of arbitrary size does not change the computational schema.

Moreover, thanks to the stationarity assumption, the number of the free parameters of the model is constant with respect to the different instances of structures (with different number of vertexes or topology) occurring in the data set.

The model is also able to process unstructured data and, in that case, no changes are needed to perform the task on vectorial or sequential data. In such cases, since the model reduces itself to standard neural networks (or recurrent neural networks), it also shows the same efficiency of standard models.

However, these characteristics are also responsible for the limitations of the method. The first aspect concerns the class of graph that the model can process in a efficient manner: the structures employed are labeled DOAG and DPAG. Extensions to cyclic graphs are studied in [155]. Specifically, the introduction of cycles give rise to recursion in the in state transition system and therefore in the encoding network. The problem became to find a unique equilibrium of the network’s dynamics. Hence, relaxation techniques can be applied. However, using this strategy, although in principle feasible, the processing of cyclic graphs is paid in terms of efficiency of the learning algorithm. A different proposal to deal with cyclic graph has been recently introduced in [18] representing (in the pre-processing phase) the cyclic directed structures by tree structures. A recent proposal for the processing of DAG, i.e. removing of the ordering assumption, is in [17].

The assumptions of stationarity and causality are explained in the following.

Note that the headings chosen to describe the concepts 1 (structured form of computation), 2 (parsimony), and 3 (adaptive transduction) are motivated by our research aims. The challenge is to study the possibility to have structured form of computation which does not rely on strong causality assumption, or parsimony which does not assume stationarity, or adaptive encoding which does not use gradient methods.
5.2.1 The Stationarity Assumption

As mentioned above, stationarity of the model offers an easy solution to deal with variability and allows us to reduce significantly the number of independent parameters by the weight sharing constraint. Note that the constraint of weights within certain groups is an already known technique in connectionism as a way to reduce the effective complexity of a network with a large number of weights. For example, this method was applied to build translation invariance into networks used for image interpretation [116].

Stationarity is a common and quite natural assumption when the model deals with time sequences. The underlined assumption is that the variability of the statistical properties of the time series is very low and, therefore, it is natural that the model that processes the data does not vary with time. The same concept becomes more complex when dealing with non-sequential structures. Our extension of the concept to SD is based at the vertex level, i.e. the encoding function is invariant when applied to different vertexes. The universal approximation property of RNN mentioned in Section 5.1 assume such stationarity concept. Hence, in principle, this assumption does not reduce the expressive power of RNN. However, in general, it is possible to image a different scenario in which the stationarity may be defined at different levels such are edges or sub-structures. For example, the constraint may be harder sharing the weights among the connections associated with the subgraph spaces: a single free-parameter can be used to process each child of a given vertex instead of one for each children. On the contrary more parameters may be used for subgraphs, allowing to assign a neural connection to each edge of a specified template substructure. A formal evaluation of the different stationarity restrictiveness, relating it to the bias of the model, deserves further research. The most general case is to have a method that allows the selection of the particular encoding model realization to be used, depending on the context (label of the vertexes, position of the vertex in the structure, etc.). Non-stationary models may be useful for problems characterized by data structures which present information of various kind, categories, or abstraction level. A dynamical selection of the parameters to be used can lead to an information-driven stationarity. In these case the main difficulty is to find a suitable trade-off between expressiveness and complexity of the model. Indeed, stationary bias will be assumed throughout the thesis.

5.2.2 The Causality Assumption

Causality is inherently related to our approach because it is a necessary and sufficient condition to admit recursive state representation. As in the case of the stationarity assumption it is inherited from the adaptive processing of sequences. Hence, the discussion on causality can begin with the "recurrent" case.

When considering sequences, the causality assumption states that the output of the network at time $t_0$ only depends on input at times $t \leq t_0$. Causality is a natural assumption dealing with the modeling of the dynamics, e.g. temporal dynamics, of physical systems. In such cases is proper to assume that the system does not account input events that have not be presented as input data (relative to future time). The concept of context in recurrent
neural networks, as introduced by Elman [42], is to be understood in the restrictive sense as the recursive code/state computed by the causal recurrent/recursive dynamic to store “past” information.

However, in several applications causality is not sufficient. For example, all the tasks that require contextual information involving both the past and the future (i.e. that depend on both left and right sides of the sequence or on event located upstream and downstream) violate the causality assumption. The model does not violate the meaning of a physical system if we deal with finite sequences (structures) that are already entirely known at time of computation.

A notable example of application domain hampered by causal assumption can be found in the area of computational biology, whereas the task is the prediction of secondary structure of proteins from the string that represents the sequence of amino acids, i.e. their primary structure. The response of the model should depend on the elements in the sequence without the strong bias of a unidirectional (backward or forward) sequence reading as imposed by the causal models. Due to the pragmatic relevance of the problem, various approaches have been investigated. In case of sequences a simple IDNN model (see Section 4.1) allows to consider a fixed sized (symmetric) window of sequence elements centered in the current elements to feed a standard MLP neural networks. Despite the success in specific applications, including the cited prediction of secondary structure of proteins [143], this finite memory approach suffers of various limitations, as already discussed in Section 4.1. For recurrent neural networks the problem has been addressed by Baldi et al. [6, 7] which devised a bidirectional recurrent neural network, that for each element computes two internal states processing the sequence in the two directions, and then it computes the output combining these two internal states and the current input. The model constitutes a first approach to remove “causal” limitations, specialized for sequence domains (and applied to the prediction of secondary structure of proteins problem). An interesting research issue could be the development of further approaches with different bias and that can address some neural computing problems such as the problem of devise the right size of the network (approaches based on constructive methods).

Relaxing the causal assumption can be useful also for the structured domain whenever the meaning of a sub-structure depends on the “context” in which it is found. In such way it is possible to consider in which position within a larger structure the given substructure does occur. The context in such frame enlarges its meaning to consider the whole neighborhood of each vertex in the structure, i.e. not restricted to the descending vertex in the DPAG. Contextual processing can be referred to computations that yield for each vertex a response that depends on the whole information represented in the structured data.

The meaning of the causality assumption for SD can be easily understood looking at the unfolding process (encoding networks) (e.g see Figure 3.2) that allows to observe the functional dependencies of the code computed for each vertex of the input structure on the codes computed for the set of descendant vertexes.

The consequences of the causality assumption are even more complex for graph domains than for sequential domains.

Assuming causality and stationarity in fact we can note the following:
• Not any transduction can be computed: IO-isomorphic transductions that do not respect the causality assumption are trivially not representable by RNN models.

• Not any causal transduction on DOAG/DPAGs can be computed. An example of supersource transduction on DOAG/DPAGs that cannot be computed is introduced in [52]: to show this it is sufficient to devise two distinct DPAGs that are necessarily encoded into the same output value by any choice of the function $\tau$ and $g$ that realize $T_S$ (an instance will be presented in Chapter 6).

Because it has been proved that, under some circumstances, the causality assumption can be relaxed for sequence processing, it is valuable to study contextual extension that can be developed for recursive networks to deal with DPAGs.

Chapter 6 is devoted to a deeper and formal investigation of the above arguments and to propose an innovative approach to the relaxation of the causality assumption in the general setting of structured domains by means of contextual connectionist processing models (CRCC). A formal analysis of the computational properties of the contextual model will be devised to complete the characterization of the approach.

5.2.3 Learning Tasks

The learning tasks considered in the thesis are both supervised and unsupervised learning task for SD. The models presented so far originate in the framework of supervised learning. Studying the unsupervised paradigm we should first answer to demands concerning the applicability of the recursive approach to unsupervised learning:

• Can we extend the recursive approach to unsupervised learning?

• Can we formulate a suitable general framework based on distance functions to investigate properties independently from the specific realization?

Various methods are emerging to tackle the first problem. Self-organizing map methods for sequence and structure learning exploiting a recursive dynamics have been devised (e.g. recursive SOM [167, 168, 169], temporal Kohonen map [29], SOM-SD [71, 69, 153]). They effectively show that the recursive dynamics can be exploited in unsupervised learning. However, so far, such approaches are unrelated each other.

Starting from the observation that previous approaches to the problem of unsupervised learning by self-organizing map in SD share the same basic recursive dynamics, we pose here an issue trying to formulate in a general fashion the dynamics of the different approaches.

Our aim is to show that the recursive dynamics can be exploited in unsupervised learning in quite general form and to study the implications in such specific setting. In fact, such idea can be a basis to study specific aspects of SD processing in the frame of similarity measures. A unifying framework allows to combine different existing approaches and to constitute a base for new implementations and new models.
Various aspects of the approach, concerning the learning methods, and the theoretical properties arise:

- Is it possible to formulate Hebbian learning for such models in a uniform fashion?
- Is it possible to formulate the objectives of the learning process for SD in terms of an error/energy function?
- Is the Hebbian learning derivable as an exact gradient descent method of such functions?
- Is it possible to extend such methodology to alternative of the SOM?

The Chapter 7 presents our proposal to the above issues.

5.2.4 Other Approaches

The fact that there is no universal “best” learning method (e.g. see [38, 130]) is the main motivation to extend the study of SD processing to various ML methodologies. A comparison between the proposed recursive approach and approaches belonging to different areas of ML is a natural demand, and it can be the basis for further development of models for SD. In Chapter 8 we propose a first introduction aimed at relating the proposed approach to different approaches describing the relative advantages and pitfalls.

5.2.5 Assessments

The practical assessment is a critical phase for every new proposal. Since the research in recursive neural networks has begun to be a fertile area of research, the evaluation of the behavior of such models in applications is a relevant topic. We can distinguish two main classes of applications:

- **Proof-of-the-principle** applications, i.e. comparative assessments aimed at evaluating the real advantage of new models versus alternative models, in terms of performance. Benchmark data sets can be used to perform the comparison independently from their real significance. Examples of such experiments are included in Part II to evaluate the new proposed models.

- Applications to real-world problems and evaluation of the impact of the new approach. Given an interesting real-word problem and the new model, there are two aims:

  - the first one is to investigate if the application can really take advantage in adopting the proposed approach in terms of refinement of the modeling activity. The goal is to build a better modeling schema of the problem at hand or to deal with problems not yet faced, or not solved in satisfactory manner using standard approaches.
– the second aim is mainly related to the result outcoming from the applications, i.e. the goal is to obtain results interesting by themselves that cannot be achieved by standard methodologies. Of course, modeling real-world problems place new interesting issues: ideas for innovative solutions and new theoretical problems can arise thinking in terms of problems that need to be solved.

Both these types of aims are described in PART III with relationship to Computational Chemistry problems. The high pressure in searching for a new computational method that can reduce the burden of developing new materials and new drugs posed by the current research in Chemistry and Pharmacology is a strong motivation to study such problems (see Section 2.3.3).

Clearly, the application of all the proposed models to all the presented problems is out of the scope of this set of demonstration applications. In fact, although some experimental results are interesting by themselves (e.g. in the field of Chemistry and Drug Design), the applications should mainly be considered a step to present the general methodology proposed in the thesis. The main expected outcome is a set of new insights and new approach proposals.

5.3 Some Notes on the Comparison between Neural Recursive Models

In Chapter 4 we presented two different realizations of recursive models: the RNN fully-connected (with one hidden layer) architecture (RNN-fc for short in the following) and recursive cascade correlation (RCC) models. A practical demand emerges on the differences between the two models. In particular the comparison can address the differences in the application behavior and in the learning efficiency of the two approaches. We began a preliminary study of the performance comparison for RNN-fc architecture and RCC model applied to data structure problems ([68]) that has been successively extended in [66].

Besides the specific empirical comparison results, that are out of the main aim of this thesis, some interesting conclusions can be drawn. In fact, from the comparison between constructive and fixed-size RNN architectures we deduced some suggestions to deal with the problem of design optimal recursive neural network architecture.

Various experiments were considered to compare the models on an empirical base. The tasks concern both classification and regression tasks and the data set included two benchmark SD problems for image processing, i.e. the “Policeman” and the “extended Policeman” benchmark, as described in [70], and the activity prediction of benzodiazepines, as described in PART III of this dissertation.

1The result summarized in this section are joint work with A.C. Tsoi and M. Hagenbuchner
5.3. ON THE COMPARISON BETWEEN NEURAL RECURSIVE MODELS

Basically, experimental results have showed that the two approaches produce comparable results. Nevertheless, some interesting differences can be observed. The results shown that RNN-fc improves generalization performance on classification learning problems. However, RNN-fc networks do not always converge if the number of hidden units is minimal. RCC always converged to a solution. This indicates that RNN-fc networks are more sensitive to local minima problems.

Clearly, the fundamental difference between RNN-fc and RCC models is that in RCC the number of hidden layer neurons required for a given learning problem is detected automatically. In contrast, for RNN-fc networks the number of hidden units needs to be chosen before conducting network training. This is often a difficult task, especially if there is no information about the complexity of the given learning task. Choosing too few hidden units, the RNN-fc network will be unable to solve a given problem satisfactory. On the other hand, selecting too many hidden units will result in long training times and in the increase of overfitting probability. Thus, the ability of RCC to detect the number of hidden layer neurons required for any given learning task is a significant advantage.

Furthermore, these experiments confirm that the theoretical computational limits of RCC do not have a significant negative effect on real world problems. It has been observed that a less powerful model such as RCC is able to obtain efficient results in specific applications (that are characterized by finite structures).

In general RCC may find a very efficient solution without any prior knowledge on the problem. At least RCC is a useful instrument to make the first approach to a new problem and to have an idea about the number of hidden units needed for the problem under analysis. When conducting these experiments the general finding was that RCC could predict the minimum number of hidden layer units required for a corresponding RNN-fc network. RNN-fc has shown that, using a number slightly larger than the minimum number of hidden neurons, the generalization performance may be improved as well as the network becomes more stable as it converges to low error levels for a wide range of initial network states.

Additional arguments can be considered regarding the peculiarities of the structured domains. Even considering a bounded out-degree and a finite alphabet for the structure labels, it is easy to see that the number of distinct possible structures is very high. This give raise to a combinatorial explosion of the number of possible input pattern (for example with respect to the height of the trees). We can deduce that the sampling of the SD will be, in general, quite poor. In such cases, finding an optimal solution to the minimization of the risk function is to be considered a critical task. The specific constraints given by the fixed size of the RNN-fc models and the possibility to fall in local minima using a gradient descent technique can exacerbate the problem. Constructive approaches, such as RCC, can mitigate such constrains. The main drawback is to cope with the overfitting problems that a constructive approach can encounter. In Section 5.4 we analyze some methods to deal with this problem.
Computational Complexity of Learning Algorithms of RCC

The algorithm presented in Chapter 4 implements a gradient technique (local search) and thus it is not possible to guess the number of iterations necessary to converge. In the following, to have a feeling of the complexity, we suppose that each hidden unit is trained for a bounded number of iterations\(^2\). Let \(V\) be the number of vertexes in the DOAG. Let us start with the RCC model. The number of free parameters (connections and threshold) \(N_h\) for the \(h\)th hidden unit in RCC is

\[
N_h = n_L + hK + \frac{(h - 1)}{2} + 1
\]  

(5.1)

where \(K\) is the valence of the domain. Consequently, observing that during learning the output of the frozen hidden units computed on the training set can be stored, the output of the \(h\)th hidden unit on a single pattern (i.e., graph vertex) can be calculated in \(O(hK)\). This complexity dominates the cost of computing the output of the output unit which is proportional to

\[
\frac{n_L}{\text{label dim.}} + \frac{h}{\text{frozen units}} + 1
\]

Concerning learning, it is trivial to note that the cost of training a single hidden unit dominates the cost of training the output unit. According to learning equations of RCC (Section 4.3), the derivatives for the \(h\)th hidden unit with respect to a single pattern can be computed in \(O(hK^2)\) in time, since \(\text{out deg}(v) \leq K\) (the full cost is \(O(K(n_L + hK))\)). Thus, considering the full training set it takes \(O(VhK^2)\) in time. Finally, when building a network with \(h\) hidden units, the computation of all the derivatives takes \(O(Vh^2K^2)\) in time\(^3\). The complexity in space is dominated by the space necessary for storing the derivatives of the current trained unit, that is in principle up to \(O(VhK)\). However, such cost has been reduced in our implementation of the algorithm to \(O(V)\).

When considering a diagonal connection matrix, i.e., hidden units with self-recursive connections only, the complexity of learning became \(O(VhK^2)\) in time and \(O(VK)\) in space (\(O(V)\) in the current implementation).

Considering the number of candidate units in the pool (\(C\)) and the number of epochs used for training (\(e\)) the total become \(O(Vh^2K^2Ce)\). Following the same reasoning, for RNN-fc architecture using BPTS algorithm we obtain \(O(Vh^2K\epsilon)\).

Note that the difference between RNN-fc with BPTS and the RCC is due to the use of a RTRL strategy for RCC. It is possible to use a BPTS learning algorithm also for training of candidate units of RCC: in this way the computational complexity\(^4\) is \(O(Vh^2KC\epsilon)\) that is the same as for BPTS fully connected architectures.

\(^2\)In practice, a bound on the number of iterations for the training of a single hidden unit is always used.

\(^3\)The total number of computations in one step is proportional to \(\sum_{i=1}^{h} ViK^2 = VK^2 \sum_{i=1}^{h} i = VK^2\frac{h(h+1)}{2}\).

\(^4\)It is exactly \(O(Vh(h+1)/2KC\epsilon)\).
5.4 Avoiding Overfitting in RCC

Constructive models presented in Section 4.3 are based on a greedy approach that can lead to the building of complex hypotheses that over-fit the training data and that cannot achieve an “optimal” generalization performance.

In fact, according to the theory of ML (Section 2.1), overfitting can be easily recognized by screening the performance of the network on a test set during learning: initially the test error decreases along with the training error, till it reaches a minimum and then it starts to increase, while the training error keeps decreasing. The increase in the test error indicates that the network is starting to learn regularities in the training set which are not of general validity, thus showing overfitting.

A very common approach to avoid overfitting is to stop training as soon as the minimum on the test error is reached. This approach, however, implies the availability of enough data for both the training and test set, and for a further validation set used to estimate the true generalization performance. In all of the applications problem studied in PART III we do not have enough data to adopt this strategy.

Hence, aiming at mitigating the overfitting phenomenon we resort to several refinements of the RCC implementation.

First of all, no connections between hidden units were allowed. This allows us to reduce the number of connections of the units without introducing a lost of information in the processing of the structures: the codes of the substructures are fed to the current unit from all the frozen units.

Then, the gain of the sigmoids of the hidden units is set to a low value (0.4). This allows us to extend the linear range of the activation function.

Finally, an incremental strategy (i-strategy) on the number of training epochs was adopted for each new inserted hidden node. The number of training iteration is set to low values for the firsts units and progressively increased adding new units to the RCC network. This was done because by allowing few epochs to the first units we avoid the increase of the weight values and the subsequent saturation of the units. The complexity of the model grows proportionally to the number of effective parameters, i.e. parameters with values far from zero. On the other hand, lately introduced units, which work with small gradients due to the reduction of the residual error, take advantage from the increased number of epochs.

The advantages of this strategy on the generalization performance will be show in the applications of PART III: an example of the efficacy of the method on the generalization performance is introduced in Chapter 11 evaluating the prediction performance of “benzodiazepines” experiments (see in particular Section 11.1.3). The work of Bartlett [10] gives theoretical support for techniques, like the i-strategy, that allows producing networks with small weights.

Beyond the concrete performance advantages of the proposed i-strategy, related to the specific application settings (an example is discussed in Chapter 11), we show here by an instance the dynamical behavior of the i-strategy. Specifically we aim at showing the effectiveness of weight value size reduction provided by the i-strategy.
Without loss of generality, we consider an illustrative instance of application (in the case concerning the alkane data set introduced in Section 2.3.3). A comparison between the two possible cases, applying (i.s.) and not applying (no-i.s.) the strategy, is shown in Figure 5.1: we plot the weight vectors norm (Euclidean norm) for the firsts units of a RCC model. We can observe that the no-i.s. training process of RCC allows the norm of weights for the firsts units to grow toward high values. In contrast, the adoption of the \textit{i-strategy} allows to effectively reduce such values.

![Figure 5.1](image-url)

**Figure 5.1:** Norm of the weight vectors of the first 23 units for 2 RCC models: applying (i.s.) and not applying (no-i.s.) the \textit{i-strategy}. 
Chapter 6

Contextual Processing of Structured Domains by RCC

In this chapter we propose a first approach to deal with contextual information in structured domains by recursive neural networks. As pointed out in Section 5.2 the analysis of the causality assumption places a research request toward the extension of the capability of the recursive neural models. The proposed model, i.e. Contextual Recursive Cascade Correlation (CRCC), a generalization of the Recursive Cascade Correlation (RCC) model, is able to partially remove the causality assumption by exploiting contextual information stored in frozen units.

Extensions of the Causality Assumption Recursive neural networks, as well as almost all the recurrent neural network models proposed in literature are based on the causality assumption. When considering sequences, the causality assumption states that the output of the network at time $t_0$ only depends on input at times $t \leq t_0$. In the framework of structure processing, a model is causal (i.e., it strictly satisfies the causality assumption) if the output for a given vertex of a directed acyclic graph only depends on the information conveyed by the current vertex and the vertexes descending from it. This assumption allows the model to use internal states of the network to memorize information about substructures.

Nevertheless, several prediction tasks involving items both in sequences and structured data domains, where we assume the availability of the whole finite sequence or structure at the time of processing, require processing of information from both the “past” and the “future”, i.e., contextual information. The DNA and proteins analysis, as well as language understanding, are examples of these tasks. In such cases the response of the system depends on information located both upstream and downstream of the current processed portion of the input sequence. Standard solutions to this problem, in the framework of sequence processing, involve feedforward neural networks that look at the input through a fixed window of predefined size [138] [170]. These approaches, however, are not practical if a priori knowledge is not available on the “optimal” size of the window. Some authors suggested to solve the fixed size window problem, still in the
case of sequences, by specific models that compute the output by combining information propagated from both the “past” and the “future”. This is performed by spanning the sequence in the two directions. For example, in the recurrent model proposed in [6, 7] for sequence processing, the recurrent dynamics is factorized into a forward transition and a backward transition. In particular the devised Bi-directional Recurrent Neural Network (BRNN) is composed of three sub-networks: one for computing the “past” information, one for computing the “future” information, and finally one sub-network which combines all the information to produce the output. The general idea of the approach presented in [6, 7] can be described in our framework considering a factorization of the internal state into two parts, \( x(t) = [x_1(t), x_2(t)]^T \), where \( x_1 \in \mathbb{R}^m \) and \( x_2 \in \mathbb{R}^m \) keep information about the past (left-to-right direction) and the future (right-to-left direction), respectively. The state transition function of the system is then defined as:

\[
x(t) = \begin{bmatrix} x_1(t) \\
                        x_2(t) \end{bmatrix} = \begin{bmatrix}
                        \tau_1(l(t), x_1(t-1)) \\
                        \tau_2(l(t), x_2(t+1))
                        \end{bmatrix}
\]

(6.1)

and the output is produced as

\[
y(t) = g(l(t), x_1(t), x_2(t))
\]

(6.2)

where each item of the sequences is identified by the time-stamp \( t \). The approach presented in [6, 7] is also an example of a successful application of recurrent neural network to sequence domains in the area of structural genomics (prediction of the protein secondary structure) that clearly take advantage in overcoming the original causal limitations of the recurrent neural networks. However, new proposals are needed to include the approach in constructing methods, allowing to automatically discover the network size (i.e., the number of hidden units) and to deal with more complex domains than sequences.

A different approach is presented in this chapter. To introduce the basic idea we can firstly restrict to sequence domain (indeed, the first idea was developed for the specific case of sequence domain and appear in [124] where the proposed model was termed as Bi-causal Recurrent Cascade Correlation (BRCC)). The proposed model is a variant of the basic Recurrent Cascade Correlation [46]. Actually, when training a Recursive Cascade Correlation Network, hidden units are frozen one by one as new units are added. Since weights of frozen units are not allowed to change, it is possible to use the state information of the frozen units to also analyze an internal representation of the “future” inputs. When training a new hidden unit the information stored in frozen units can be accessed. In this way, when processing a sequence \( s \) at a time \( t \), it is possible to use the stored activations for all the following subsequences of \( s \),

\[
s_{[0],1}, s_{[0,2]}, \ldots, s_{[0,l]}, s_{[0,l+1]}, \ldots, s_{[0,l_u]}
\]

where \( s_{[0,j]} \) is the subsequence of \( s \) in the interval \( [0, j] \) and \( t_s \) is the length of the sequence \( s \).

The above approaches to contextual processing of sequences are faced partially relaxing the strictly causal assumption of the original recurrent approach. In general, to capture
the underlying idea, we could define the bi-causality as a double and simultaneous causal dependency over the past and the future.

As in the case of sequences, causality is not sufficient when considering structured domains where the computational task requires complete contextual information, or, more in general, when there is no knowledge supporting the causality assumption. For instance, non-causal models can be useful when dealing with structured data where the meaning of sub-structures depends on the context in which they are found, i.e., in which position, within a larger structure, the given substructure does occur. The challenge is therefore to study the possibility to process structures by a recursive neural network model, relaxing the causality assumption.

In the following we describe a Contextual Recursive Cascade Correlation (CRCC) for the processing of directed positional acyclic structures, based on an extension of the Recursive Cascade Correlation model (RCC, Section 4.3). and of the cited BRCC [124].

Structure of the Chapter The chapter is organized as follows. In Section 6.1 we specialize notations on structured domains to cope with context and DPAGs. In Section 6.2 we introduce the contextual recursive model (CRCC). Then, in Section 6.3 we formally show that CRCC can compute contextual structural transductions which cannot be computed by RCC. Moreover, we demonstrate that some causal supersource transductions which cannot be computed by RCC, can be computed by CRCC, which on the other hand is able to compute all the transductions that can be computed by RCC. We are also able to formally elucidate how the “shape” (i.e., which state variables are accessed) of the contextual information evolves with the addition of hidden units. Due to the relative amount of details, the proofs of the main theorems for the formulation of the context are postponed in a specific section (Section 6.4). In Section 6.5 we propose a neural network implementation of the general model, with details concerning the learning algorithm. Special cases and extensions of the approach are introduced in Section 6.6. Section 6.7 is devoted to the experimental part. Experimental results on controlled sequences and on a real-world task involving chemical structures confirm that CRCC is basically equivalent to RCC when considering a fully causal prediction task, while it is superior when considering contextual transductions (that RCC cannot compute) or prediction tasks where no information about the validity of the causality assumption is available. The analysis of the internal representation of the CRCC (code space) allows to drawn other insight in the analysis of the method. Conclusions are drawn in Section 6.8.

As mentioned above the first idea of CRCC appeared in [124] applied to sequence domain. The basic idea of CRCC approach already appears in [125], and the formal determination of context (Sections 6.3 and 6.4) are based on idea developed in [127] [126].
6.1 Preliminaries

Since we consider the extension of the models for contextual processing of SD, we recall and extend the notation introduced in Chapter 3 to better capture the concepts of context in DPAGs. The domain \( \mathcal{G} \) is still characterized by a set of labeled directed positional acyclic graphs \((G)\) with supersource and with bounded out-degree \( \text{out} \) and in-degree \( \text{in} \).

In particular we need to consider the predecessors of a vertex and we redefine as a set the predecessors and successors of a vertex \( v \), as follows

- \( \text{out\_set}(v) = \{ u | (v, u) \in \text{egd}(v) \} \);
- \( \text{in\_set}(v) = \{ u | (u, v) \in \text{egd}(v) \} \);
- \( \text{out\_deg}(v) = |\text{out\_set}(v)| \) (out-degree);
- \( \text{in\_deg}(v) = |\text{in\_set}(v)| \) (in-degree).

In this chapter we assume that for each vertex \( v \in \text{vert}(G) \), two injective functions \( P_v : \text{egd}(v) \to [1, 2, ..., \text{in}] \) and \( S_v : \text{egd}(v) \to [1, 2, ..., \text{out}] \) are defined on the edges entering and leaving from \( v \). In this way, a positional index is assigned to each entering and leaving edge from a node \( v \). Moreover, we define \( \forall u \in \text{vert}(G) \)

\[
\forall j \in [1, \ldots, \text{in}] \quad \text{in\_set}_j(u) = \begin{cases} v & \text{if } \exists v \in \text{vert}(G) \mid P_u((v, u)) = j \\ \text{nil} & \text{otherwise} \end{cases}
\]

\[
\forall j \in [1, \ldots, \text{out}] \quad \text{out\_set}_j(u) = \begin{cases} v & \text{if } \exists v \in \text{vert}(G) \mid S_u((u, v)) = j \\ \text{nil} & \text{otherwise} \end{cases}
\]

With \( \text{dist}(u, v) \), where \( u, v \in G \), we denote the shortest (directed) path in \( G \) from the vertex \( u \) to the vertex \( v \).

Examples of notation are shown in Figure 6.1.

For the domain \( \mathcal{G} \) we consider in the following IO-isomorphic transductions \( \mathcal{T}_G \) as defined in Section 3.2. It is well known that recursive neural networks, which include the Recursive Cascade Correlation model, can only implement causal IO-isomorphic transductions, i.e., the output computed at vertex \( v \) only depends on the information (i.e., labels and structural information) stored in \( v \) and descendants of \( v \). Moreover, if both causality and stationarity are assumed, any causal IO-isomorphic transduction can be described by a supersource transduction, i.e., a transduction that computes an output only for the supersource of the input DPAG (Section 3.2). It is worth noting that under these conditions, a recursive neural network cannot compute any causal transduction (see [52] and Section 6.3). In the following we discuss a new model able to extend the class of functions that can be computed on structured domains.
6.2 Contextual Recursive Model

Recursive neural networks possess, in principle, the ability to memorize “past” information to perform structural mappings. The state transition function \( \tau() \) and the output function \( g() \), in this case, prescribe how the state variable, or better the state vector \( \mathbf{x}(v) \) associated to a vertex \( v \) is used to obtain the state and output vectors corresponding to other vertexes, respectively. The machinery to describe the models, based on generalized shift operators, was introduced in Section 3.2. In the following, we recall, and we extend to predecessors of a vertex such definitions.

Specifically, given a state vector \( \mathbf{x}(v) \equiv [x_1(v), \ldots, x_m(v)]^t \), we define the generalized shift operators (extending Definition 3.2 to consider both in_set and out_set operators):

- If \( \text{out_set}_j(v) = \text{null} \) then \( q_j^{-1}x_k(v) \equiv x_0 \), the null state\(^1\),
  \( q_j^{-1}x_k(v) \equiv x_k(\text{out_set}(v)) \) otherwise;

- If \( \text{in_set}_j(v) = \text{null} \) then \( q_j^{+1}x_k(v) \equiv x_0 \)
  \( q_j^{+1}x_k(v) \equiv x_k(\text{in_set}(v)) \) otherwise.

\(^1\)In this Chapter, we assume \( x_0 = 0 \). Other assumptions can be considered, e.g. having a different null state for each \( j \) and missing entering/leaving edge.
Moreover, we define

\[
\mathbf{q}^{-1} x_k(v) = \begin{bmatrix}
q_1^{-1} x_k(v) \\
\vdots \\
q_{\alpha}^{-1} x_k(v)
\end{bmatrix},
\]

and, given \( e \in \{-1, +1\} \), we can (re)group a set of \( \mathbf{q}^e x_k(v) \) for \( k = 1, \ldots, m \), as in the following

\[
\mathbf{q}^e \mathbf{x}(v) = [\mathbf{q}^e x_1(v), \ldots, \mathbf{q}^e x_m(v)].
\]

Also, we recall that the mapping implemented by a recursive neural network can be described in terms of these definitions by the following equations:

\[
\begin{align*}
\mathbf{x}(v) &= \tau(l(v), \mathbf{q}^{-1} \mathbf{x}(v)) \\
\mathbf{y}(v) &= g(\mathbf{x}(v))
\end{align*}
\]

where \( \mathbf{x}(v) \) is the network state associated to vertex \( v \). This formulation, however, is based on a structural version of the \textit{causality} assumption, i.e., the output \( \mathbf{y}(v) \) of the network at vertex \( v \) only depends on descendants of \( v \). Specifically, RCC equations, where we disregard direct connections between hidden units, can be written, for \( j = 1, \ldots, m \), as

\[
x_j(v) = \tau_j(l(v), \mathbf{q}^{-1}[x_1(v), \ldots, x_j(v)]),
\]

where \( x_i(v) \) is the output of the \( i \)-th hidden unit in the network when processing vertex \( v \). Since RCC is a constructive algorithm, training of a new hidden unit is based on already frozen units. Thus, when training hidden unit \( k \), the state variables \( x_1, \ldots, x_{k-1} \) for all the vertexes of all the DPAGs in the training set are already available, and can be used in the definition of \( x_k \). This observation is very important since it yields to the realization that \textit{contextual} information is already available in RCC, but it is not exploited.

Our proposal is to exploit this contextual information when training of a new hidden unit. Specifically, Equations 6.7 can be expanded in a contextual fashion by using, where possible, the shift operator \( \mathbf{q}^+ \):

\[
\begin{align*}
x_1(v) &= \tau_1(l(v), \mathbf{q}^{-1} x_1(v)) \\
x_j(v) &= \tau_j(l(v), \mathbf{q}^{-1}[x_1(v), \ldots, x_j(v)]), \\
\mathbf{q}^+ x_j(v) &= \mathbf{q}^+ [x_1(v), \ldots, x_{j-1}(v)], \\
&= \mathbf{q}^+ [x_1(v), \ldots, x_m(v)],
\end{align*}
\]

which constitute the equations for the proposed Contextual Recursive Cascade Correlation (CRCC). It should be noted that in Equation 6.9 the shift operator \( \mathbf{q}^+ \) cannot be applied
6.2. CONTEXTUAL RECURSIVE MODEL

Figure 6.2: Graphical model for $x_k$ in CRCC. Only the functional dependencies for $k$ and $k-1$, besides to the input, are shown explicitly. Here we assume $in = out = 2$.

to $x_j(v)$ since this would introduce a cyclic dependence in the definition of the variable, i.e., a dependence of $x_j(v)$ from the state variables of its parents which, however, via the shift operator $q^{-1}$ depend on the value of $x_j(v)$ itself. A graphical model for $x_k$ in CRCC is shown in Figure 6.2.

Following the line developed in Sections 3.3 and Chapter 4 it is easy to define neural network models that realize Adaptive SD-Recursive processing system based on the CRCC approach. In fact, it is possible to realize any $\tau_j$ of Equation 6.9 by a neural unit associating each connections (i.e., arrows) of the graphical model in Figure 6.2 to the free parameters $W$ of the neural model.

Given a vertex $v$, the new model takes in input for each neural unit:

- connections associated to the input label of $v$, and connections associated to the state variables (both the output of already frozen hidden units and the current $j$-th hidden unit) of the children of the current vertex $v$ (as for the RCC model presented in Section 4.3);

- connections associated to the state variables of the parents of $v$ which are already frozen (i.e., the output of already frozen hidden units computed for the parents of $v$), $q^{i+1}[x_1(v), \ldots, x_{j-1}(v)]$.

The connections associated to parents add new parameters to the model that are fundamental to the contextual processing. It is worth to note that, however, the number of parameters is still of order $O(m^2)$ as for RCC, since for each unit the number of input connections grows of a constant number $in$. The equations of the proposed neural realizations of the model are introduced in Section 6.5. Since the results presented in the following are independent from a specific realization, we do not need to deal now with the realization details.
6.3 Computational Properties of CRCC

Since CRCC exploits contextual information, it is important to fully understand the implications on the class of functions that the proposed model can compute. In the following, we give a definition of “contextual window” and elucidate how the “shape” of the “contextual window” evolves with the addition of hidden units. Finally we show that CRCC can compute functions that cannot be computed by RCC.

In this section, we characterize the “shape” of the context exploited by a state variable in terms of the set of state variables which contribute directly or indirectly to its determination. Specifically, we propose a definition of “contextual window” and elucidate how the “shape” of the “contextual window” evolves with the addition of hidden units, both in the CRCC model and in the RCC model.

In order to formalize the above concepts, let us define the “descendants” operator \( \downarrow \):

**Definition 6.1 (Descendants operator)** The “descendants” operator \( \downarrow \) applied to a subset of vertexes \( V \subseteq \text{vert}(G) \) returns the union of \( V \) with the set of all its descendants

\[
\downarrow V = V \cup \{ u \mid v \in V \land \exists \text{ path}(v, u) \},
\]

or alternatively, using a recursive definition

\[
\downarrow V = V \cup \downarrow \text{out set}(V),
\]

where the descendants set of an empty set is still an empty set, i.e., \( \downarrow \emptyset = \emptyset \).

Moreover, let the set functions \( \text{in set}(\cdot) \) and \( \text{out set}(\cdot) \) be defined also for sets of vertexes as argument (e.g., \( \text{in set}(V) = \bigcup_{v \in V} \text{in set}(v) \)) and let denote the repeated application with the powered function (e.g., \( \text{in set}^p(V) \) corresponds to the repeated application of the function \( \text{in set} \) for \( p \) times):

\[
\text{in set}^p(V) \equiv \underbrace{\text{in set} \ldots (\text{in set}(V)) \ldots}_{p \text{ times}}
\]

Actually, we will use the repeated application of the \( \text{in set} \) function composed with the “descendants” operator \( \downarrow \), i.e,

\[
(\downarrow \text{in set})^p(V) \equiv \downarrow \text{in set} \ldots (\downarrow \text{in set}(V)) \ldots
\]

For representational purposes we also introduce the following notation

**Definition 6.2 (Dot operator)** Given a subset of vertexes \( V \), with the notation \( x_j.V \) we refer to the set of state variables \( \{ x_j(v) \mid v \in V \} \), where for an empty subset of vertexes we define \( x_j.\emptyset = \{ x_0 \} \).
Finally, let formally define the concept of context window, i.e. the set of internal states which may contribute to the computation of a new internal state

**Definition 6.3 (Context)**

The context window of a state variable \( x_k(v) \), denoted by \( \mathcal{C}(x_k(v)) \), is defined as the set of all state variables (directly or indirectly) contributing to its determination.

In the following we will use the above definition to characterize the context window of the CRCC and RCC models.

For CRCC, it is possible to show the following result

**Theorem 6.4** Given a DAG \( G \), for any vertex \( v \in \text{vert}(G) \), and for any index \( k \geq 2 \), the following equation holds for CRCC

\[
\mathcal{C}(x_k(v)) = \bigcup_{i=1}^{k-1} x_i. (\downarrow \text{in-set})^{k-i} (\downarrow v) \cup x_k. \downarrow \text{out-set}(v). \tag{6.10}
\]

**Proof.** See Section 6.4 for proof. 

This theorem clearly shows that the introduction of each new hidden unit in CRCC increases of one “step” the size of the context in the direction of the “future”.

This expansion of the context can be shown for different class of graphs, as in the following examples:

**Example 6.1 (\( \mathcal{C}(\cdot) \) for Sequences)** The expansion of the context can easily be visualized when considering a temporal sequence, which in our framework is represented as a list where the last element is denoted by \( v_1 \) and element \( v_t \) corresponds to the item occurring at time \( t \). In this case, the context becomes

\[
\mathcal{C}(x_k(v_t)) = \bigcup_{i=1}^{k-1} x_i. \downarrow v_{t+k-i} \cup x_k. \downarrow v_{t-1} \tag{6.11}
\]

An example of visualization of Equation 6.11 is reported in Figure 6.3 for \( k = 3 \) and \( t = 2 \). Specifically, the graphical of RCC is unfolded through time \( t \) for the input sequence up to \( k = 3 \) units. Note that each unit takes input both from the past and from the future of preceding units. Concerning the future, in Figure 6.3 we have highlighted the dependency of \( x_k(t) \) on input labels at times up to \( t + k - 1 \), so to show that the window size on the future is directly proportional to the number of hidden units.
Example 6.2 (\(C(\cdot)\) for Trees) When considering a tree, the context grows (via the in_set function) including all the subtrees (because of \(\downarrow\)) rooted in vertexes met along the (inverse) path between the current vertex and the root (i.e., supersource) of the tree. An example of expansion of the context for a tree is given in Figure 6.4, where we show how adding new hidden units to the CRCC network leads to an increase of the “context window” associated to each vertex \(v\). Specifically, the shown example focuses on the state computation of the vertex labeled “d” in the input tree, and describes for it, in a pictorial way through boxes, the functional dependences introduced by any new hidden unit inserted in the network. Unit 1 (see Equation 6.8) implements only causal computation. After adding unit 2, contextual information concerning the subtree rooted in the vertex labeled “g”, contributes to the state definition of the vertex labeled “d”. Finally, after adding unit 3, the context is extended to the whole tree.

Example 6.3 (\(C(\cdot)\) for DPAGs) The growing of the context for a DPAG is a bit harder to understand. This is due to the fact that in a DPAG a vertex may have more than one entering edge. So the context grows (via the in_set function) including all the sub-DPAGs (because of \(\downarrow\)) rooted in vertexes met along all the (inverse) paths between the current vertex and the supersource. Moreover, paths reaching descendants of vertexes included in the current context must be taken into account when considering the new context obtained
6.3. COMPUTATIONAL PROPERTIES OF CRCC

Figure 6.4: Evolution of the “context window” for the vertex labeled “d” in the input tree with the growing of the network, shown at the lower part of the figure via the evolution of the graphical model of CRCC. The factorization of the context, as shown in Equation 6.10, is visualized by boxes referring to corresponding state variables.

by adding a new hidden unit. An example of how the context grows for a DPAG is shown in Figure 6.5.

In general, for CRCC, the evolution of the context with respect to the addition of hidden units is characterized by the following property.

**Proposition 6.5** Given a vertex \( v \) in a DPAG \( G \) with supersource \( s \), such that \( \text{dist}(s, v) = d \), the contexts \( C(x_h(v)) \), with \( h > d \) involve all the vertexes of \( G \).

**Proof.** According to Equation 6.10, when computing \( C(x_{d+1}(v)) \), in_set is recursively applied \( d \) times. Thus the shortest path from \( s \) to \( v \) is fully followed in a backward fashion starting from \( v \), so that \( x_1(s) \) is included in \( C(x_{d+1}(v)) \). Moreover, since \( x_1(s) \) is included in \( C(x_{d+1}(v)) \), by definition of Equation 6.10, also the state variables \( x_1(u) \) for each \( u \in \text{vert}(G) \) are included in \( C(x_{d+1}(v)) \). The statement follows from the fact that \( C(x_{d+1}(v)) \subseteq C(x_h(v)) \).
Moreover, when considering a target function that for each vertex \( v \) depends on the whole structure, the following proposition suggests that such information becomes available to the CRCC model.

**Proposition 6.6** Given a DPAG \( G \) there exists a finite number \( h \) of hidden units such that for each \( v \in \text{vert}(G) \) the context \( C(x_h(v)) \) involves all the vertexes of \( G \). In particular, given \( r = \max_{v \in \text{vert}(G)} \text{dist}(s, v) \), any \( h > r \) satisfies the proposition.

**Proof.** Let consider \( h = r + 1 \). The proposition follows immediately by the application of Proposition 6.5 for each \( v \) since \( h > \text{dist}(s, v) \).

Note that, differently from CRCC, the context for RCC is characterized by the following:

**Theorem 6.7** Given a DAG \( G \), for any vertex \( v \in \text{vert}(G) \), and for any index \( k \geq 1 \), the following equation holds for RCC

\[
C_{RCC}(x_k(v)) = \bigcup_{i=1}^{k} x_i \cdot \downarrow_{\text{out-set}}(v)
\]  

(6.12)

**Proof.** See Section 6.4 for proof.

The explicit formulation of the context stated in Theorems 6.4 and 6.7 allows to formal characterize the differences between contextual model CRCC and the causal model RCC. Note that, since \( \text{in-set} \) does not occur in the formula, Proposition 6.5 and Proposition 6.6
cannot hold. In fact, RCC can only use information about the descendants of a vertex \( v \), and for this reason it will never be able to include other vertexes in any context of \( v \). From a computational point of view this implies that RCC will never be able to compute a function whose output for a given vertex \( v \) depends on vertexes of the input structure that are not descendants of \( v \).

It is worth to note that also considering other realizations of causal Adaptive SD-Recursive processing systems (such the generic defined RNN), formula similar to 6.12 can be devised. The basic difference between causal and contextual model is focused on the capability to include information from the parent of the current processed vertex.

In addition to that, focusing the attention to supersource transductions, the following result holds

**Lemma 6.1** There exist two distinct DPAGs, \( G_1 \) with supersource \( s_1 \) and \( G_2 \) with supersource \( s_2 \), such that the state vectors \( x^{RCC}(s_1) \) and \( x^{RCC}(s_2) \) computed by RCC are identical, while the state vectors \( x(s_1) \) and \( x(s_2) \) computed by CRCC are different.

**Proof.** Let consider the two graphs in Figure 6.6, where \( s_1 \equiv v_1 \) and \( s_2 \equiv v_5 \). We show that while RCC returns, regardless of the state vector dimension \( k \), \( x^{RCC}(s_1) = x^{RCC}(s_2) \), CRCC is able to produce different state vectors \( x(s_1) \) and \( x(s_2) \) as soon as the dimension of the state vector is higher than 1, i.e., \( k \geq 2 \). Specifically, when considering vertexes \( v_1, v_8, \) and \( v_9 \) (see Figure 6.6), we have

\[
x^{RCC}_1(v_4) = x^{RCC}_1(v_8) = x^{RCC}_1(v_9) = x_a,
\]

\[
x_1(v_1) = x_1(v_8) = x_1(v_9) = x_a,
\]

where \( x_a \) is the state computed using Equation 6.7 with \( k = 1 \) (in this case, equivalent to Equation 6.8), and

\[
C_R(x_1(v_1)) = C_R(x_1(v_8)) = C_R(x_1(v_9)) = \{x_0\},
\]

\[
C(x_1(v_1)) = C(x_1(v_8)) = C(x_1(v_9)) = \{x_0\}.
\]

However, for \( k \geq 2 \), since the label attached to vertexes \( v_4, v_8, \) and \( v_9 \) is the same (“a”), we again have

\[
x^{RCC}_k(v_4) = x^{RCC}_k(v_8) = x^{RCC}_k(v_9) = x_a,
\]

and

\[
C_R(x_k(v_4)) = C_R(x_k(v_8)) = C_R(x_k(v_9)) = \{x_0\}.
\]

Because of that and Equation 6.7, for any value of \( k \) we have \( x^{RCC}(s_1) = x^{RCC}(s_2) \). On the contrary, for CRCC this is not true since, e.g. \( k = 2 \),

\[
C(x_2(v_1)) = x_1 \downarrow \text{in}_\text{set}([v_4]) \cup x_2 \downarrow \text{out}_\text{set}(v_4)
\]

\[
= x_1 \downarrow \{v_2, v_3\} \cup \{x_0\}
\]

\[
= \{x_1(v_2), x_1(v_3), x_1(v_4), x_0\} \cup \{x_0\}
\]
Figure 6.6: Two graphs for which RCC produces state variables with identical values for each value of $k$, while CRCC is able to produce state variables with different values for $k \geq 2$. In particular, CRCC returns different state variables for the two vertexes labeled “a” in the right side graph since their in_set is different, i.e., they have different context.

$$C(x_2(v_8)) = x_1 \downarrow \text{in}_\text{set}(v_8) \cup x_2 \downarrow \text{out}_\text{set}(v_8)$$

$$= x_1 \downarrow \{v_6\} \cup \{x_0\}$$

$$= \{x_1(v_6), x_1(v_8), x_0\} \cup \{x_0\}$$

$$= \{x_1(v_6), x_1(v_8), x_0\}$$

$$C(x_2(v_9)) = x_1 \downarrow \text{in}_\text{set}(v_9) \cup x_2 \downarrow \text{out}_\text{set}(v_9)$$

$$= x_1 \downarrow \{v_7\} \cup \{x_0\}$$

$$= \{x_1(v_7), x_1(v_9), x_0\} \cup \{x_0\}$$

$$= \{x_1(v_7), x_1(v_9), x_0\}$$

and, even if $x_1(v_4) = x_1(v_8) = x_1(v_9)$ (because of the same label “a”), in general we have that $x_1(v_2) \neq x_1(v_3)$ and $x_1(v_6) \neq x_1(v_7)$ (since vertexes $v_2$ and $v_3$, as well as $v_6$ and $v_7$, have different labels), which implies

$$C(x_2(v_4)) \neq C(x_2(v_8)) \neq C(x_2(v_9)).$$

Thus, because of that and Equation 6.9, there exists a CRCC network able to compute different state vectors for $v_4, v_8, v_9$, and consequently also to produce state vectors $x(s_1)$ and $x(s_2)$ that are different. □

Note that, as seen in the proof of Lemma 6.1, CRCC is able to differentiate (to produce state variables with different values) between the two vertexes labeled “a” in the right hand
The above Lemma can be used to state the following

**Theorem 6.8** *The class of functions that can be computed by *RCC* is properly included in the class of functions that can be computed by *CRCC*.*

**Proof.** The class of functions computed by *RCC* is included in the class of functions computed by *CRCC* since Equation 6.9 can be reduced to Equation 6.7 by considering functions $\tau_j(\cdot)$ which do not consider inputs $q^{t+1}x_1(v), \ldots, q^{t-1}x_{j-1}(v)$. Moreover, because of Lemma 6.1, there exist at least one supersource transduction which cannot be computed by *RCC*, because it cannot distinguish between the supersource states of at least two input DPAGs, while *CRCC* can distinguish them and compute such function.

Note that, because of Proposition 6.5 and Proposition 6.6, *CRCC* can also compute transductions that are not strictly causal. Examples of such functions are given in Section 6.7, where we show that *CRCC* is able both to compute and learn them.

Summarizing, the propositions and theorems stated in this section allow us to characterize the computational power of *CRCC* model. In particular, adding the new connections in the model give rise to an increase of the complexity of the model that allows considering, with respect to causal models, the following extension to the treatable classes of target functions:

- extension to *contextual* IO-isomorphic transductions, including the cases where the desired output for a given vertex $v$ depends on vertexes of the input structure (sequences, tree or DPAG) that are not descendants of $v$ (e.g., the desired response for each vertex depending on the whole structure) (Proposition 6.6);

- extension to the class of supersource transductions that involve DPAGs that cannot be computed by causal models (Lemma 6.1);

- while supporting all the function computable by *RCC* models (Theorem 6.8).

However, the properties and theorems cannot characterize tasks where it is unknown the reliability of a causality assumption (for instance in the case of supersource transduction that both *RCC* and *CRCC* can in principle compute). Specific experiments of Section 6.7 are devoted to investigating an example of such cases to assess the possible benefits of *CRCC* approach beyond the above theoretical results.
6.4 Formal Determination of Context

In this section, we characterize the “shape” of the context exploited by a state variable in terms of the set of state variables that contribute directly or indirectly to its determination. The main aim is to give proof of the Theorems stated in Section 6.3. In fact, the relevance of the results obtained in this section is due to the possibility to use them in order to characterize the computational power of the RCC and CRCC models. Moreover, the approach used in this section, which is independent from the specific neural implementation, can easily be applied to other recurrent and recursive models in order to compare them from a computational point of view.

We will use in the following the descendants and dot operators defined in Section 6.3. Specifically, we are interested in the associative properties of the above operators:

**Lemma 6.2 (Associative rules)** For any couple of vertexes sets \( V, Z \subseteq \text{vert}(G) \), the following equations hold

\[ \downarrow V \cup \downarrow Z = \downarrow (V \cup Z) \]

and

\[ x_j.\downarrow V \cup x_j.\downarrow Z = x_j.\downarrow (V \cup Z) \]

**Proof.** It directly follows from the respective definitions of “descendants” operator (Definition 6.1) and “dot” operator (Definition 6.2).

Let us state some basic results that will be used to derive the main theorems concerning the context window of the CRCC and RCC models.

**Lemma 6.3** \( \mathcal{C}(x_0) = \emptyset \)

**Proof.** Clearly, since the (constant value) state \( x_0 \) is not the result of a computation performed on other states, the set of states contributing to its determination is an empty set.

All the results reported in the following will concern the CRCC model. Referring to Equation 6.8 it is now possible to derive the following

**Lemma 6.4** Given a DPAG \( G \), for any vertex \( v \in \text{vert}(G) \) the following equation holds

\[ \mathcal{C}(x_1(v)) = x_1.\downarrow \text{out}_\text{set}(v) \].
Proof. It can be proved by induction on the partial order of vertexes. By definition of context window (direct and indirect contribution), and by Equation 6.8 for any \( v \) in \( \text{vert}(G) \), the context window for the state variable \( x_1(v) \) can be expressed as

\[
C(x_1(v)) = \bigcup_{i=1}^{\text{out}} q_i^{-1}x_1(v) \cup \bigcup_{i=1}^{\text{in}} C(q_i^{-1}x_1(v))
\]  

(6.13)

**Base Case:** \( v \) is a leaf of \( G \), i.e. a vertex without outgoing edges.

By definition of \( \text{out}_i \), \( \forall i \in [1, \ldots, \text{out}] \) \( \text{out}_i(v) = \emptyset \), thus by definition of “descendants” operator \( x_1 \cdot \downarrow \text{out}_i(v) = x_1.\emptyset = \{x_0\} \).

Moreover, by definition of extended shift operator, \( \forall i \in [1, \ldots, \text{out}] \) \( q_i^{-1}x_1(v) = x_0 \), and by Lemma 6.3, \( C(q_i^{-1}x_1(v)) = C(x_0) = \emptyset \). This proves that

\[
C(x_1(v)) = x_1. \downarrow \text{out}_i(v) = \{x_0\}
\]

for any leaf \( v \) of the data structure \( G \).

**Inductive Step:** \( v \) is an internal vertex of \( G \), i.e. a vertex with at least one outgoing edge.

Assume that

\[
\forall u \in \text{out}_i(v) \quad C(x_1(u)) = x_1. \downarrow \text{out}_i(u) \quad \text{(Inductive Hypothesis)}
\]

By definition of generalized shift operator,

\[
\bigcup_{i=1}^{\text{out}} q_i^{-1}x_1(v) = \bigcup_{i=1}^{\text{out}} x_1(\text{out}_i(v)) = \bigcup_{u \in \text{out}_i(v)} x_1(u)
\]

and

\[
\bigcup_{i=1}^{\text{out}} C(q_i^{-1}x_1(v)) = \bigcup_{u \in \text{out}_i(v)} C(x_1(u))
\]

thus, Equation 6.13 becomes

\[
C(x_1(v)) = \bigcup_{u \in \text{out}_i(v)} [x_1(u) \cup C(x_1(u))]
\]

By the induction hypothesis,

\[
C(x_1(v)) = \bigcup_{u \in \text{out}_i(v)} [x_1(u) \cup x_1. \downarrow \text{out}_i(u)]
\]

Since \( x_1(u) = x_1.\{u\} \), and by the associative rule for “dot” (Lemma 6.2), we have

\[
C(x_1(v)) = \bigcup_{u \in \text{out}_i(v)} [\{u\} \cup x_1. \downarrow \text{out}_i(u)]
\]
and by the definition of “descendants” (Definition 6.1), the thesis follows:

\[ C(x_1(v)) = \bigcup_{u \in \text{out}\_set(v)} x_1. \downarrow u = x_1. \downarrow \text{out}\_set(v) \]

where in the last step we have again used the associative rule for “dot”.

\[ \square \]

**Corollary 6.1** Given a DPAG \( G \), for any couple of vertexes \( u, v \in \text{vert}(G) \) connected by a path from \( v \) to \( u \) then

\[ C(x_1(u)) \subseteq C(x_1(v)). \]

**Proof.** It follows directly from Lemma 6.4, from the definitions of “descendants” operator and context window, and from the fact that the graph is acyclic. \( \square \)

Referring to Equation 6.9 it is now possible to derive the following lemmata, which are then used to prove Theorem 6.4

**Lemma 6.5** Given a DPAG \( G \), for any vertex \( v \in \text{vert}(G) \) the following equation holds

\[ C(x_2(v)) = x_1. \downarrow \text{in}\_set(\downarrow v) \cup x_2. \downarrow \text{out}\_set(v). \quad (6.14) \]

**Proof.** It can be proved by induction on the partial order of vertexes.
By definition of context window, and by Equation 6.9 for any \( v \in \text{vert}(G) \) the context window for the state variable \( x_2(v) \) can be expressed by

\[ C(x_2(v)) = \bigcup_{i=1}^{\text{out}} \left[ q_i^{-1} x_2(v) \cup C(q_i^{-1} x_2(v)) \right] \cup \bigcup_{i=1}^{\text{out}} \left[ q_i^{-1} x_1(v) \cup C(q_i^{-1} x_1(v)) \right] \cup \bigcup_{i=1}^{\text{in}} \left[ q_i^{+1} x_1(v) \cup C(q_i^{+1} x_1(v)) \right] \quad (6.15) \]

**Base Case:** \( v \) is a leaf of \( G \), i.e. a vertex without outgoing edges.

By the definitions of \( \text{out}\_set, \text{in}\_set, \) and “descendants” operator it follows that \( x_2. \downarrow \text{out}\_set(v) = x_2. \emptyset = \{x_0\} \), and \( x_1. \downarrow \text{in}\_set(\downarrow v) = x_1. \downarrow \text{in}\_set(v) \). Thus Equation 6.14 becomes

\[ C(x_2(v)) = x_1. \downarrow \text{in}\_set(v) \cup \{x_0\}. \]
By definition of generalized shift operators, \( q_i^{-1} x_2(v) = x_2.\text{out}_i(v) = \{x_0\} \), and \( q_i^{+1} x_1(v) = x_1.\text{in}_i(v) \). Moreover, by Lemma 6.3, \( \mathcal{C}(q_i^{-1} x_2(v)) = \mathcal{C}(x_2.\text{out}_i(v)) = \mathcal{C}\{x_0\} = \emptyset \), and by Lemma 6.4, \( \mathcal{C}(q_i^{+1} x_1(v)) = \mathcal{C}(x_1.\text{in}_i(v)) = x_1. \downarrow \text{out}_i(\text{in}_i(v)) \). Thus, by the associative rule on “dot” and by definition of “descendants” operator, Equation 6.15 becomes

\[
\mathcal{C}(x_2(v)) = \{x_0\} \cup x_1.\text{in}_i(v) \cup x_1. \downarrow \text{out}_i(\text{in}_i(v)) = x_1. \downarrow \text{in}_i(v) 
\]

The thesis follows for all leaves of the structure.

**Inductive Step:** \( v \) is an internal vertex of \( G \), i.e. a vertex with at least one outgoing edge. Assume that \( \forall u \in \text{out}_i(v) \)

\[
\mathcal{C}(x_2(u)) = x_1. \downarrow \text{in}_i(\downarrow u) \cup x_2. \downarrow \text{out}_i(u) \quad \text{(Inductive Hypothesis).}
\]

By definition of generalized shift operator and by Lemma 6.4, Equation 6.15 becomes

\[
\mathcal{C}(x_2(v)) = \bigcup_{u \in \text{out}_i(v)} [x_2(u) \cup \mathcal{C}(x_2(u))] \cup \bigcup_{u \in \text{out}_i(v)} [x_1(u) \cup x_1. \downarrow \text{out}_i(u)] \cup \bigcup_{u \in \text{in}_i(v)} [x_1(u) \cup x_1. \downarrow \text{out}_i(u)] 
\]

By the associative rule for “dot” and the definition of “descendants” operator

\[
\mathcal{C}(x_2(v)) = \bigcup_{u \in \text{out}_i(v)} [x_2(u) \cup \mathcal{C}(x_2(u))] \cup \bigcup_{u \in \text{out}_i(v)} [x_1(u) \cup x_1. \downarrow \text{out}_i(u)] \cup \bigcup_{u \in \text{in}_i(v)} [x_1(u) \cup x_1. \downarrow \text{in}_i(v)] 
\]

By the inductive hypothesis and by the definition of “descendants” operator

\[
\mathcal{C}(x_2(v)) = \bigcup_{u \in \text{out}_i(v)} [x_2(u) \cup x_2. \downarrow \text{out}_i(u) \cup x_1. \downarrow \text{in}_i(\downarrow u)] \cup x_1. \downarrow \text{out}_i(v) \cup x_1. \downarrow \text{in}_i(v) 
\]

By the associative rules and by definition of \( \text{in}_i \)

\[
\mathcal{C}(x_2(v)) = x_2. \downarrow \text{out}_i(v) \cup x_1. \downarrow [\text{in}_i(v) \cup \text{in}_i(\downarrow \text{out}_i(v))] \cup x_1. \downarrow \text{out}_i(v) 
\]

\[
\mathcal{C}(x_2(v)) = x_2. \downarrow \text{out}_i(v) \cup x_1. \downarrow \text{in}_i(v) \cup \downarrow \text{out}_i(v) \cup x_1. \downarrow \text{out}_i(v) 
\]

\[
\mathcal{C}(x_2(v)) = x_2. \downarrow \text{out}_i(v) \cup x_1. \downarrow \text{in}_i(v) \cup x_1. \downarrow \text{out}_i(v) 
\]
Since for any \( v \in \text{vert}(G) \) it holds \( \downarrow \text{out.set}(v) \subseteq \downarrow v \subseteq \downarrow \text{in.set}(v) \) the thesis follows:

\[
\mathcal{C}(x_2(v)) = x_2 \cdot \downarrow \text{out.set}(v) \cup x_1 \cdot \downarrow \text{in.set}(v).
\]

\[\square\]

**Lemma 6.6** Given a DPAG \( G \), for any vertex \( v \in \text{vert}(G) \) and for any \( k \geq 2 \), it holds

\[
\mathcal{C}(x_k(v)) \supseteq \mathcal{C}(x_{k-1}(v)).
\]

**Proof.** By definition of context window and by Equation 6.9, for any \( v \) in \( \text{vert}(G) \) the context window for the state variable \( x_k(v) \) can be expressed by

\[
\mathcal{C}(x_k(v)) = \bigcup_{i=1}^{\text{out}} [x_1(v) \cup \mathcal{C}(q_i^{-1}x_1(v)) \cup \cdots \cup q_i^{-1}x_k(v) \cup \mathcal{C}(q_i^{-1}x_k(v))]
\]

by the definition of generalized shift operator

\[
\mathcal{C}(x_k(v)) = \bigcup_{u \in \text{out.set}(v)} [x_1(u) \cup \mathcal{C}(x_1(u)) \cup \cdots \cup x_k(u) \cup \mathcal{C}(x_k(u))]
\]

and reorganizing the unions of sets in a different way

\[
\mathcal{C}(x_k(v)) = \bigcup_{u \in \text{out.set}(v)} [x_1(u) \cup \mathcal{C}(x_1(u)) \cup \cdots \cup x_{k-1}(u) \cup \mathcal{C}(x_{k-1}(u))]
\]

which can be rewritten as

\[
\mathcal{C}(x_k(v)) = \mathcal{C}(x_{k-1}(v)) \cup \mathcal{C}(x_k(\text{out.set}(v))) \cup \mathcal{C}(x_{k-1}(\text{in.set}(v))) \cup x_k(\text{out.set}(v)) \cup x_{k-1}(\text{in.set}(v))
\]

hence

\[
\mathcal{C}(x_k(v)) \supseteq \mathcal{C}(x_{k-1}(v)).
\]

\[\square\]
We are now ready to (re)state and prove the main theorem (Theorem 6.4), i.e., the expression which explicitly defines the CRCC context for any state variable:

**Theorem 6.9 (i.e. Theorem 6.4)** Given a DPAG $G$, for any vertex $v \in \text{vert}(G)$, and for any index $k \geq 2$, the following equation holds

$$
\mathcal{C}(x_k(v)) = \bigcup_{i=1}^{k-1} x_i . (\downarrow \text{in_set})^{k-i} (\downarrow v) \cup x_k . \downarrow \text{out_set}(v). \tag{6.16}
$$

**Proof.** The theorem can be proved by induction on the partial order of vertexes, and on the order of indexes of variables.

By definition of context window, by Equation 6.9, and by definition of generalized shift operator, for any $v$ in $\text{vert}(G)$ the context window for the state variable $x_k(v)$ can be expressed by

$$
\mathcal{C}(x_k(v)) = \bigcup_{u \in \text{out_set}(v)} [x_1(u) \cup \mathcal{C}(x_1(u)) \cup \cdots \cup x_k(u) \cup \mathcal{C}(x_k(u))] \cup
\bigcup_{u \in \text{in_set}(v)} [x_1(u) \cup \mathcal{C}(x_1(u)) \cup \cdots \cup x_{k-1}(u) \cup \mathcal{C}(x_{k-1}(u))]
$$

Applying lemma 6.6 (i.e., $\forall i < k, \mathcal{C}(x_k(u)) \supseteq \mathcal{C}(x_i(u))$)

$$
\mathcal{C}(x_k(v)) = \bigcup_{u \in \text{out_set}(v)} [x_1(u) \cup \cdots \cup x_k(u)] \cup \bigcup_{u \in \text{out_set}(v)} \mathcal{C}(x_k(u)) \cup
\bigcup_{u \in \text{in_set}(v)} [x_1(u) \cup \cdots \cup x_{k-1}(u)] \cup \bigcup_{u \in \text{in_set}(v)} \mathcal{C}(x_{k-1}(u))
$$

$$
= \bigcup_{i=1}^{k-1} x_i . \text{out_set}(v) \cup \mathcal{C}(x_{k-1} . \text{out_set}(v)) \cup \bigcup_{i=1}^{k-1} x_i . \text{in_set}(v) \cup \mathcal{C}(x_{k-1} . \text{in_set}(v)) \tag{6.17}
$$

**Base Case:** $v$ is a leaf of $G$, i.e. a vertex without outgoing edges.

For $k = 2$, Equation 6.16 reduces to Equation 6.14 (the base case for which the theorem has been proved). Now, let assume that $\forall j, \ 2 \leq j \leq k - 1$

$$
\mathcal{C}(x_j(v)) = \bigcup_{i=1}^{j-1} x_i . (\downarrow \text{in_set})^{j-i} (\downarrow v) \cup x_j . \downarrow \text{out_set}(v) \quad \text{(Inductive Hypothesis)} \tag{6.18}
$$

Since the vertex $v$ is a leaf, Equation 6.17 can be reduced to

$$
\mathcal{C}(x_k(v)) = \{x_0\} \cup \emptyset \cup \bigcup_{i=1}^{k-1} x_i . \text{in_set}(v) \cup \mathcal{C}(x_{k-1} . \text{in_set}(v))
$$
and, applying the inductive hypothesis 6.18, it becomes

\[
C(x_k(v)) = \bigcup_{i=1}^{k-1} x_i.\text{in.set}(v) \cup \\
\bigcup_{i=1}^{k-2} \big( x_i.\text{(in.set)}^{k-1-i} \cup \text{in.set}(v) \big) \cup x_{k-1}.\text{out.set(in.set}(v))
\]

which can be easily rewritten as

\[
C(x_k(v)) = x_{k-1}.\text{in.set}(v) \cup \text{out.set(in.set}(v)) \cup \\
\bigcup_{i=1}^{k-2} [x_i.\text{in.set}(v) \cup x_i.\text{(in.set)}^{k-1-i}(v)]
\]

By definition of “descendants” operator, and because of the fact that for any \( v \in \text{vert}(G) \) and for any \( j \geq 1 \) it trivially holds \( \text{in.set}(v) \subseteq \text{in.set}(v) \subseteq (\downarrow \text{in.set})^j(v) \), it follows that

\[
C(x_k(v)) = x_{k-1}.\downarrow \text{in.set}(v) \cup \bigcup_{i=1}^{k-2} x_i.\text{(in.set)}^{k-1-i}(v)
\]

\[
= \bigcup_{i=1}^{k-1} x_i.\text{(in.set)}^{k-1-i}(v)
\]

On the other hand, since \( v \) is a leaf, notice that \( v = \downarrow v \) and \( x_k.\downarrow \text{out.set}(v) \) can be reduced to \( \{x_0\} \), thus Equation 6.16 can be rewritten as

\[
C(x_k(v)) = \bigcup_{i=1}^{k-1} x_i.\text{(in.set)}^{k-1-i}(v)
\]

Which proves the correctness of the assertion for all leaves of \( G \) and for any \( k \geq 2 \)

**Inductive Step:** \( v \) is an internal vertex of \( G \). i.e. a vertex with at least one outgoing edge.

Assume that

\[
C(x_j(u)) = \bigcup_{i=1}^{j-1} x_i.\text{(in.set)}^{j-i}(\downarrow u) \cup x_j.\text{out.set}(u) \quad \text{(Inductive Hypothesis)}
\]

(6.19)

holds \( \forall u \in \text{vert}(G) \) if \( 2 \leq j \leq k - 1 \), and \( \forall u \in \text{out.set}(v) \) if \( j = k \), then, Equation 6.17 becomes

\[
C(x_k(v)) = \bigcup_{i=1}^{k} x_i.\text{out.set}(v) \cup C(x_k.\text{out.set}(v)) \cup \bigcup_{i=1}^{k-1} x_i.\text{in.set}(v) \cup \\
\bigcup_{i=1}^{k-2} \big( x_i.\text{(in.set)}^{k-1-i}(\downarrow \text{in.set}(v)) \cup x_{k-1}.\downarrow \text{out.set(in.set}(v)) \big)
\]
\[
\begin{align*}
&= \bigcup_{i=1}^{k} x_i.\text{out\_set}(v) \cup C(x_k.\text{out\_set}(v)) \cup \bigcup_{i=1}^{k-1} x_i.\text{in\_set}(v) \cup \\
&\quad \left[ \bigcup_{i=1}^{k-2} x_i.\downarrow \text{in\_set}^{k-i}(v) \cup x_{k-1}.\downarrow \text{out\_set}(\text{in\_set}(v)) \right]
\end{align*}
\]

By the associative properties of “dot” and “descendants” operators

\[
C(x_k(v)) = \bigcup_{i=1}^{k} x_i.\text{out\_set}(v) \cup C(x_k.\text{out\_set}(v)) \cup \bigcup_{i=1}^{k-2} x_i.\text{in\_set}(v) \cup \\
\quad \bigcup_{i=1}^{k-2} x_i.\downarrow \text{in\_set}^{k-i}(v) \cup x_{k-1}.\downarrow \text{in\_set}(v)
\]

\[
= \bigcup_{i=1}^{k} x_i.\text{out\_set}(v) \cup C(x_k.\text{out\_set}(v)) \cup \bigcup_{i=1}^{k-2} x_i.\text{in\_set}(v) \cup \\
\quad \bigcup_{i=1}^{k-2} x_i.\downarrow \text{in\_set}^{k-i}(v) \cup x_{k-1}.\downarrow \text{in\_set}(v)
\]

\[
= \bigcup_{i=1}^{k} x_i.\text{out\_set}(v) \cup C(x_k.\text{out\_set}(v)) \cup \bigcup_{i=1}^{k-2} x_i.\text{in\_set}(v) \cup \\
\quad \bigcup_{i=1}^{k-2} x_i.\downarrow \text{in\_set}^{k-i}(v)
\]

Applying again the induction hypothesis 6.19 on \(C(x_k.\text{out\_set}(v))\)

\[
C(x_k(v)) = \bigcup_{i=1}^{k} x_i.\text{out\_set}(v) \cup \\
\quad \bigcup_{i=1}^{k-1} x_i.\downarrow \text{in\_set}^{k-i}(\downarrow \text{out\_set}(v)) \cup x_k.\downarrow \text{out\_set}(\text{out\_set}(v)) \cup \\
\quad \bigcup_{i=1}^{k-2} x_i.\text{in\_set}(v) \cup \bigcup_{i=1}^{k-1} x_i.\downarrow \text{in\_set}^{k-i}(v)
\]

By the associative properties of “dot” and “descendants” operators applied to \(x_i.\downarrow \text{in\_set}^{k-i}\) and \(x_k.\text{out\_set}\)

\[
C(x_k(v)) = \bigcup_{i=1}^{k-1} x_i.\text{out\_set}(v) \cup \bigcup_{i=1}^{k-2} x_i.\text{in\_set}(v) \cup \\
\quad \bigcup_{i=1}^{k-1} x_i.\downarrow \text{in\_set}^{k-i}[\downarrow \text{out\_set}(v) \cup v] \cup \\
\quad x_k.\downarrow \text{out\_set}(\text{out\_set}(v)) \cup x_k.\text{out\_set}(v) \cup \\
\quad \bigcup_{i=1}^{k-1} x_i.\text{out\_set}(v) \cup \bigcup_{i=1}^{k-2} x_i.\text{in\_set}(v) \cup
\]
\[
\bigcup_{i=1}^{k-1} x_i \cdot (\downarrow \text{in_set})^{k-i} (\downarrow v) \cup x_k \cdot \downarrow \text{out_set}(v)
\]

Since for any \(v \in \text{vert}(G)\) and for any \(j \geq 1\) it trivially holds that

\[
\text{out_set}(v) \subseteq \downarrow \text{in_set}(v) \subseteq (\downarrow \text{in_set})^j (v) \subseteq (\downarrow \text{in_set})^j (\downarrow v),
\]

and similarly

\[
\text{in_set}(v) \subseteq \downarrow \text{in_set}(v) \subseteq (\downarrow \text{in_set})^j (v) \subseteq (\downarrow \text{in_set})^j (\downarrow v),
\]

the thesis follows:

\[
\mathcal{C}(x_k(v)) = \bigcup_{i=1}^{k-1} x_i \cdot (\downarrow \text{in_set})^{k-i} (\downarrow v) \cup x_k \cdot \downarrow \text{out_set}(v)
\]

\(\blacksquare\)

For the sake of comparison, let us now consider the “context window” for the RCC model (Equation 6.7). The following theorem holds:

**Theorem 6.10 (i.e. Theorem 6.7)** Given a DPAG \(G\), for any vertex \(v \in \text{vert}(G)\), and for any index \(k \geq 1\), the following equation holds

\[
\mathcal{C}_{RCC}(x_k(v)) = \bigcup_{i=1}^{k} x_i \cdot \downarrow \text{out_set}(v).
\]  

(6.20)

**Proof.** For \(k = 1\) the proof is given by Lemma 6.4, since Equation 6.7 is equivalent to Equation 6.8. For \(k > 1\) the proof is similar to the proof given for Theorem 6.4 where, this time,

\[
\mathcal{C}_{RCC}(x_k(v)) = \bigcup_{u \in \text{out_set}(v)} [x_1(u) \cup \mathcal{C}_{RCC}(x_1(u))] \cup \cdots \cup x_k(u) \cup \mathcal{C}_{RCC}(x_k(u))]
\]

\(\blacksquare\)
6.5 Contextual Recursive Cascade Correlation

According to schema introduced in Chapter 4 to derive Neural Structure Transductions we can now propose a neural realization of the CRCC model. The state vector over the current vertex \( v \) can be computed as

\[
x(v) = \sigma\left(\text{net}(v)\right) = [\sigma(\text{net}_1(v)), \ldots, \sigma(\text{net}_m(v))]^t
\]

\[
\text{net}(v) = \begin{bmatrix}
w_1^t \\
w_2^t \\
w_3^t \\
\vdots \\
w_m^t \\
\end{bmatrix} l(v) + \\
\begin{bmatrix}
\sum_{i=1}^{2} \tilde{w}_1 i q^{-1} x_i(v) \\
\sum_{i=1}^{3} \tilde{w}_2 i q^{-1} x_i(v) \\
\vdots \\
\sum_{i=1}^{m} \tilde{w}_m i q^{-1} x_i(v) \\
0 \\
\sum_{i=1}^{2} \tilde{w}_2 i q^{-1} x_i(v) \\
\vdots \\
\sum_{i=1}^{m-1} \tilde{w}_m i q^{-1} x_i(v)
\end{bmatrix}
\]  

(6.23)

where \( \sigma(\cdot) \) is a sigmoidal (or radial) activation function; \( w_k \) is the label weight vector associated to the \( k \)-th hidden unit; \( \tilde{w}_k i, \) with \( 1 \leq i \leq k \leq m \), are the \( k \) weight vectors associated with the \( k \)-th hidden unit, each one associated to the “past” information arriving through the outgoing edges of the current vertex from the \( i \)-th (frozen, for \( i < k \)) hidden unit; \( \tilde{w}_k i, \) with \( 1 \leq i < k \leq m \), are the \( k-1 \) weight vectors associated to the \( k \)-th hidden unit, each one associated to the “future” information arriving through the ingoing edges of the current vertex from the \( i \)-th (frozen, for \( i < k \)) hidden unit.

Note that the first component (6.22) corresponds to the contribution of the “present” information, i.e., the label attached to \( v \), the second component (6.23) corresponds to the contribution of the “past” information coming from descendants of \( v \), while the last component (6.24) corresponds to the contribution of the “future” information coming from the subgraphs with supersource in the set in_set(\( v \)). Given an input structure, the network output function \( g() \) (see Equation 6.6) is implemented by one or more standard neurons

\[
y(v) = \Phi(x(v), W) = \sigma(\text{mx}(v))
\]

(6.25)

\[\text{Notwithstanding the vectorial representation, here the computation has to be understood in a data-flow fashion, i.e., the component } i \text{-th of the vector can be computed only when the component } i - 1 \text{-th has already been computed.}\]
where $\boldsymbol{m}$ is the output state weight matrix, which is increased in size each time a new hidden unit is added.

Learning is performed as in Recursive Cascade Correlation (based on learning algorithm for Cascade Correlation family), see Section 4.3, by interleaving the minimization of the total error function (LMS) and the maximization of the (non-normalized) correlation $S$, i.e. the covariance, of the new inserted hidden unit $k$ with the residual error, i.e computing $\frac{\partial S}{\partial w}$ for each $w$ of hidden units. The main difference with respect to presentation in Section 4.3 is in the calculation of the derivatives of the output of the current hidden unit. According to equation 6.22 we can determine the following:

$$\frac{\partial x_k(v)}{\partial w_k} = \sigma' \left( l(v) + \frac{\partial q^{-1}x_k(v)}{\partial w_k} \tilde{w}_{kk} \right)$$

(6.26)

$$\frac{\partial x_k(v)}{\partial \tilde{w}_{ki}} = \sigma' \left( q^{-1}x_i(v) + \frac{\partial q^{-1}x_k(v)}{\partial \tilde{w}_{ki}} \tilde{w}_{kk} \right)$$

(6.27)

$$\frac{\partial x_k(v)}{\partial \tilde{w}_{ki}} = \sigma' \left( q^{+1}x_i(v) + \frac{\partial q^{-1}x_k(v)}{\partial \tilde{w}_{ki}} \tilde{w}_{kk} \right)$$

(6.28)

where $\sigma'$ is the first derivative of the activation function $\sigma(\cdot)$. Note that Equations 6.26, and 6.27 are the same used in standard RCC for structured data (Section 4.3), while Equation 6.28 is added so to include also contextual (“future” in sequences) information from frozen units. The above equations are recurrent and can be computed by observing that for all the leaves of the structured data (all vertexes with null out-degree) Equation 6.26 becomes $\frac{\partial x_k(v)}{\partial w_k} = \sigma' l(v)$, the derivatives for Equation 6.27 are null, and Equation 6.28 reduces to $\frac{\partial x_k(v)}{\partial w_{ki}} = \sigma' q^{+1}x_i(v)$. Consequently, we only need to store the output values of the unit and its derivatives for each component of the structure.

### 6.6 Special Cases and Extensions

It should be stressed that when considering a structured domain with maximum in-degree and out-degree equal to 1, e.g. temporal sequences, Equations (6.3)-(6.4) reduce to the following:

$$q^{-1}x(t) = x(t - 1).$$

(6.29)

$$q^{+1}x(t) = x(t + 1).$$

(6.30)

In this framework, RCC reduces to the Recurrent Cascade Correlation model [46], and the CRCC model becomes the **Bi-causal Recurrent Cascade Correlation** [124], where Equation 6.9 reduces to

$$x_j(t) = \tau_j(l(t), x_1(t - 1), \ldots, x_j(t - 1), x_1(t + 1), \ldots, x_{j-1}(t + 1)).$$

(6.31)
An example of encoding network and the formulation of the context shape for the Bi-causal Recurrent Cascade Correlation are reported in Example 6.1.

On the other hand, the architecture described in Section 6.2 can of course be extended in several ways. For example, we can consider the composition of shift operators:

\[ q^{-p} = \underbrace{q^{-1}q^{-1}\cdots q^{-1}}_{p \text{ times}} \]

and

\[ q^{+p} = \underbrace{q^{+1}q^{+1}\cdots q^{+1}}_{p \text{ times}} \]

where, given \( e \in \{-1, +1\} \), we have the following rule

\[ q^e \begin{bmatrix} x_{11} & \cdots & x_{1t} \\ \vdots & \ddots & \vdots \\ x_{r1} & \cdots & x_{rt} \end{bmatrix} = \begin{bmatrix} q^ex_{11} & \cdots & q^ex_{1t} \\ \vdots & \ddots & \vdots \\ q^ex_{r1} & \cdots & q^ex_{rt} \end{bmatrix}. \]

So it is possible to extend Equations (6.8) and (6.9) by introducing shift operators with \( p > 1 \) and/or \( p < -1 \), or combinations of them\(^3\). Moreover, when considering the neural realization, both \( \tau() \) and \( g() \) can be implemented by a multilayer network instead of a single layer of neurons. For all of these extensions it is not difficult to see that gradients can be easily computed and new suitable learning rules can be devised.

### 6.7 Experimental Results

In the following we report the results obtained with the Contextual Recursive Cascade Correlation model applied to regression tasks in different structured domains involving sequences and trees, respectively.

For the sequence domain we have decided to use artificial data sets in order to have the control of the causality/contextual conditions in evaluating the ability of CRCC in learning contextual mappings. The aim of our experiments in this domain is to show that while RCC is unable to learn a contextual mapping, CRCC can do it. Furthermore, through the use of carefully controlled experiments, we show that this ability of the CRCC does not impair the prediction ability of the model under strict causality conditions.

The structured domain involving trees, on the other hand, concerns a real-world problem in Chemistry involving supersource transductions, i.e., the Quantitative Structure Property Relationship (QSPR) analysis of alkanes (see Section 2.3.4). Note that, this experiment allows us to compare RCC and CRCC on a fair ground, since: \( i \) RCC is able to generate different state vectors for distinct trees; \( ii \) the general form of the target

---

\(^3\)Please, note that \( q^{-1} \) and \( q^{+1} \) are not commutative.
function is unknown and so it is not clear whether the causality assumption holds. Hence, these are interesting cases for which the properties and the theorems derived above cannot be directly applied to show the different behavior of the two models, while the experiments allow us to complete the comparison.

In all the experiments described below, both for RCC and CRCC, we have adopted the following parametrization for the output function (one single linear output)

\[ y(v) = m \cdot x(v). \]

### 6.7.1 Learning Contextual Mappings for Sequences

For this domain, we have considered regression problems. Different sets of randomly generated artificial sequences have been produced. The training sets are composed of 200 sequences while the test sets are composed of 100 sequences. Of course, in this case the maximum in-degree and out-degree is 1. The sequences, of length between 5 and 20, are composed of symbols in the alphabet \( \Sigma = \{a, \ldots, j\} \). Each symbol is selected according to a uniform distribution over the alphabet and it is coded as a 10-bit string, with one specific bit turned on \((+1)\) and all others turned off \((-1)\). Moreover, in order to define the target, a function \( v: \Sigma \rightarrow \{0, 0.1, \ldots, 0.9\} \) is defined (i.e., \( v(a) = 0, \ldots, v(j) = 0.9 \)). Different prediction tasks were obtained by defining different target functions for each element of a sequence.

The first target function, strictly dependent on the next position in the sequence, is defined as in the following

\[ \text{target}_{1f}(t) = v(s_i(t + 1)), \quad (6.32) \]

where \( s_i \) is the \( i \)-th sequence, and \( s_i(t + 1) \) returns the sequence element in position \( t + 1 \). For comparison with RCC we have also used the following causal target function (which depends only on the past element):

\[ \text{target}_{1p}(t) = v(s_i(t - 1)). \quad (6.33) \]

Other target functions involving the average on a window of size 4 have been used:

\[ \text{target}_{4f}(t) = \frac{\sum_{j=0}^{3} v(s_i(t + j))}{4}, \quad (6.34) \]
\[ \text{target}_{4p}(t) = \frac{\sum_{j=0}^{3} v(s_i(t - j))}{4}. \quad (6.35) \]

Finally, we have defined a moving average target over the future:

\[ \text{target}_{ma}(t) = \frac{\text{target}_{ma}(t + 1) + v(s_i(t))}{2}. \quad (6.36) \]

Clearly, both \( \text{target}_{1p} \) and \( \text{target}_{4p} \) define computational tasks for which the causality assumption fully holds. On the other hand, the causality assumption does not hold when using \( \text{target}_{1f}, \text{target}_{4f}, \) and \( \text{target}_{ma} \), which can be considered “contextual” functions.
We performed several training trials with all the above defined target functions. Here we report examples of specific trials which are representative of the behavior of CRCC and RCC.

An example of the results obtained by CRCC and RCC over $target_{1f}$ are given in Figure 6.7(a-b). The performance of a theoretical null model$^4$ for the test set is shown as well. Note that, as expected, the RCC is not able to improve over the null model. The difficulty of RCC to deal with the prediction task is also evident from the increase in the maximum error corresponding to the increase of the number of hidden units into the network (see Figure 6.7(b)). On the contrary, CRCC is able to decrease the maximum error along with the increase in the number of hidden units.

As shown in Figure 6.8(a), when considering $target_{1p}$ (i.e., the causality assumption holds) the results obtained by CRCC are comparable with those obtained by RCC. This shows that the CRCC’s ability to use contextual information does not impair the performance of the model under strict causal conditions. A confirmation of this behavior is given when experimenting with $target_{1p}$ (see Figure 6.8(a)).

Finally, CRCC seems to be able to cope well with longer dependencies in the future, as encoded in $target_{4f}$ (see Figure 6.8(b)), as well as with the moving average over the future, i.e., $target_{ma}$ (see Figure 6.9).

In all the experiments we let the training to insert many more hidden units than necessary for solving the regression problems, however, it can be noticed that no overfitting was observed. This is due to the adoption of a regularization strategy called $i$-strategy (see Section 5.4).

---

$^4$See Section 4.4: the null model is obtained by computing the expected value for the target over the training set.
Figure 6.7: Mean (a) (Top) and maximum (b) (Bottom) error of CRCC and RCC for 
*target* $1_f$. Note that the mean error of RCC is near the mean error of the null model, and
the maximum error diverges.
Figure 6.8: Mean error for RCC and CRCC for $\text{target}_{1p}$ (a) (Top), $\text{target}_{4p}$ and $\text{target}_{4f}$ (b) (Bottom).
6. CONTEXTUAL PROCESSING OF STRUCTURED DOMAINS BY RCC

6.7.2 QSPR Analysis of Alkanes

Here we consider a regression problem on a structured domain involving chemical compounds represented as trees. The problem consists in the prediction of the boiling point for a group of acyclic hydrocarbons (alkanes). The problem has been introduced in Section 2.3.4 as a typical example of benchmark in testing Quantitative Structure-Property Relationship (QSPR) models. For this problem, the causal model (RCC) has been proved to be competitive with respect to ad-hoc techniques: in fact, the results obtained by RCC compares favorably versus the approach proposed by Cherqaoui et al. [30]. They apply a multilayer feedforward neural network to a vectorial representation of alkanes able to retain the structural information which is known to be relevant to the prediction of the boiling point. An extended presentation of the approach of Cherqaoui et al. and the one based on RCC, with the first comparison of the two, is described in Chapter 10 (and in [21]), i.e. in PART III (Applications).

Here, this task has been selected in order to have a direct comparison of the new approach (CRCC) versus the standard causal model (RCC) in a real-world application. Since the target property is related to global characteristics of the structures, such as the molecular size and the molecular shape, we believe that a model able to capture contextual information should improve the performance on this task.

Moreover, these experiments allow us to investigate the coherence of the causality assumption, and the effect of its relaxation, on a real-world application.

We recall from Subsection 2.3.4 that prediction of the boiling point yields to a regres-
6.7. EXPERIMENTAL RESULTS

<table>
<thead>
<tr>
<th></th>
<th>CRCC</th>
<th>RCC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Absolute Test Error</td>
<td>Absolute Test Error</td>
</tr>
<tr>
<td>Mean</td>
<td>Variance</td>
<td>Mean</td>
</tr>
<tr>
<td>fold-01</td>
<td>2.40</td>
<td>0.080</td>
</tr>
<tr>
<td>fold-02</td>
<td>1.85</td>
<td>0.200</td>
</tr>
<tr>
<td>fold-03</td>
<td>2.63</td>
<td>0.233</td>
</tr>
<tr>
<td>fold-04</td>
<td>2.44</td>
<td>0.173</td>
</tr>
<tr>
<td>fold-05</td>
<td>1.70</td>
<td>0.120</td>
</tr>
<tr>
<td>fold-06</td>
<td>2.26</td>
<td>0.116</td>
</tr>
<tr>
<td>fold-07</td>
<td>2.66</td>
<td>0.388</td>
</tr>
<tr>
<td>fold-08</td>
<td>2.95</td>
<td>0.026</td>
</tr>
<tr>
<td>fold-09</td>
<td>3.97</td>
<td>0.231</td>
</tr>
<tr>
<td>fold-10</td>
<td>2.78</td>
<td>1.768</td>
</tr>
<tr>
<td><strong>Total Average</strong></td>
<td><strong>2.56</strong></td>
<td><strong>0.639</strong></td>
</tr>
<tr>
<td>Average # of Hidden Units</td>
<td>137.8</td>
<td>140.1</td>
</tr>
<tr>
<td>Average Max. Test Error</td>
<td>8.29</td>
<td>10.03</td>
</tr>
</tbody>
</table>

Table 6.1: CRCC versus RCC

The regression task with a target associated to the root vertex of each tree (supersource transduction). The target is the boiling point expressed in Celsius degrees (°C) into the range [−164, 174] (scaled by a factor of 100 in the simulations). The data set (D) is composed by 150 alkanes, represented as tree, with up to 10 carbon atoms (vertexes of the tree) with a total of 1331 vertexes.

For the sake of comparison, we tested the prediction ability of the contextual versus the causal model using the same data set and learning parameters used for testing the causal model in [21]. Learning was stopped when the maximum absolute error for a single compound was below 8 °C, or when a maximum number of hidden unit was reached (160 units for this set of experiments)\(^5\). All parameters have been chosen after an initial set of preliminary trials performed in order to determine an admissible range for the RCC models.

A 10-fold cross-validation on the dataset has been performed. For each fold 5 different learning trials were performed. In Table 6.1 the mean and the variance, over each fold and globally, of the mean absolute error obtained for the test set are reported. Specifically, the errors are expressed in °C. Moreover, the average number of hidden units and the average maximum absolute test error, computed over all the folds, are reported as well.

\(^5\) Actually, for few trials the maximum number of hidden units is reached before the maximum error on the training data set was below 8 °C. However, we found that in such cases, both the mean error and the maximum error on the training data set are comparable to the values obtained with the trials that respect the stopping criterion on maximum train error.
In particular, it is possible to observe that the average of the mean test errors obtained by the contextual cascade correlation is around \(0.3\) less than the one obtained with the basic recursive cascade correlation, which corresponds to a relative improvement which is above 10%. Moreover, the CRCC is also more stable than the RCC since the variance on the obtained test mean error is lower than the RCC model. The improvement of CRCC in generalization is also testified by the relevant reduction of the average absolute maximum error, which turns out to be very close to the stop criterion used for the training, i.e., \(8^\circ\). This does not hold for the RCC.

Moreover, notice that while improving the efficacy on the error results with the new model, the efficiency does not decrease, since the number of inserted hidden units by the two models, at the end of the training phase, is comparable. Of course, one hidden unit in CRCC will have more parameters than a hidden unit in RCC due to the contextual connections.

![Example of Causal vs Contextual Learning Curves](image)

Figure 6.10: Comparison of the learning curves obtained for RCC and CRCC models.

An example of comparison of learning curves between CRCC and RCC is shown in Figure 6.10 (see [125]). In the figure the typical behavior of the two cascade models, reporting the mean training and test absolute errors obtained by the two models during learning as a function of the number of inserted hidden units, is shown. These curves suggest that also the dynamical behavior of the CRCC is more effective than the dynamical behavior of the standard RCC. In fact, it can be noted that the CRCC model can obtain the same fitting or generalization results of the RCC model, with much less hidden units. Specifically, the generalization results obtained by the RCC at the end of the training phase can be obtained by CRCC with a number of hidden units which is less than half.
6.7. EXPERIMENTAL RESULTS

6.7.3 Analysis of Internal Representations

In order to investigate the properties of the contextual encoding developed by the CRCC, we analyzed the internal representation, i.e. the output of the hidden units or state vector \( \mathbf{x} \), obtained after training for the alkane dataset. Principal component analysis (PCA) was used to reduce the dimensionality of the representational space so to have the possibility to visualize the internal representations in a 2-D plot. Of course, in this way only global features of the internal representations are retained, however, we will see that this general features are enough to show how context is encoded by CRCC. Specifically, Figure 6.11 shows the plot of the first two principal components of the internal representations generated from a sample experiment of the alkanes data set (on training set of fold-09). Similar plots, with differences restricted to rotations of the axes and scaling of the principal component values, can be obtained from different experiments with the same fold or using different folds. In the plot each point represents the code developed by CRCC for a (sub)structure of the alkanes data. i.e. the state vector \( \mathbf{x}(v) \) for each vertex \( v \) in the data set. A (sub)structure can play either the role of chemical fragment or the role of compound (represented as a box in the plot). Some (sub)structures can be both compounds themselves and fragments belonging to more complex compounds, e.g. a single carbon group (C in Figure 6.11) can represent both the methane and a group belonging to other compounds. Other (sub)structures only represent chemical fragments, and some of them may occur in the same compound, but in different positions (i.e., contexts), and/or they may occur into different compounds.

For the sake of clarity, in the plot we have highlighted only information relevant to some type of small and frequent substructures occurring in the data set. Specifically, we just consider structures with 1, 2, 3 and 4 vertexes. Different internal representations corresponding to the same structure (but occurring in different contexts) are grouped together by drawing them inside a solid line closed shape (cluster) associated with a picture showing the corresponding carbon tree. If a structure can play also the role of a chemical compound, the name of the compound is reported on top of the carbon tree. In this case, the internal representation associated to the compound, i.e., associated to the tree without context (in the sense that it is not part of a larger tree, as it happens when considering the same carbon tree as a fragment included in a larger chemical compound), is marked by a box. Moreover, a dashed line closed shape groups different structures with the same number of nodes.

Roughly, in Figure 6.11, the simplest compounds, i.e. the ones with a small number of vertexes, are located on the left part of the plot, while more complex structures occur on the right part of the plot. Therefore, most of the compounds are on the right part of the plot and most of the small fragments are on the left part of the plot. Since small fragments can belong to a large variety of compounds, i.e., contexts, our analysis will focus on the left part of the plot.

First of all, let us recall that RCC computes for each fragment, i.e. subtree, a unique code which is independent from the context where the fragment does occur. In fact,
Figure 6.11: PCA of internal representations of CRCC for Alkanes.

According to Equation 6.7 (and Equation 6.12), the generated state vector is not influenced by the location (inside a larger tree) where the considered subtree does occur. Hence, the analogous for RCC of each cluster shown in Figure 6.11, necessarily will contain a single point that represents the state vector generated for the subtree.

Differently from RCC, and according to Propositions 6.5 and 6.6, each fragment in CRCC can be represented in different ways depending on the context, i.e., the position (inside a larger tree) where the subtree occurs. Thus, different points in the same cluster may represent either the same fragment occurring in different compounds, or different occurrences of the same fragment in the same compound.

In addition to that, the internal representations are spatially organized so to encode the complexity of the structure represented by each point, which in this prediction task is known to be correlated to the target information.

Hence, Figure 6.11 shows that CRCC is actually able to develop internal representations that take into account the context in which each subtree does occur, and to organize these representations in a way functional to the solution of the prediction task.
6.8 Conclusions

We presented an extension of the RCC model based on the contextual analysis of structured domains, i.e., the Contextual Recursive Cascade Correlation (CRCC) model. The basic idea was to exploit contextual information already present in frozen hidden units. The proposed model constitutes the first example of a recursive neural model for processing of structured data able to exploit contextual information. In fact, both BRNN and the model defined in [171] can only deal with sequences, as well as BRCC that by definition constitutes a special case of CRCC when applied to sequences.

We were able to formally elucidate how the “shape” of the contextual information evolves with the addition of hidden units, and to show that CRCC can compute contextual transductions which cannot be computed by RCC at all. In addition to that, we demonstrated that some causal supersource transductions, which cannot be computed by RCC, can be computed by CRCC, which on the other hand is able to compute all the transductions that can be computed by RCC.

It is worth to note that theorems and definitions, characterizing the functional dependencies between state variables, are introduced abstracting from any specific neural realization of the CRCC model, allowing the further applications of the approach to other recurrent and recursive method in order to formally devise their properties and compare them from a computational point of view. Note also that, in spite of the relative complexity of the contextual information flow that must be considered to process DPAGs, the expression for the shape of the context results very compact.

Experimental results on controlled sequences have confirmed the theoretical results, and also have shown that CRCC is basically equivalent to RCC when considering a fully causal prediction task. Moreover, using the CRCC model we afforded a real-world task, i.e. the prediction of the boiling point of a set of alkanes, in order to show the improvements that can be obtained using a contextual approach versus a pure causal approach (standard RCC) when no information about the validity of the causality assumption is available. It should be noted that, even if the prediction task in this case was defined only on root vertexes, the CRCC model was able to develop internal representations taking into account the context. Specifically, the PCA analysis of the internal representations shows that each chemical fragment does get a different code in CRCC and the performance results allow us to conclude that these codes are more suited for the application task than the ones developed by RCC, which are independent from the context.

In conclusion, both theoretical and experimental results suggest that the new model can be adopted as an alternative to the basic RCC model whenever it is not possible to guarantee the soundness of the causality assumption for the application domain under analysis.
Chapter 7

A General Framework for Unsupervised Processing of Structured Data

7.1 Introduction

In this chapter, we will focus on the study of the *unsupervised* paradigm in the proposed recursive approach to the processing of structured domains. In particular, we analyze the extensions of an important class of connectionist models based on self-organization to the processing of sequences and tree structures. In fact, self-organization constitutes an important paradigm in machine learning with successful applications e.g. in data- and web-mining. However, so far most approaches have been proposed in order to process data contained in a fixed and finite dimensional vector space (see Section 2.2.2). Various extensions of the standard self-organizing map (SOM) to sequences or tree structures have been proposed in the literature ([29, 167, 168, 169, 44, 85, 92, 103, 71, 69, 153], see the next paragraph for details). These methods enhance the standard SOM by utilizing recursive connections. In this chapter, we define a *general recursive dynamic* which provides recursive processing of structured data by recursive computation of internal representations for the given context. The above mentioned mechanisms of SOMs for structures are special cases of the proposed general dynamic.

The general framework proposed in the following is based on concepts typically used in the unsupervised scenario, such as the concept of *distance* (and similarity measure). However, the relationships with the framework presented in the previous chapters can be easily shown. For example, the dynamic proposed in this chapter can also be adapted to cover the supervised case of recurrent and recursive networks. In the case of unsupervised learning, the general framework offers a uniform notation for training mechanisms such as Hebbian learning. Moreover, the transfer of computational alternatives such as vector quantization or the neural gas algorithm to structure processing networks can easily be achieved.
The formal definition of the recursive dynamic for unsupervised networks allows the transfer of theoretical results from the SOM literature to the structure processing case. One can formulate general cost functions corresponding to vector quantization, neural gas, and a modification of SOM. The cost functions can be compared to Hebbian learning which can be interpreted as an approximation of a stochastic gradient descent. For comparison purposes, we derive the exact gradients for general cost functions.

**Motivations and Topics** Unsupervised learning as an alternative important paradigm for neural networks has been successfully applied to data mining and visualization (see [99]). Since additional structural information is often available in possible applications of self-organizing maps (SOMs), a transfer of standard unsupervised learning methods to sequences and more complex tree structures would be valuable. Several approaches have extended SOM to sequences: SOM constitutes a metric based approach. Hence it can be applied directly to structural data if data are represented in such a way that an appropriate metric for the respective structural data is defined and a notion of how to adapt within the space given by structural data can be found. This has been proposed e.g. in [64, 94, 165]. Various approaches, on the other hand, extend SOM by recurrent dynamics such as leaky integrators or more general recurrent connections which allow the recursive processing of sequences. Examples are the temporal Kohonen map (TKM) [29], the recursive SOM (RecSOM) [167, 168, 169], or the approaches proposed in [44, 85, 92, 103]. The SOM for structured data (SOMSD) [71, 69, 153] constitutes a recursive mechanism capable of processing tree structured data, and thus also sequences, in an unsupervised way. Alternative models for unsupervised time series processing use, for example, hierarchical network architectures. An overview of important models can be found e.g. in [9].

In Section 5.2.3 we introduced some issues and some research demands aiming at motivating the study of a general formulation of the recursive dynamics underpinning the current approaches and the investigation of the methods in such a framework. In this chapter we address such problems as described in the following:

First, we focus on models based on recursive dynamics for structured data and we derive a generic formulation of recursive self-organizing maps. We propose a general framework which transfers the idea of recursive processing of complex data for supervised recurrent and recursive networks to the unsupervised scenario. This general framework covers TKM, RecSOM, SOMSD, and the standard SOM. These methods share the basic recursive dynamic but they differ in the way in which structures are internally represented by their neural map. TKM, RecSOM, SOMSD, and the standard SOM can be obtained by an appropriate choice of internal representations in the general framework. Moreover, the dynamic of supervised recurrent and recursive networks can be integrated into the general framework as well. The approaches reported in [44, 85, 103] can be simulated with slight variations to parts of the framework. Hence we obtain a uniform formulation which allows a straightforward investigation of possible learning algorithms and theoretical properties of several important approaches proposed in the literature for SOMs with recurrence.
A second argument concerns the learning topics. The reported models are usually trained with Hebbian learning. This general formulation allows the formalization of Hebbian learning in a uniform manner and the immediate transfer of alternatives like the neural gas algorithm [120] or vector quantization to the existing approaches. For standard vector-based SOM and alternatives like neural gas, Hebbian learning can be (approximately) interpreted as a stochastic gradient descent method on an appropriate error function [82, 120]. One can uniformly formulate analogous cost functions for the general framework of structural self-organizing maps and investigate the connection to Hebbian learning. It turns out that Hebbian learning can be interpreted as an approximation of a gradient mechanism for which contributions of substructures are discarded. This gives some hints to understand the dynamics of unsupervised network training.

A third benefit of the general formulation lies in the possibility of comparing the above methods with respect to their capacity. Note that the above methods have been applied to monitor time series, language, or image data. A first evaluation criterion of structure processing SOMs, the depth of a quantizer, has been proposed in [168]. However, a clear and uniform measure of the success of unsupervised processing of structured data which is adequate for all reasonable situations is not yet available. Hence the success of the approaches is often measured on supervised classification tasks or, alternatively, experts rate the visualization obtained from the neural maps. The reported results for the above mentioned concrete models proposed in the literature seem very promising, however, a general theoretical framework for a deeper insight is helpful. This allows us to formulate general demands like the representation capacity and test these properties for the specific choices of representation. As an example, we will investigate the noise tolerance and the representation capability. It is possible to prove that specific choices of the internal representation of context, the part which distinguishes the various specific approaches, are more appropriate for structure processing than others. These investigations are the first step toward a general theory of unsupervised recurrent and recursive networks.

**Structure of the Chapter** Following the above objectives, the chapter is organized as follows. In Section 7.2 we recall the previous approaches to unsupervised learning of flat and structured domains and we specialize the notation for the specific purpose of the Chapter. The proposed general framework for the general recursive dynamic (GSOMSD) is introduced in Section 7.3; we also show that SOMs with recursive dynamics as proposed in the literature can be recovered as special cases of the general framework and we show the relationships with the abstract scheme introduced in the previous chapters. In Section 7.4 and Section 7.5 we study learning inside the devised framework. Conclusions are drawn in Section 7.7.

The basic framework for unsupervised learning in SD described in this chapter appears in [75]. This has also been recently accepted for journal publication in [76]. A more extensive presentation is in [74].
7.2 Structure Processing Self-Organizing Maps

We first clarify a notation: the term ‘self-organizing map’ used in the literature refers to both, the paradigm of a neural system which learns in a self-organizing fashion, and the specific and very successful self-organizing map which has been proposed by Kohonen [99]. In order to distinguish between these two meanings, we refer to the specific architecture proposed by Kohonen by the shorthand notation SOM. When we speak of self-organization, the general paradigm is referred to.

The notions and details on unsupervised learning in flat domains by the SOM neural network model have been introduced in Section 2.2 following the (literature) classical notation. In order to introduce the apt notation used in this chapter, we briefly recall in the following the basic SOM idea. In addition, before moving toward structures processing, we introduce the background notions for standard vector quantization and Neural Gas approaches.

The SOM as proposed by Kohonen is a biologically motivated neural network which learns, via Hebbian learning, a topological representation of a data distribution from examples. Assume data are taken from the real-vector space $\mathbb{R}^n$ equipped with the Euclidean metric $\| \cdot \|$. The SOM is defined as a set of neurons $\mathcal{N} = \{n_1, \ldots, n_N\}$ together with a neighborhood structure of the neurons $nh: \mathcal{N} \times \mathcal{N} \rightarrow \mathbb{R}$. This is often determined by a regular lattice structure, i.e. neurons $n_i$ and $n_j$ are direct neighbors with $nh(n_i, n_j) = 1$ if they are directly connected in the lattice; for other neurons, $nh(n_i, n_j)$ reflects the minimum number of direct connections needed to link $n_i$ to $n_j$ (different neighbor functions have been introduced, for example, in Section 2.2.2). A two-dimensional lattice offers the possibility of easy visualization which is used e.g. for applications in data mining [100]. Each neuron $n_i$ is equipped with a weight $w_i \in \mathbb{R}^n$ which represents the corresponding region of the data space. Given a set of training patterns $\mathcal{D} = \{a_1, \ldots, a_d\}$ in $\mathbb{R}^n$, the weights of the neurons are adapted by Hebbian learning including neighborhood cooperation such that the weights $w_i$ represent the training points $\mathcal{D}$ as accurately as possible and the topology of the neurons in the lattice matches the topology induced by the data points. The learning rule (introduced in Section 2.2.2) can be summarized as follows:

repeat: choose $a_i \in \mathcal{D}$ at random
compute the neuron $n_{i*}$ with minimum distance $\|a_i - w_{i*}\|^2 = \min_j \|a_i - w_j\|^2$
adapt for all $j$: $w_j := w_j + \eta(nh(n_j, n_{i*}))(a_i - w_j)$

where $\eta(nh(n_j, n_{i*}))$ is a learning rate function\(^1\) which is maximum for the winner $n_j = n_{i*}$ and decreasing for neurons $n_j$ which are not direct neighbors of the winner $n_{i*}$. Often, it is of the form $\exp(-nh(n_j, n_{i*}))$ possibly with additional constant factors involved in the term. The incorporation of topology in the learning rule allows to adapt the winner

---

\(^1\)With this short notation the learning rate and the shape of the neighborhood function are enclosed in the $\eta$ function.
and all neighbors at each training step. After training, the SOM is used with the following dynamic: given a pattern \( a \in \mathbb{R}^n \), the map computes the winner, i.e. the neuron with smallest distance \( \| a - w_j \|^2 \) or its weight, respectively. This allows us to identify a new data point with an already learned prototype. Starting from the winning neuron, browsing through the map in order to find similar known data is possible.

Popular alternative self-organizing algorithms are vector quantization (VQ) and the neural gas algorithm (NG) [121]. VQ aims at learning a representation of the data points without topology preservation. Hence no neighborhood structure is given in this case and the learning rule adapts only the winner at each step:

\[
\text{repeat: choose } a_i \in \mathcal{D} \text{ at random}
\]

\[
\text{compute the neuron } n_{i^*} \text{ with minimum distance } \| a_i - w_{i^*} \|^2 = \min_j \| a_i - w_j \|^2
\]

\[
\text{adapt } w_{i^*} := w_{i^*} + \eta (a_i - w_{i^*})
\]

where \( \eta > 0 \) is a fixed learning rate. NG does not priorly fix a topology of the neurons since a wrong choice may yield topological defects, and hence an inappropriate representation of data. Rather, a neighborhood is defined posteriorly through training data. The recursive update reads as

\[
\text{repeat: choose } a_i \in \mathcal{D} \text{ at random}
\]

\[
\text{compute all distances } \| a_i - w_j \|^2
\]

\[
\text{adapt for all } j: w_j := w_j + \eta (rk(i, j))(a_i - w_j)
\]

where \( rk(i, j) \) denotes the rank of neuron \( n_j \) according to the distances \( \| a_i - w_j \|^2 \), i.e. the number of neurons \( n_k \) for which \( \| a_i - w_k \|^2 < \| a_i - w_j \|^2 \) holds. \( \eta (rk(i, j)) \) is a function with maximum at 0 and decreasing values for larger numbers, e.g. \( \eta (rk(i, j)) = \exp(-rk(i, j)) \) possibly with additional constant factors. This results in the closest neuron to be adapted most, all other neurons are adapted according to their distance from the given data point. Hence, the respective order of the neurons with respect to a given training point determines the current neighborhood. Eventually, those neurons which are the neurons closest to at least one data point become neighbored. One can infer a data-adapted no longer regular lattice after training in this way [120, 121].

Various possibilities exist of extending self-organizing maps so that they can deal with more complex structures instead of simple vectors in \( \mathbb{R}^n \). The article [9] provides an overview of self-organizing networks which have been proposed for processing spatio-temporal patterns. Naturally, a very common way of processing structured data with self-organizing mechanisms relies on adequate preprocessing of sequences. They are represented through features in a finite dimensional vector space for which standard pattern recognition methods can be used (see Section 2.3). A simple way of sequence representation is obtained by a truncated and fixed dimensional time-window of data. Since this method often yields too large dimensions, SOM with the standard Euclidean metric suffers from the curse of dimensionality, and methods which adapt the metric as proposed
for example in [77, 96, 149] are advisable. Hierarchical and adaptive preprocessing methods which involve SOMs at various levels can be found e.g. in the WEBSOM approach for document retrieval [100]. Since self-organizing algorithms can immediately be transformed to arbitrary metrical structures instead of the standard Euclidean metric, one can alternatively define a complex metric adapted to structured data instead of adopting a complex data preprocessing. SOMs equipped with the edit distance constitute one example [64]. Data structures might be contained in a discrete space instead of a real vector space in these approaches. In this case one has to additionally specify how the weights of neurons are adapted. If the edit distance is dealt with, one can perform a limited number of operations which transform the actual weight of the neuron to the given data structure, for example. Thereby, some unification has to be done since the operations and their order need not be unique. Some methods propose the representation of complex data structures in SOM with complex patterns, e.g. the activation of more than one neuron. Activation patterns arise through recursive processing with appropriate decaying and blocking of activations like in SARDNET [92]. Alternative methods within these lines can be found in [9].

Here we are interested in approaches which equip the SOM with additional recursive connections which make recursive processing of a given recursive data structure possible. In particular, no prior metric or preprocessing is chosen but the similarity of structures evolves through the recursive comparison of the single parts of the data. Various approaches have been proposed in the literature. Most of them deal with sequences of real vectors as input data. One approach has been proposed for more general tree structures and thus including sequences as a special case. We shortly summarize important recursive dynamics in the following.

**Notations for SD**

Special notations are used in this chapter for structured data and functions over the domain. The introduced formalism is thought to be appropriate for an unsupervised paradigm, which is the objective of the current chapter. However, the majority of the notations introduced in Chapter 3 are used without modifications. Moreover, most of the novelties constitute slight variations in the previous notations, and are useful to shorten and simplify the formulas, and to match, as possible, the background in the unsupervised area: they will be summarized in the following. Although it is not necessary in order to read the chapter, the relationships with the previous notations will be clarified whenever it is necessary.

A very compact prefix notation for a $K$-ary tree, useful if the single parts of a tree (root label and subtrees) are referred to, is the following: $T(r) = a(t_1, t_2, \ldots, t_K)$ where $a$ is a shorthand for the root label of the current (sub)tree $T(r)$, i.e. $a \equiv l(t)$ and $a \in L$, and $t_j \equiv q_j^{-1}T(r) = T(ch_j(r))$. The empty tree is still denoted by $\zeta$.

Sequences over $L$ (typically over $\mathbb{R}^n$) with entries $a_1 \in L$ to $a_t \in L$ are denoted by $[a_1, \ldots, a_t]$. $[\ ]$ denotes the empty sequence.
Temporal Kohonen Map

The temporal Kohonen map proposed by Chapels and Taylor [29] extends the SOM using recurrent self-connections of the neurons in such a way that the neurons act as leaky integrators. Given a sequence \([a_1, \ldots, a_t]\), the activation of neuron \(n_i\) with weight \(w_i\) is computed as

\[
d_i(t) = \sum_{j=1}^{t} \alpha \cdot (1 - \alpha)^{(t-j)} \cdot ||a_j - w_i||^2
\]

where \(\alpha \in (0, 1)\) is a constant which determines the integration of context information when the winner is computed. This formula has the form of a leaky integrator which integrates previous activations of neuron \(n_i\) given the entries of the sequence. Hence a neuron becomes the winner if its weight is close to the given data point and, in addition, if the exponentially weighted distance of previous entries of the sequence from the weight is small. Obviously, an alternative recursive formalization of the integrated distance of neuron \(n_i\) after the \(t\)th time step is given by

\[
d_i(t) = \alpha \|a_j - w_i\|^2 + (1 - \alpha)d_i(j - 1)
\]

where \(d_i(0) := 0\). Training of the TKM is performed in [29] with Hebbian learning after each time step, i.e. the weights \(w_i\) are adapted at each time step with respect to the current input \(a_j\) according to the standard SOM update rule. Thereby, the winner is computed as the neuron with the least activation according to the above leaky integration formula.

The recurrent SOM (RSOM) as defined in [103], for example, uses a similar dynamic. However, it integrates the directions of deviations of the weights. Hence the activation \(d_i(j)\) which is now an element of \(\mathbb{R}^n\) is recursively computed by \(d_i(0) = 0\), \(d_i(j) = \alpha (a_j - w_i) + (1 - \alpha)d_i(j - 1)\). The winner is determined as the neuron with the smallest integrated distance, i.e. the smallest \(\|d_i(j)\|^2\). Naturally, this procedure stores more information than the weighted distance of the TKM. Again, Hebbian learning can be used at each time step, like in TKM. In [103] an alternative direct training method is proposed for both, SOM and TKM, which evaluates the condition that at an optimum in the weight space the derivative of the quantization error is zero. This quadratic equation can be solved analytically which yields solutions for optimum weights \(w_i\). Apart from sequence recognition tasks, these models have been successfully applied for learning motion-directivity sensitive maps as can be found in the visual cortex [48].

Recursive SOM

The recursive SOM (RecSOM) has been proposed by Voegtlin [167, 169] as a mechanism for sequence prediction by recursive processing the symbols based on the already computed context\(^2\). Each neuron is equipped with a weight \(w_i \in \mathbb{R}^n\) and, additionally, with a context vector \(c_i \in \mathbb{R}^N\) which stores an activation profile of the whole map, indicating in which sequential context the vector \(w_i\) should arise. Given a sequence \([a_1, \ldots, a_t]\), the

\(^2\)The context considered in the following is to be understood in the restrictive sense as the recursive code/state computed by the causal recurrent/recursive dynamic to store "past" information, i.e. as used in literature for the Elman style models.
activation of neuron \( n_i \) at time step \( j \) is defined as \( d_i(0) = 0 \) and

\[
d_i(j) = \alpha \cdot \|a_j - w_i\|^2 + \beta \cdot \|\exp(-d_i(j-1), \ldots, \exp(-d_N(j-1))) - c_i\|^2
\]

where \( \alpha, \beta > 0 \) are constants to control mediation between the amount of pattern match versus context match. Hence the respective symbol is compared with the weights \( w_i \).

In addition, the already computed context, i.e. the activation of the whole map in the previous time step, has to match with the context \( c_i \) of neuron \( n_i \) such that the neuron becomes the winner. The comparison of contexts is done by involving the exponential function \( \exp \) in order to avoid numerical explosion. If this exponential transform was not included, the activation \( d_i(j) \) could become huge because the distances with respect to all of the \( N \) components of the context could accumulate. The RecSOM has been applied to sequence recognition and prediction of the Mackey-Glass time series and to the recognition of sequences induced by a randomized automaton. Training has been done with Hebbian learning in these cases for both, the weights \( w_i \) of the neurons and their contexts \( c_i \). Thereby, the parameters \((w_i, c_i)\) of the recursively computed winner neuron \( n_i \) are adapted toward the current input \( a_j \) and the recursively computed context; the neighbors of \( n_i \) are adapted accordingly with a smaller learning rate. The RecSOM shows a good capability of differentiating input sequences. The winners represent different sequences which have been used for training \([167, 169]\).

**SOM for Structured Data**

The SOM for structured data (SOMSD) has been proposed for the processing of labeled trees with fixed out-degree \( K \) \([69, 153]\). The specific case \( K = 1 \) covers sequences. It is assumed that the neurons are arranged on a rectangular \( d \)-dimensional lattice structure. Hence neurons can be enumerated by tuples \( \mathbf{i} = (i_1, \ldots, i_d) \) in \( \{1, \ldots, N_1\} \times \ldots \times \{1, \ldots, N_d\} \) where \( N_1 \cdot \ldots \cdot N_d = N \). Each neuron \( n_{\mathbf{i}} \) is equipped with a weight \( w_{\mathbf{i}} \) and \( K \) context vectors \( c^1_{\mathbf{i}}, \ldots, c^K_{\mathbf{i}} \) in \( \mathbb{R}^d \). They represent the context of a processed tree given by the \( K \) subtrees and the indexes of their respective winners. The index \( I(t) \) of the winning neuron given a tree \( t \) with root label \( a \) and subtrees \( t_1, \ldots, t_K \) is recursively defined by

\[
I(\zeta) = \text{nil} \\
I(a(t_1, \ldots, t_K)) = \arg\min_{\mathbf{i}} \{\alpha \|a - w_{\mathbf{i}}\|^2 + \beta (\|I(t_1) - c^1_{\mathbf{i}}\|^2 + \ldots + \|I(t_K) - c^K_{\mathbf{i}}\|^2)\}
\]

Hence the respective context of a label in the tree is given by the winners for the \( K \) subtrees. Hereby, the empty tree is represented by an index not contained in the lattice, e.g. \( \text{nil} = (-1, \ldots, -1) \). The weights of the neurons consist of compact representations of trees with prototypical labels and prototypical winner indexes which represent the subtrees. Starting from the leaves, the winner is recursively computed for an entire tree. Even for SOMSD, Hebbian learning is applied. Starting from the leaves, the index \( \mathbf{i} \) of the winner for the subtrees is computed. The attached weights \((w_{\mathbf{i}}, c^1_{\mathbf{i}}, \ldots, c^K_{\mathbf{i}})\) are moved
into the direction \((a, I(t_1), \ldots, I(t_K))\) after each recursive processing step, where \(I(t_i)\) denotes the winning index of the subtree \(t_i\) of the currently processed part of the tree. The neighborhood is updated into the same direction with a smaller learning rate.

This method has been used for the classification of artificially constructed pictures represented by tree structured data [71, 69]. During learning, a representation of the pictures in the SOMSD emerges which groups together pictures described by similarly structured trees with similar labels. In the reported experiments, the pictures generated by a plex grammar correspond to ships, houses, and policemen with various shapes and colors. SOMSD is capable of properly arranging objects of these categories, e.g. houses. Within the clusters a differentiation with respect to the involved features can be observed like in the standard SOM. Based on this clustering a good classification accuracy of the objects can be obtained by attaching appropriate classes to the nodes [71].

### 7.3 A General Framework for the Dynamic

The main idea for a general framework for these models is derived from the observation that all models share the basic recursive dynamics. They differ in the way that tree structures or sequences, respectively, are internally represented with respect to the recursive processing and the weights of neurons. The above models deal with either sequences over a real vector space or labeled trees with out-degree \(K\). For simplicity, we will here focus on binary trees with labels in some set \(\mathcal{L}\). The generalization to trees with out-degree \(K\), in particular sequences which are trees with out-degree 1, is obvious. Labels of trees may originate from an arbitrary set \(\mathcal{L}\): we allow an arbitrary set \(\mathcal{L}\) instead of a real vector space to include the case of discrete labels such as symbolic data. Let us denote by \(B\) the set of binary trees with labels in \(\mathcal{L}\). We use a prefix notation if the single parts of a tree are referred to; \(a(t_1, t_2)\) represents the tree with root label \(a\) and subtrees \(t_1\) and \(t_2\). As above, the empty tree is denoted by \(\zeta\). How can this type of data be processed in an unsupervised fashion? We here define the basic ingredients and the basic dynamic or functionality of the map: given a trained map and a new input tree, how is the winner for this tree computed? Based on this general dynamic, which as a special case includes the models described above, i.e., TKM, RecSOM, and SOMSD, a canonic formulation of Hebbian learning will be derived in the next section.

The processing dynamic is based on the following main ingredients which define the general SOM for structured data or GSOMSD, for short:

1. The set of labels \(\mathcal{L}\) together with a similarity measure \(d_\mathcal{L} : \mathcal{L} \times \mathcal{L} \to \mathbb{R}\).

2. A set of formal representations \(\mathcal{R}\) for trees together with a similarity measure \(d_\mathcal{R} : \mathcal{R} \times \mathcal{R} \to \mathbb{R}\). Denote the priorly fixed formal representation of the empty tree \(\zeta\) by \(r_\zeta\). The adequate formal representation of any other trees will be computed via the GSOMSD.

3. A set of neurons \(\mathcal{N}\) of the self-organizing map which we assume to be enumerated with \(1, \ldots, N = |\mathcal{N}|\), for simplicity. A weight function \(\mathcal{W} = (\mathcal{W}_0, \mathcal{W}_1, \mathcal{W}_2) : \)}
A general framework for unsupervised processing of SD attaches a “weight” to each neuron. For each neuron this consists in the map of a triple consisting of a prototype label and two formal representations.

4. A representation mapping \( rep : \mathbb{R}^N \rightarrow \mathcal{R} \) which maps a vector of activations of all neurons into a formal representation.

The idea behind these ingredients is as follows: a tree can be processed iteratively; starting from the leaves, it can be compared with the information contained in GSOMSD until the root is reached. At each step of the comparison the context information resulting from the processing of the two subtrees should be taken into account. Hence a neuron weighted with \( a \) in the map is a proper representation of the entire tree if the first part of the weight attached to the neuron, \( w \), represents a properly, and \( c_1 \) and \( c_2 \) represent correct contexts, i.e. they correspond to \( t_1 \) and \( t_2 \). \( w \) and \( a \) can be compared using the similarity measure \( d_\mathcal{L} \) directly. \( c_1 \) and \( c_2 \) are formal descriptions of the contexts, which could be some pointers or some reduced description for the context and which should be compared with \( t_1 \) and \( t_2 \), respectively. For this purpose, \( t_1 \) and \( t_2 \) can be iteratively processed yielding a context, i.e. an activation of all neurons in the map. This activation can be transferred to a formal description of the context via the representation mapping \( rep \). The comparison of the output with \( c_i \) is then possible using the metric \( d_\mathcal{R} \) on the formal representations.

This informal description can be formalized by the following definition which allows to compute the recursive similarity or activation \( \tilde{a} \) of a neural unit \( n_i \) given a tree \( a(t_1, t_2) \):

**Definition 7.1 (GSOMSD)** The generalized SOM for structured data or GSOMSD refers to the dynamic:

\[
\tilde{a}(a(t_1, t_2), n_i) = \alpha \cdot d_\mathcal{L}(a, \mathcal{W}_0(n_i)) + \beta \cdot d_\mathcal{R}(R_1, \mathcal{W}_1(n_i)) + \beta \cdot d_\mathcal{R}(R_2, \mathcal{W}_2(n_i))
\]

where \( a(t_1, t_2) \in \mathcal{B} \), \( n_i \in \mathcal{N} \), and

\[
R_j = \begin{cases} 
\rho \zeta & \text{if } t_j = \zeta \\
\text{rep}(\tilde{a}(t_j, n_1), \ldots, \tilde{a}(t_j, n_N)) & \text{otherwise}
\end{cases}
\]

for \( j = 1, 2 \), and \( \alpha, \beta > 0 \) are weighting factors.

In the Definition 7.1, the choice of \( \alpha \) and \( \beta \) determines the importance of a proper context in comparison to the correct root label of the representation. The set \( \mathcal{R} \) enables a precise notation of how trees are internally stored in the map. The function \( \text{rep} \) constitutes the interface which maps activity profiles to internal representations of trees. The recursive similarity yields the activation of all neurons for an input tree. Applying \( \text{rep} \), we can obtain a formal representation of the tree. In order to use the resulting map, for example for information storing and recovering, we can determine a neuron with highest responsibility, i.e. a neuron which fires, or the winner, given the input tree. This could be the neuron/neurons with the highest or lowest recursive similarity, depending on the meaning of \( \tilde{a} \), e.g. depending on the fact as to whether dot products or distances are used for the
7.3. A GENERAL FRAMEWORK FOR THE DYNAMIC

Figure 7.1: One recursive processing step: given a tree \( a(t_1, t_2) \), the distance from a neuron \( n_i \) weighted with \( (\mathcal{W}_0(n_i), \mathcal{W}_1(n_i), \mathcal{W}_2(n_i)) \) is computed weighting the distances of \( a \) from \( \mathcal{W}_0(n_i) \), and the distances of \( \mathcal{W}_1(n_i) \) and \( \mathcal{W}_2(n_i) \), respectively, from the representations of \( t_1 \) and \( t_2 \). These latter representations can be obtained recursively processing the trees and applying \( rep \) to the obtained activity profile of the map.

computation of the similarities. A picture which explains the processing in one recursive step can be found in Figure 7.1.

Note the following:

1. The labels of the input trees might come from a proper subset of \( \mathcal{L} \), e.g. if discrete data are processed, and the representations of the neurons lie in a real vector space. \( d_\mathcal{L} \) is often the squared standard Euclidean metric or induced by any other metric; however, at this point, we do not need special properties of \( d_\mathcal{L} \).

2. The formal representations \( \mathcal{R} \) should represent trees in a compact way, e.g. in a finite dimensional vector space. As an example, they could be chosen simply as the index of the neuron with best recursive similarity if a tree is processed with a trained self-organizing map. The idea behind this is that a connectionist distributed representation of a tree emerges from the processing of the tree by the map. The formal description could be a shorthand notation, a pointer for this activity profile. Based on this assumption we can compare trees by comparing their formal descriptions.
3. The neurons can be considered as prototypes for trees, i.e. their root label and their subtrees. The latter are represented by their formal representations. Note that we have not introduced any lattice or topological structure of the neurons at this point. The definition of a topology of the self-organizing map does not affect the recursive similarity of neurons given a tree. However, a topological structure might be used for training or it might prove beneficial for specific applications such as browsing in topological representations of the given data space on the map.

4. The mapping \( rep \) maps the recursively processed trees which yields an activity profile of the neurons into a formal description of trees. It might simply be the identity or an appropriate compactification, for example. However, it is no longer a symbolic tree representation but a connectionist description based on the activation of the GSOMSD.

Obviously, the GSOMSD can be easily defined for trees with out-degree \( K \) where \( k \neq 2 \). In particular, the case of sequences, i.e. \( K = 1 \) is then included. When considering trees with out-degree \( K \), the weight function is of the form:

\[
\mathcal{W} = (\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_K) : \mathcal{N} \to \mathcal{L} \times \mathcal{R}^K
\]

i.e. \( K \) contexts are attached to the neurons corresponding to the out-degree \( K \) instead of 2. The recursive processing reads as

\[
\bar{d}(a(t_1, \ldots, t_K), n_i) = \alpha \cdot d_{\mathcal{L}}(a, \mathcal{W}_0(n_i)) + \beta \cdot d_{\mathcal{R}}(R_1, \mathcal{W}_1(n_i)) + \ldots + \beta \cdot d_{\mathcal{R}}(R_2, \mathcal{W}_K(n_i))
\]

where

\[
R_j = \begin{cases} 
  r_{\zeta} & \text{if } t_j = \zeta \\
  rep(\bar{d}(t_j, n_1), \ldots, \bar{d}(t_j, n_N)) & \text{otherwise}
\end{cases}
\]

This abstract definition captures the above approaches of structure processing self-organizing networks. We prove this fact for the case of binary input trees. The transfer to sequences is obvious.

**Theorem 7.2** GSOMSD includes the dynamic of SOM, TKM, RecSOM, and SOMSD via appropriate choices of \( \mathcal{R} \).

**Proof.** **SOM:** For SOM, we choose \( \mathcal{R} = \emptyset \) and \( d_{\mathcal{R}} \equiv 0 \). Then no context is available and hence only the labels of the root of the trees, or elements in \( \mathcal{L} \), respectively, are taken into account.

**TKM:** For the TKM, the ingredients are as follows:

1. Define \( \mathcal{L} = IR^n \). \( d_{\mathcal{L}} \) is the squared Euclidean metric.

2. \( \mathcal{R} = IR^N \) explicitly stores the activation of all neurons if a tree is processed, \( N \) denotes the number of neurons. The similarity measure \( d_{\mathcal{R}} \) is here given by the dot product. The representation for the empty tree \( \zeta \) is the vector \( r_{\zeta} = (0, \ldots, 0) \).
3. The weights of the neurons have a special form: in neuron \( n_i \) with \( \mathcal{W}(n_i) = (w, c_1, c_2) \) the first parameter may be an arbitrary value in \( \mathbb{R}^n \) obtained by training. The contexts \( c_1 = c_2 \) coincide with the unit vector which is one at position \( i \) and 0 otherwise. Hence the context represented in the neurons is of a particularly simple structure. One can think of the context as a focus: the neuron only looks at its own activation when processing a tree; it doesn’t care about the global activation produced by the tree. Mathematically, this is implemented by using the dot product \( d_{\mathcal{R}} \) of the context with the unit vector stored by the neuron.

4. \( \text{rep} \) is simply the identity, no further reduction takes place.

Then the recursive dynamic reduces to the computation

\[
\tilde{d}(a(t_1, t_2), n_i) = \alpha \cdot \| a - \mathcal{W}_0(n_i) \|^2 + \beta \cdot \tilde{d}(t_1, n_i) + \beta \cdot \tilde{d}(t_2, n_i) \quad \text{for} \ t_1, t_2 \neq \zeta.
\]

If we choose \( \beta = 1 - \alpha \) and consider the case of sequences, this is just a leaky integrator as defined above. An analogous dynamic results for tree structures which does not involve global context processing but focuses on the activation of a single neuron. The winner can be computed as the neuron with the smallest value \( \tilde{d} \).

**RecSOM:** For RecSOM, we define:

1. Define \( \mathcal{L} = \mathbb{R}^n \). \( d_\mathcal{L} \) is the squared Euclidean metric.

2. Define \( \mathcal{R} = \mathbb{R}^N \) where \( N \) is the number of neurons in the self-organizing map. \( d_\mathcal{R} \) is the squared Euclidean metric. Here the formal description of a tree is merely identical to the activation of the neurons in the self-organizing map. No reduction with respect to the dimensionality takes place. The representation for the empty tree \( \zeta \) is the origin \((0, \ldots, 0)\).

3. The neurons are organized on a lattice in this approach; however, this arrangement does not affect the processing dynamics.

4. Choose \( \text{rep} = \text{rep}_\mathcal{R} \) where \( \text{rep}_\mathcal{R}(x_1, \ldots, x_N) = (\exp(-x_1), \ldots, \exp(-x_N)) \). The exponential function is introduced for stability reasons. On the one hand, it prevents the similarities from blowing up during recursive processing; on the other hand, it scales the similarities in a nonlinear fashion to make small similarities getting close to zero. Noise which could disturb the computation due to the large dimensionality of \( \mathcal{R} \) is suppressed in this way. However, no information is gained or lost by the application of the exponential function.

These definitions lead to the recursive formula

\[
\tilde{d}(a(t_1, t_2), n_i) = \alpha \cdot \| a - \mathcal{W}_0(n_i) \|^2 + \beta \cdot \| R_1 - \mathcal{W}_1(n_i) \|^2 + \beta \cdot \| R_2 - \mathcal{W}_2(n_i) \|^2 \quad \text{where} \ R_j = (\exp(-\tilde{d}(t_j, n_1)), \ldots, \exp(-\tilde{d}(t_j, n_N)))
\]

for \( j = 1, 2 \) and \( t_1, t_2 \neq \zeta \). If we restrict the out-degree to 1, we get the dynamic of RecSOM as introduced above. The winner is again computed as the neuron with the smallest value \( \tilde{d} \).

**SOMSD:** We choose for SOMSD:

1. Define \( \mathcal{L} = \mathbb{R}^n \). \( d_\mathcal{L} \) is the squared Euclidean metric.
2. $\mathcal{R}$ is the real vector space which contains the set of indexes of neurons in the self-organizing map. If the neurons lie on a $d$-dimensional lattice, $\mathcal{R} = \mathbb{R}^d$. $d_\mathcal{R}$ is the squared Euclidean distance of the lattice points. In addition, the representation of the empty tree $\zeta$ is a singular point $r_\zeta$ outside the lattice, e.g. $(-1, \ldots, -1)$.

3. The neurons have a topological structure: they lie on a $d$-dimensional lattice. The weighting attaches appropriate values in $\mathcal{L} \times \mathcal{R} \times \mathcal{R}$ to all neurons. The weights are trained by Hebbian learning. Note that the above assumed enumeration $1, \ldots, N$ of neurons can be substituted by an enumeration based on this lattice with elements $\vec{r} \in \{1, \ldots, N_1\} \times \cdots \times \{1, \ldots, N_d\}$.

4. We choose $\text{rep} = \text{rep}_S$ as follows: the input domain of $\text{rep}_S$, the set $\mathbb{R}^N$ where $N$ denotes the number of neurons, can be identified by $\mathbb{R}^{N_1 \times \cdots \times N_d}$. $\text{rep}_S$ maps a vector of similarities to the index of the neuron with best similarity: $\text{rep}_S(x_{(1,\ldots,1)}, \ldots, x_{(N_1,\ldots,N_d)}) = \vec{r}$ such that $x_{\vec{r}}$ is the smallest input. Note that this need not be unique. We assume here that an ordering of the neurons is given. In the case of multiple optima, we assume implicitly that the first optimum is chosen.

With this definition we obtain the general formula for the recursive similarity
$$d(a(t_1, t_2), n_i) = \alpha \cdot ||a - \mathcal{W}_0(n_i)||^2 + \beta \cdot ||R_1 - \mathcal{W}_1(n_i)||^2 + \beta \cdot ||R_2 - \mathcal{W}_2(n_i)||^2$$
for nonempty trees $t_1$ and $t_2$, where $R_1$ and $R_2$ are the indexes of the neurons which are the winners for $t_1$ and $t_2$, respectively. The winner of the map can again be computed as the neuron with smallest $d$.

Note that in all of the above examples $\tilde{d}$ corresponds to a distance which has been computed within the GSOMSD. This is the standard setting which can also be found in the standard SOM: given an input, the neurons compute an activation which measures their distance from the respective input. Afterward, the winner can be determined as the neuron with smallest distance. Note that this interpretation as distance is common, but more generally we could interpret the similarity $\tilde{d}$ as the activation of the neurons for an input signal. Then, general supervised recurrent and recursive networks can be shown included in the above framework, too, as can be seen in the following subsection.

**The Framework as SD-Recursive Processing System**

We show in the following how the above framework, although designed for the unsupervised paradigm, can be exploited to describe supervised RNN, that have been presented in Chapter 4 through the realization of the functions that compose $\mathcal{T}_G$. In other words the dynamic described by SD-Recursive processing systems can be included in this framework. On the other hand, we can complete the study of the correspondence between the two formalisms showing how the formulation of the above framework (GSOMSD) can be considered an instance of the transduction functions, i.e. the GSOMSD is a SD-Recursive processing system, and showing the shape that the structure encoding function can assume for specific self-organizing models. Of course, while the definition of SD-Recursive processing system is quite general, the GSOMSD framework is specifically useful in the
unsupervised paradigm since it allows us to make the concept of similarity measure in its formulation explicit.

Let us start by showing that supervised recurrent and recursive networks can be included in the above framework. We focus on simple Elman-dynamic, i.e., the recurrent and recursive models presented in Chapter 4. As presented in Chapters 4 and 5, these architectures are illustrative and, although various different dynamics for discrete time recurrent neural networks (RNN) have been established in the literature [161] [110], most of the models can at least be simulated or approximated within the following simple Elman-dynamic [73].

Assume that sequences with entries in \( \mathbb{R}^n \) are dealt with. The \( N \) neurons \( n_1, \ldots, n_N \) are equipped with weights \( w_i = (w_i^0, \hat{v}_i^1) \in \mathbb{R}^{n+N} \). Denote by \( sgd \) the sigmoidal activation function. Then the activation of the neurons \( \tilde{D} \in \mathbb{R}^N \) given a sequence \( [a_1, \ldots, a_t] \) can be computed by

\[
\tilde{D}(a) = (\tilde{D}([a_1, \ldots, a_{t-1}]), \ldots, \tilde{D}([a_1, \ldots, a_{t-1}]))
\]

where ‘-’ denotes the dot product of vectors. Often, an output function (e.g., a linear function) is added in order to obtain the desired output from \( \tilde{D} \).

The same arguments can be generalized to study recursive neural networks (RNN) dealing with structures. Here, we introduce the dynamic for binary trees with labels in \( \mathbb{R}^n \). Neurons are weighted with \( w = (w_i^0, \hat{v}_i^1, \hat{v}_i^2) \in \mathbb{R}^{n+N+N} \). The activation \( \tilde{D} \in \mathbb{R}^N \) can recursively be computed by

\[
\tilde{D}(\zeta) = (0, \ldots, 0),
\]

\[
\tilde{D}(a(t_1, t_2)) = (\tilde{D}([a + \hat{v}_1^1 \cdot \tilde{D}(t_1) + \hat{v}_1^2 \cdot \tilde{D}(t_2)]), \ldots, \tilde{D}([a + \hat{v}_1^1 \cdot \tilde{D}(t_1) + \hat{v}_1^2 \cdot \tilde{D}(t_2)]))
\]

where again an output function is added to obtain the final output.

**Theorem 7.3** RNNs can be formulated within the dynamic of GSOMSD.

**Proof.** Choose \( \alpha = \beta = 1 \) and choose in the recursive computation the following ingredients:

1. \( \mathcal{L} = \mathbb{R}^n \), \( d_\mathcal{L} \) is the dot product.
2. \( \mathcal{R} = \mathbb{R}^N \) stores the activation of all (hidden) neurons, \( d_\mathcal{R} \) is the dot product, \( r_\zeta = (0, \ldots, 0) \).
3. the \( N \) neurons are equipped with the weights \( \mathcal{W}_0(n_i) = w_i^0 \), \( \mathcal{W}_1(n_i) = \hat{v}_i^1 \), and \( \mathcal{W}_2(n_i) = \hat{v}_i^2 \).
4. \( rep(x_1, \ldots, x_N) = (sgd(x_1), \ldots, sgd(x_N)) \).
In this way we obtain the so-called input net of the neurons as recursive similarity \( \tilde{d} \), i.e. \( \text{sgd}(\tilde{d}) = \tilde{D} \) where \( \text{sgd} \) denotes component-wise application of the hyperbolic tangent. Hence, if we substitute the computation of the final winner by the function \( \text{sgd} \) with a possibly added further output function, we obtain the processing dynamic of recurrent and recursive networks within the GSOMSD dynamic.

On the other hand, we can at opposite look at the current framework inside the definitions of Chapter 3. The GSOMSD dynamic describes an adaptive SD-Recursive processing system \( \langle \mathcal{G}, \mathcal{T}_G, \mathcal{A}_W \rangle \); in fact, to see GSOMSD as a graph transduction \( \mathcal{T}_G : \mathcal{G} \rightarrow \mathcal{O} \) we consider the following equations:

\[
\mathcal{T}_G = g \circ \tau_E \quad (7.1)
\]

\[
\tau_E(T(r)) = \text{rep}(\tilde{d}(T(r), n_1), \ldots, \tilde{d}(T(r), n_N)) = \text{rep}(\tilde{d}(T(r)), \mathcal{N}) \quad (7.2)
\]

The function \( \tau \), that realizes each step of the graph encoding, is specialized for GSOMSD as a \( \tau_M \), i.e. a self-organizing mapping equipped with the parameters \( \mathcal{W} \) of \( \mathcal{N} \) and appropriate metric:

\[
\tau_M : \mathcal{L} \times \mathcal{R} \times \cdots \times \mathcal{R} \rightarrow \mathcal{R} \quad \text{K times}
\]

Note that, in this context, \( \mathcal{R} \) is defined over the set of neurons \( \mathcal{N} \) of the SOM, and it corresponds to the code space \( \mathcal{X} \) used in the framework of Chapter 3. The learning algorithm \( \mathcal{A} \) is based on the learning rule of the self-organizing map.

Concerning the output function \( g \), it represents a mapping that can be added to the feature map once the type of information that the unsupervised method should produce (domain \( \mathcal{O} \)) has been decided. Hence, the function \( g \) can be used to extract information from the map and eventually to compute the output (prediction) values. This also allows us to use the models described by GSOMSD for classification/regression tasks through the appropriate modeling of the \( g \) function. In the GSOMSD paradigm an identity function has been considered for \( g \).

Let us clarify the approach by the following example that is an instance of the framework applied to the SOM-SD model.

**Example 7.1 (SOM-SD [71, 69])** Consider \( \mathcal{G} \) the domain of labeled DPAGs with super-source and \( G(s) \in \mathcal{G} \). A SOM-SD map is realized by an encoding function \( \tau_E : \mathcal{G} \rightarrow \mathcal{R} \), as in the following:

\[
\tau_E(G(s)) = \begin{cases} 
\text{nil}_{\mathcal{R}} & \text{if } G = \zeta \\
\tau_M(l(s), \tau_E(G(ch_1[s])), \ldots, \tau_E(G(ch_K[s]))) & \text{otherwise}
\end{cases} \quad (7.3)
\]

where \( \mathcal{R} \) represents the discrete output display space, i.e. the grid of SOM units represented by the lattice coordinates (e.g. in 2D), and \( \text{nil}_{\mathcal{R}} \) are void coordinates. \( \tau_M \) is a
SOM applied to the vertexes of $G$, where each vertex $v$ is represented in the SOM as a vector with the label and the subgraph code values, i.e. $[I(s) x(ch_1[v]), \ldots, x(ch_K[v])]$ whereby we used the same notations used for states of a SD-Recursive processing system, i.e. $x(ch_j[v]) = \tau_E(G(ch_j(v)))$, that is a vector associated to the winner unit for the $j$-th subgraphs of $v$. Here, $g$ is the identity function.

Naturally, the GSOMSD framework makes explicit use of the similarity measure and it is therefore specifically useful in the unsupervised scenario. The message expressed above is that the two approaches are equivalent and complementary.

Other Instances of the Framework  We could even allow activations of the neurons to be more general than simple one-dimensional vectors and also not corresponding to distances. A dimensionality $l$ of the activation vector of the neurons could be chosen, and $d \in \mathbb{R}^l$ and $d_R$ and $d_C$ would be chosen as $l$-dimensional vectors. In this case, the sum $\alpha \cdot d_C + \beta \cdot d_R + \beta \cdot d_R$ denotes scalar multiplication and addition of vectors. This setting allows us to model RSOM, for example, which, alternatively to the TKM, stores the leaky integration of the input directions instead of the integration of distances. We obtain an analog of RSOM for binary trees if we choose $R = (\mathbb{R}^N)^l$, $W_1(n_i) = W_2(n_i) = (e_1, \ldots, e_i)$, $e_i$ denoting the $i$th unit vector in $\mathbb{R}^N$, $d_R((r_1, \ldots, r_i), (r_1, \ldots, r_i)) = (r_1 \cdot r_i, \ldots, r_i \cdot r_i)$, rep as the identity, and $d_C(a_1, a_2) = a_1 - a_2$. Then the recursive dynamic becomes $d(a(t_1, t_2), n_i) = \alpha(a - W_0(n_i)) + \beta d(t_1, n_i) + \beta d(t_2, n_i)$. For out-degree 1 and $\beta = (1 - \alpha)$, this is the recursive formula of RSOM.

Of course, a computation of the combination of similarities which is more general than a simple weighted sum is also possible. In this case the computation $\alpha d_C + \beta d_R + \beta d_R : \mathcal{L}^2 \times \mathcal{R}^2 \times \mathcal{R}^2 \rightarrow \mathbb{R}^l$ would be substituted by a mapping $\Psi : \mathcal{L}^2 \times \mathcal{R}^2 \times \mathcal{R}^2 \rightarrow \mathbb{R}^l$ which maps the inputs given by the actual processed tree and the weights attached to a neuron, to the new activation of this neuron. This more general mechanism could for example model the so-called extended Kohonen feature map (EKFM) or the self-organizing temporal pattern recognizer (SOTPAR) [44, 85]. See [74] for details on these topics.

In the following, we will focus on the dynamics of GSOMSD, i.e. the states of the neurons are one-dimensional and correspond to distances or similarity values, and $\Psi$ has the specific form as defined above. We will discuss Hebbian learning and learning based on an energy function for GSOMSD. Moreover, we will describe the concrete setting for SOMSD and RecSOM adapted for binary trees. These two methods both rely on reasonable contexts, the winner, or the exponentially transformed activation of the entire map, respectively. SOMSD and RecSOM can be seen as two prototypical mechanisms with a different degree of information compression: SOMSD only stores the winner whereas RecSOM stores an activity profile which, due to the exponential transform, focuses on the winners, too. We will not describe the concrete setting for TKM, because this mechanism is restricted with respect to its capability of tree recognition: it can be proved that local contexts as used in TKM are not sufficient for the task of storing trees of arbitrary depth [74].
7.4 Hebbian Learning

In order to find a suitable neural map, we assume that a finite set of training data $\mathcal{D} = \{T_1, \ldots, T_d\} \subset \mathcal{B}$ is given. We would like to find a neural map such that the training data is represented as accurately as possible by the neural map. In the following, we discuss training methods which adapt the weights of the neurons, i.e. the triples $(x, r_1, r_2)$ attached to the neurons. This means that we assume that only the function $\mathcal{W}$ can be changed during training, for simplicity. Naturally, other parts could be adaptive as well, such as the weighting terms $\alpha$ and $\beta$, the computation of formal representations $\text{rep}$, or even the similarities $d_\mathcal{L}$ and $d_\mathcal{R}$. Note that some of the learning algorithms discussed in the following can be immediately transferred to the adaptation of additional parameters involved in e.g. $\text{rep}$.

There are various ways of learning in self-organizing maps. One of the simplest and most intuitive learning paradigms is Hebbian learning. It has got the advantage of producing simple update formulas and it does not require any further properties of e.g. the functions involved in $d$. However, it constitutes at a first glance a heuristic. A mathematical investigation of the learning behavior is often difficult. Alternatives can be found if an objective of the learning process is explicitly formulated in terms of a cost function which must be optimized. Then gradient descent methods or other optimization techniques, such as a genetic algorithm, can be applied. A gradient descent requires that the involved functions are differentiable and that $\mathcal{L}$ and $\mathcal{R}$ are real-vector spaces. Actually, most unsupervised Hebbian learning algorithms for simple vectors, although designed as a heuristic method, can be interpreted as a stochastic gradient descent on an appropriate cost function. Hence the two paradigms often yield to the same learning rules for simple vectors. We investigate the situation for structured data in the following. Here we will first discuss Hebbian learning for structured data and adapt several cost functions for self-organizing maps to our approach, afterward.

We would like to find a proper representation of all trees $T_i$ of the training set $\mathcal{D}$ and all its subtrees in the neural map. Since the computation of the recursive similarities for a tree $T_i$ uses the representation of all its subtrees, it can be expected that an adequate representation of $T_i$ can only be found in the neural map if all subtrees of $T_i$ are properly processed, too. Therefore, we assume that for each tree contained in $\mathcal{D}$ all its subtrees are contained in $\mathcal{D}$, too. This property of $\mathcal{D}$ is essential for efficient Hebbian learning. It can be expected that this property is beneficial for learning based on a cost function as well. Therefore we implicitly assume in the following that the training set $\mathcal{D}$ is enlarged to contain all subtrees of each $T_i \in \mathcal{D}$. Hebbian learning characterizes the idea of iteratively making the weights of the neuron which fires in response to a specific input more similar to the input. For this purpose some notation of ‘moving into the direction of’ is required. We assume the following two functions to be given:

- A function $\text{mv}_\mathcal{L} : \mathcal{L} \times \mathcal{L} \times \mathbb{R} \rightarrow \mathcal{L}$, $(w_1, w_2, \eta) \mapsto \text{mv}_\mathcal{L}(w_1, w_2, \eta)$ which allows an adaptation of the weights $w_1$ stored in the first position of a neuron into the direction of the input $w_2$ by degree $\eta$. If $\mathcal{L}$ is a real vector space, this function is usually just
given by the addition of vectors: \( mv_{\mathcal{L}}(w_1, w_2, \eta) = w_1 + \eta(w_2 - w_1) \). If discrete weights not embedded in a real vector space are dealt with, adaptation can be based on a discrete decision:

\[
    mv_{\mathcal{L}}(a, b, \eta) = \begin{cases} 
        a & \text{if } \eta \leq 0.5 \\
        b & \text{otherwise}
    \end{cases}
\]

If \( a \) and \( b \) are compared using the edit distance, for example, a specific number of operations which transform \( a \) into \( b \) could be applied depending on the value of \( \eta \), as proposed in [64] for the standard SOM.

- A function \( mv_{\mathcal{R}} : \mathcal{R} \times \mathcal{R} \times \mathbb{R} \to \mathcal{R}, (r_1, r_2, \eta) \mapsto mv_{\mathcal{R}}(r_1, r_2, \eta) \) which allows us to adapt the formal representations in the same way. If the formal representations are contained in a real vector space as in SOMSD or the recursive SOM, for example, this is given by the addition of vectors, too. Alternatively, we can use discrete transformations if \( \mathcal{R} \) consists of a discrete set.

Note that we can also handle discrete formal representations in this way. We first formulate Hebbian training for the analogue of SOM-training for the GSOMSD. Here, an additional ingredient, the neighborhood structure or topology is used: assume there is given a neighborhood function

\[
    nh : \mathcal{N} \times \mathcal{N} \to \mathbb{R}
\]

which measures the distance of two neurons with respect to the topology. Often, a lattice structure is fixed a priori and this lattice of neurons is spread over the data during training. It might consist of a two-dimensional lattice and the neurons enumerated by \((1, 1), (1, 2), \ldots, (N_1, N_2)\) accordingly. For this two-dimensional lattice \( nh(n_{(i_1, j_1)}, n_{(i_2, j_2)}) \) could be the distance of the indexes \(|i_1 - i_2| + |j_1 - j_2|\), for example. However, the lattice structure could be more complex such as a hexagonal grid structure or a grid with an exponentially increasing number of neighbors [141]. In this case, an algorithm can be written as follows:

initialize the weights at random
repeat: choose some training pattern \( T \)
    for all subtrees \( t = a(t_1, t_2) \) in \( T \) in inverse topological order:
        compute \( \tilde{d}(a(t_1, t_2), n_i) \) for all neurons \( n_i \)
        compute the neuron \( n_{i^*} \) with greatest similarity
        adapt the weights of all neurons \( n_i \) simultaneously:
            \( \mathcal{W}_0(n_i) := mv_{\mathcal{L}}(\mathcal{W}_0(n_i), a, \eta(nh(n_i, n_{i^*}))) \)
            \( \mathcal{W}_1(n_i) := mv_{\mathcal{R}}(\mathcal{W}_1(n_i), R_1, \eta(nh(n_i, n_{i^*}))) \)
            \( \mathcal{W}_2(n_i) := mv_{\mathcal{R}}(\mathcal{W}_2(n_i), R_2, \eta(nh(n_i, n_{i^*}))) \)

where

\[
    R_j = \begin{cases} 
        r_{\zeta} & \text{if } t_j = \zeta \\
        \text{rep}(\tilde{d}(t_j, n_1), \ldots, \tilde{d}(t_j, n_N)) & \text{otherwise}
    \end{cases}
\]
for $j \in \{1, 2\}$; $\eta : \mathbb{R} \to \mathbb{R}$ is a monotonically decreasing function. Often, $\eta(x) = \eta_0 \exp(-x/\sigma_{nh}^2)$ where $\eta_0 > 0$ is a learning rate which is decreased at each step in order to ensure convergence; $\sigma > 0$ is a term which determines the size of the neighborhood which is taken into account. Usually, $\sigma$ is decreased during training, too. The dependence $(\ast)$ requires the recomputation of the recursive similarity for all subtrees of $a(t_1, t_2)$ after changing the weights. This is, of course, very costly. Therefore, the recursive similarity $\tilde{d}$ is usually computed only once for each subtree of a tree, and the values are uniformly used for the update. This approximation does not change training much for small learning rates. Hebbian learning has been successfully applied for both, SOMSD and RecSOM. See for example the literature for the classification of time series or images [71, 69, 153, 167, 169].

Alternatives to SOM

We would like to point out that the scaling term $\eta(nh(n_i, n_{io}))$ might be substituted by a more general function $\eta$ which depends not only on the winner but on the whole lattice, i.e. it gets as input the distances $\tilde{d}(a(t_1, t_2), n_i)$ of all neurons $n_i$. This more general setting covers updates based on the soft-min function or the ranking of all neurons, for example.

Two alternatives to SOM proposed in the neural networks literature are vector quantization (VQ) and neural gas (NG) as introduced in Section 7.2. We obtain an analogy for VQ in the case of structured data if we choose as neighborhood structure

$$\eta(nh(n_i, n_j)) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

This yields a learning algorithm that can handle both, sequences and trees, because of the respective context; however, no topology preservation takes place. A lattice structure is not accounted for by the VQ algorithm. VQ would constitute an alternative to SOM training for the RecSOM. For SOMSD, the metric $d_R$ of the internal representations $R$ in SOMSD would yield random values for VQ which depend on the initialization of the map and the order of presentation of the examples. As a consequence of this argumentation, adaptation of the representations is necessary to apply VQ in this case. Assume the neurons being enumerated by $1, \ldots, N$. Then the alternative $rep_R(x_1, \ldots, x_N) = (0, \ldots, 0, 1, 0, \ldots, 0)$ as the vector with entry 1 at place $i$ iff $x_i \leq x_j$ for all $j \neq i$ instead of $rep_R$ has the effect of decoupling the indexes. Note that we can easily apply a form of Hebbian learning to these representations. A fast computation can look as follows: choose the similarity measure $d_R$ as the dot product and restrict $R = \{r \in \mathbb{R}^N | \sum r_i = 1\}$. We can store the formal representations attached to a neuron in the form $(r_1, \ldots, r_N)/c$ where $c$ is a global scaling factor for the components for each representation. Given a tree $a(t_1, t_2)$ such that neuron $n^*$ is the winner for $a(t_1, t_2)$ and the neurons $n_1$ and $n_2$ are the winners for $t_1$ and $t_2$, respectively, we adapt the weight $\mathcal{W}_1(n^*)$ and $\mathcal{W}_2(n^*)$ in the following way with Hebbian learning: the component representing $n_1$ and $n_2$, respectively, is increased by a fixed constant $\epsilon$, the global scaling
factors $\epsilon$ for the two representations are increased by $\epsilon$. As a consequence, we obtain implicit normalization of the formal representations.

The neural gas algorithm as explained in Section 7.2 determines the respective neighborhood from the given training example and its distances to all neurons. Hence the update in the above algorithm given a tree $t_j$ becomes

\[
W_0(n_i) := mv^L(W_0(n_i), a, \eta(rk(j, i)))
\]

\[
W_1(n_i) := mv^R(W_1(n_i), R_1, \eta(rk(j, i)))
\]

\[
W_2(n_i) := mv^R(W_2(n_i), R_2, \eta(rk(j, i)))
\]

where $rk(j, i)$ denotes the rank of the neuron $n_i$ ordered according to the current distance from the input, i.e. if the tree $t_j$ is currently processed, $rk(j, i)$ denotes the number of neurons $n_k$ such that $\hat{d}(t_j, n_i) < \hat{d}(t_j, n_k)$. $\eta(x)$ can for example have the form $\eta_0 \exp(-x/\sigma^2_{nh})$. This update makes sure that neurons which have similar activation adapt the respective weights into the same direction. Finally, those neurons are neighbors according to this data driven implicit lattice, which constitute the first and second winner for at least one data point. Naturally, the resulting neighborhood structure is less regular. It need not be in accordance with the lattice, and it proposes an alternative to SOM with data oriented lattice. Like in the case of vector quantization, the representation $R$ in SOMSD requires adaptation. The distance between indexes should be dynamically computed during the training algorithm according to the ranking with respect to the given data point. This behavior can be obtained if we choose $R$ as $\mathbb{R}^N$, $d_R$ as the squared Euclidean metric, and if we choose the representation function $rep_N$ where the $i$th component of $rep_N(x_1, \ldots, x_n)$ is the number of $x_j$ such that $x_j < x_i$, i.e.

\[
rep_N(x_1, \ldots, x_n) = (rk(x_1), \ldots, rk(x_N)).
\]

Hereby, $rk(x_i)$ denotes the number of $x_k$ with $x_k < x_i$.

### 7.5 Learning as Cost Minimization

For SOM, NG, and VQ effort has been made to derive the Hebbian learning rules alternatively as stochastic gradient methods of appropriate cost functions for simple vector data. Assume that a given differentiable cost function $E$ can be written in the form $\sum a_i E(a_i)$, $a_i$ denoting a training pattern; a stochastic gradient descent with respect to weights $w$ then has the form

\[
\text{initialize the weights at random}
\]

\[
\text{repeat: choose a training pattern } a_i
\]

\[
\text{update } w := w - \eta \cdot \frac{\partial E(a_i)}{\partial w}
\]

where $\eta > 0$ is the learning rate. Note that Hebbian learning has a similar form. Assumed we could prove that Hebbian learning obeys a stochastic gradient descent with appropriate
choice of $\eta$ like in the case of simple vectors, then a prior mathematic objective can be identified and alternative optimization methods, in particular global optimization methods such as simulated annealing can be used as an alternative. Moreover, there exist well-known guarantees for the convergence of a stochastic gradient descent if the learning rate $\eta$ (which may vary over time) fulfills certain properties.

We first have a look at SOM, VQ, and NG for simple vectors. Denote the simple labels or vectors used for training by $a_i$. The weights of the map are denoted by $w_j$. The cost function of VQ for the case of simple vectors has the form

$$\frac{1}{2} \sum_i \sum_j \chi(a_i, w_j) \| a_i - w_j \|^2$$

where $\chi(a_i, w_j)$ denotes the characteristic function of the receptive field of $w_j$, i.e. it is 1 if $n_j$ is the winner for $a_i$ and 0 otherwise. Taking the derivatives with respect to the weights $w_j$ gives the learning rule of VQ. Note, the derivatives of $\chi$ are thereby computed e.g. using the $\delta$-function. We here ignore this point and provide a detailed derivation of the formulas including the borders of $\chi$ for the structured case. The cost function of NG is of the form [121]

$$\frac{1}{2} \sum_i \sum_j \eta(\text{rk}(i, j)) \| a_i - w_j \|^2$$

where $\text{rk}(i, j)$ denotes the rank of neuron $n_j$ if the neurons are sorted according to their distance from the actual data point. SOM itself does not possess an cost function which could be transferred to the continuous case, i.e. if a data distribution instead of a finite training set is considered. The article [82], for example, proposes a cost function for a slightly modified version of SOM:

$$\frac{1}{2} \sum_i \sum_j \chi_S(i, j) \sum_k \eta(\text{nh}(n_j, n_k)) \| a_i - w_k \|^2$$

where here $\chi_S(i, j)$ denotes the characteristic function of the receptive field of the winner whereby a slightly modified definition of the winner is used:

$$\chi_S(i, j) = \begin{cases} 1 & \text{if } \sum_k \eta(\text{nh}(n_j, n_k)) \| a_i - w_k \|^2 \leq \sum_k \eta(\text{nh}(n_j, n_k)) \| a_i - w_{j'} \|^2 \text{ for all } j' \\ 0 & \text{otherwise} \end{cases}$$

Hence the winner in the above cost function as well as the modified learning rule of SOM is not the neuron with the smallest distance but the neuron with the smallest averaged distance with respect to the local neighborhood.

Obviously, all of the above cost functions have the form

$$\sum_i f(\| a_i - w_1 \|^2, \ldots, \| a_i - w_N \|^2).$$

This general scheme allows an immediate transfer of the above cost functions to the structured case: the term $\| a_i - w_j \|^2$ is substituted by the respective recursive similarity of the
neurons given a tree structure. We again assume that $\mathcal{D} = \{ T_1, \ldots, T_d \}$ is a given set of trees which are used for training. Since usually together with the trees all subtrees should be represented properly, i.e. its corresponding quantization error should be small, we assume that $\mathcal{D}$ is complete with respect to subtrees. Given a neural map for structured data with neurons $n_i$, the general cost function has the form

$$E = \sum_{t_i \in \mathcal{D}} f(\hat{d}(T_i, N))$$

where $\hat{d}(T_i, N)$ denotes the vector $(\hat{d}(T_i, n_1), \ldots, \hat{d}(T_i, n_N))$. If $f$ is chosen corresponding to the above cost functions of VQ, NG, or SOM, this measure yields an objective for the structured case. Taking the derivatives allows us to obtain formulas for a stochastic gradient descent in the structured case. We will see that, unlike the case of simple data, the gradient descent methods for the cost functions of NG, SOM, and VQ differ from the respective Hebbian learning algorithms. Hence unlike the case of simple vectors, Hebbian learning as introduced above is only an approximation of a stochastic gradient descent on the respective cost functions of VQ, NG, or SOM.

We start computing the derivatives of the above general function $E$ with respect to the weights of the neurons. For this purpose, we assume that $\mathcal{L}$ and $\mathcal{R}$ are contained in a real vector space, and here we first assume that all involved functions including $f$, rep, $d_{\mathcal{R}}$, and $d_{\mathcal{L}}$ are differentiable (they are not e.g. for SOMSD, we will discuss this point later). Assume the derivative with respect to a weight $W_l(n_j)$ for $I \in \{0, 1, 2\}$ and $J \in \{1, \ldots, N\}$ is to be computed. We find

$$\frac{\partial f(\hat{d}(T_i, N))}{\partial W_l(n_j)} = \sum_{j=1}^{N} \frac{\partial f(\hat{d}(T_i, N))}{\partial \hat{d}(T_i, n_j)} \cdot \frac{\partial \hat{d}(T_i, n_j)}{\partial W_l(n_j)}$$

The first part depends on the respective cost function, i.e. the choice of $f$. It often involves the use of $\delta$-functions for computing the derivatives because $f$ is built of characteristic functions. The second component can be computed as follows:

- define $\partial_i f(x_1, \ldots, x_n) := \frac{\partial f(x_1, \ldots, x_n)}{\partial x_i}$ for $i \leq n$ as a shorthand notation.
- If $f = (f_1, \ldots, f_m)$ is a vector of functions, then the term $\partial_i f(x_1, \ldots, x_n)$ denotes the vector $(\partial_i f_1(x_1, \ldots, x_n), \ldots, \partial_i f_m(x_1, \ldots, x_n))$. If $x_i$ is a vector $(x_i^1, \ldots, x_i^m)$, we define $\partial_i f(x_1, \ldots, x_n) = (\partial f(x_1, \ldots, x_n)/\partial x_i^1, \ldots, \partial f(x_1, \ldots, x_n)/\partial x_i^m)$.

As above, we use the abbreviation

$$R_j = \begin{cases} r_{\zeta} & \text{if } t_j = \zeta \\ \text{rep}(\hat{d}(t_j, n_1), \ldots, \hat{d}(t_j, n_N)) & \text{otherwise} \end{cases}$$

for $j \in \{1, 2\}$ and we denote the corresponding derivative with respect to a variable $x$ by

$$\partial R_{j,x} = \begin{cases} 0 & \text{if } t_j = \zeta \\ \sum_{l=1}^{N} \partial_l \text{rep}(\hat{d}(t_j, n_1), \ldots, \hat{d}(t_j, n_N)) \cdot \partial \hat{d}(t_j, n_l)/\partial x & \text{otherwise} \end{cases}$$
for $j \in \{1, 2\}$. Then we have
\[
\partial \tilde{d}(a(t_1, t_2), n_j)/\partial \mathcal{W}_0(n_j) = \alpha \delta_{j,t} \partial_0 \mathcal{d}_c(a, \mathcal{W}_0(n_j)) \\
+ \beta \partial_1 \mathcal{d}_R(R_1, \mathcal{W}_1(n_j)) \cdot \partial R_1 \mathcal{W}_0(n_j) \\
+ \beta \partial_1 \mathcal{d}_R(R_2, \mathcal{W}_2(n_j)) \cdot \partial R_2 \mathcal{W}_0(n_j)
\]
(1.1)
(1.2)
(1.3)

for the first components of the weights where $\delta_{i,j} \in \{0, 1\}$ is the Kronecker symbol with $\delta_{i,j} = 1 \iff i = j$. \cdot \cdot denotes the dot product if $\mathcal{R}$ is contained in a vector space of a dimension larger than one. For the formal representations attached to a neuron we find
\[
\partial \tilde{d}(a(t_1, t_2), n_j)/\partial \mathcal{W}_1(n_j) = \beta \delta_{j,t} \partial_1 \mathcal{d}_R(R_1, \mathcal{W}_1(n_j)) \\
+ \beta \partial_1 \mathcal{d}_R(R_1, \mathcal{W}_1(n_j)) \cdot \partial R_1 \mathcal{W}_1(n_j) \\
+ \beta \partial_1 \mathcal{d}_R(R_2, \mathcal{W}_2(n_j)) \cdot \partial R_2 \mathcal{W}_1(n_j)
\]
(2.1)
(2.2)
(2.3)

and
\[
\partial \tilde{d}(a(t_1, t_2), n_j)/\partial \mathcal{W}_2(n_j) = \beta \delta_{j,t} \partial_2 \mathcal{d}_R(R_2, \mathcal{W}_2(n_j)) \\
+ \beta \partial_1 \mathcal{d}_R(R_1, \mathcal{W}_1(n_j)) \cdot \partial R_1 \mathcal{W}_2(n_j) \\
+ \beta \partial_1 \mathcal{d}_R(R_2, \mathcal{W}_2(n_j)) \cdot \partial R_2 \mathcal{W}_2(n_j)
\]
(3.1)
(3.2)
(3.3)

Hence the derivatives can be computed recursively over the depth of the tree. Starting at the leaves, we obtain formulas for the derivatives of the respective activation with respect to the weights of the neurons.

Note that the first summands of the derivatives yield terms which also occur in Hebbian learning: $\partial_2 d_\mathcal{R}(a, b)$ and $\partial_1 d_\mathcal{R}(a, b)$ coincide for the squared Euclidean metric with the terms $-2(a - b)$ or $2(a - b)$, respectively. Hence we get the original Hebbian learning rules if we only consider the first summands and involve the respective terms arising from $f$ for NG, VQ, and SOM. This argumentation has two consequences: first, it offers the possibility of transferring the above rules to situations where discrete data are involved and hence $d_\mathcal{C}$ and $d_\mathcal{R}$ possibly not differentiable. The above argument aims at substituting the derivative $\partial_2 d_\mathcal{C}$ in the above formulas by the function $mv_\mathcal{C}$ and the derivative $\partial_2 d_\mathcal{R}$ by the function $mv_\mathcal{R}$ in order to obtain a heuristic training method for the general case where the labels or formal representations are not given by a real vector space.

Second, Hebbian learning is different from a stochastic gradient descent if structured data are involved. Hebbian learning disregards the contribution of the subtrees because it cuts the latter two summands in the above recursive formulas for the derivatives. Hence unlike the simple vector case where Hebbian learning and a stochastic gradient descent coincide, structured data causes differences for the methods. However, the factor $\beta$ is usually smaller than 1; i.e. the weighting factor for the two additional terms in the above gradient formulas vanishes exponentially with the depth since it is multiplied in each recursive step by the small factor $\beta$. Therefore, Hebbian learning can be seen as an efficient approximation of the precise stochastic gradient descent.
Energy Function of VQ

The cost function of vector quantization reads in our case as

\[ E_V(T) := \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{N} \chi(T_i, n_j) \tilde{d}(T_i, n_j) \]

where

\[ \chi(T_i, n_j) = \begin{cases} 1 & \text{if } \tilde{d}(T_i, n_j) \text{ is minimal} \\ 0 & \text{otherwise} \end{cases} \]

Hence \( f(\tilde{d}(T_i, N)) \) has here the form \( \frac{1}{2} \sum_j \chi(T_i, n_j) \tilde{d}(T_i, n_j) \). The only term in the above derivatives which is specific for \( f \) reads as \( \partial f(\tilde{d}(a_k, N))/\partial \tilde{d}(a_k, n_j) \). We here compute the term for VQ involving in particular the precise derivative of formulas for the borders of the receptive fields where \( \delta \)-functions are to be used.

Note that

\[ \chi(T_i, n_j) = H \left( \sum_{l} H(\tilde{d}(T_i, n_l) - \tilde{d}(T_i, n_j)) - N + 1.5 \right) \]

where \( H \) is the Heaviside function with

\[ H(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \]

The derivative of \( H \) is the function \( \delta \) which is a symmetric function with \( \delta(x) \neq 0 \iff x = 0 \). Hence we find

\[ \frac{\partial f(\tilde{d}(T_i, N))}{\partial \tilde{d}(T_i, n_l)} = \frac{1}{2} \sum_j \frac{\partial \chi(T_i, n_j)}{\partial \tilde{d}(T_i, n_l)} \cdot \tilde{d}(T_i, n_j) + \frac{1}{2} \sum_j \chi(T_i, n_j) \cdot \frac{\partial \tilde{d}(T_i, n_j)}{\partial \tilde{d}(T_i, n_l)} \]

The second term yields the standard Hebb factor of VQ, i.e. the contribution \( \frac{1}{2} \chi(T_i, n_l) \) to the above general formulas for the derivatives. The first summand equals

\[ \frac{1}{2} \sum_{jk} \delta(\tilde{d}(T_i, n_k) - \tilde{d}(T_i, n_j)) \left( \frac{\partial \tilde{d}(T_i, n_k)}{\partial \tilde{d}(T_i, n_l)} - \frac{\partial \tilde{d}(T_i, n_j)}{\partial \tilde{d}(T_i, n_l)} \right) \cdot \delta \left( \sum_k H(\tilde{d}(T_i, n_k) - \tilde{d}(T_i, n_j)) - N + 1.5 \right) \tilde{d}(T_i, n_j) \]

Hence it vanishes because \( \delta \) is symmetric, and the terms involving \( \delta \) do not equal zero iff the role of \( \tilde{d}(T_i, n_k) \) and \( \tilde{d}(T_i, n_j) \) can be changed.

Therefore, we obtain the Hebbian update rules from a gradient descent, if we only take the summands (1.1), (2.1), and (3.1) into account and if we ignore all remaining summands (1.2), (1.3), (2.2), (2.3), (3.2), (3.3). Note that these latter summands contain at least one of the terms \( \alpha^i \beta \) or \( \alpha^{i+1} \) for \( i \geq 1 \), which usually is small. The precise
update rule which we obtain from this gradient approach, disregarding the contributions of substructures, looks as follows if the Euclidean metric is used:

\[
\begin{align*}
\mathcal{W}_0(n_t) &:= \mathcal{W}_0(n_t) + \eta \cdot \alpha \cdot \chi(T_t, n_t)(a - \mathcal{W}_0(n_t)) \\
\mathcal{W}_1(n_t) &:= \mathcal{W}_1(n_t) + \eta \cdot \beta \cdot \chi(T_t, n_t)(R_t - \mathcal{W}_1(n_t)) \\
\mathcal{W}_2(n_t) &:= \mathcal{W}_2(n_t) + \eta \cdot \beta \cdot \chi(T_t, n_t)(R_t - \mathcal{W}_2(n_t))
\end{align*}
\]

where \(a\) denotes the current label and \(R_t\) denotes the representation of the subtrees.

Alternatively, it is possible to transfer the above three proposals of computing the precise gradients to the recursive SOM directly since the formal representation for trees lies in a real vector space and the involved functions are differentiable. Then the component number \(l\) of \(\delta_l \text{rep}_R(x_1, \ldots, x_N)\) yields \(-\exp(-x_l)\), all other components are zero.

For SOMSD, the above method cannot be applied directly: the involved function \(\text{rep}_S\) is not differentiable; moreover, it can be expected that \(\text{rep}_S\) should be substituted by \(\text{rep}_V\) as before, which encodes winners in a unary fashion instead of referring to a lattice. Although \(\text{rep}_V\) is not differentiable, it can be approximated up to any desired degree of accuracy with a differentiable function. We hence approximate using the soft-min function

\[
\text{rep}_V(x_1, \ldots, x_N) \approx \left( \frac{\exp(-x_1/\gamma)}{\sum_l \exp(-x_l/\gamma)}, \ldots, \frac{\exp(-x_N/\gamma)}{\sum_l \exp(-x_l/\gamma)} \right)
\]

where \(\gamma > 0\) controls the quality of the approximation. The original function is recovered for \(\gamma \to 0\) except for situations where the minimum is not unique.

**Energy Function of SOM**

Simple vector quantization formalizes that the quantization error should be minimized. Adaptations which try to match the topology of the data and the lattice structure of the neuron require enhanced cost functions such that the degree of topology preservation is taken into account. The self-organizing map of Kohonen does not possess a cost function which can be transferred to the continuous case. Hence we adapt a variation proposed in [82] which uses a slightly different notation of the winner. Then a cost function can be found such that a stochastic gradient descent for simple vectors yields a variation of Hebbian learning for SOM with a slightly different notation of the winner. This cost function when adapted to structured data has the form

\[
E_S(T) = \frac{1}{2} \sum_{t=1}^{T} \sum_{j=1}^{N} \chi_S(T_t, n_j) \sum_{k=1}^{N} \eta(nh(n_j, n_k))d(T_t, n_k)
\]

where \(nh : \mathcal{N} \times \mathcal{N} \to \mathbb{R}\) describes the neighborhood function of the lattice and \(\eta(x) = \eta_0 \exp(-x/\sigma_{nh}^2)\) with \(\eta_0 > 0, \sigma_{nh} > 0\) provides a scaling factor for the update of neighbors. The function

\[
\chi_S(T_t, n_j) = \begin{cases} 
1 & \text{if } \sum_{k=1}^{N} \eta(nh(n_j, n_k))d(T_t, n_k)^2 \text{ is minimal} \\
0 & \text{otherwise}
\end{cases}
\]
determines the winner. Here the winner is the neuron with optimum weighted distance with respect to the neighborhood; i.e. the winner is not the closest neuron but the neuron with minimum averaged distance with respect to the neighborhood. Given a tree \( T_i \), the above cost function induces the function \( f \) of the form
\[
\frac{1}{2} \sum_j \chi_s(T_i, n_j) \sum_k \eta(nh(n_j, n_k)) \tilde{d}(T_i, n_k)
\]
The derivative with respect to \( \tilde{d}(T_i, n_l) \) we are interested in in order to apply the above general formulas can be computed as
\[
\frac{1}{2} \sum_j \delta \left( \sum_k \sum_l H \left( \sum_k \eta(nh(n_l, n_k)) \tilde{d}(T_i, n_k) - \sum_k \eta(nh(n_l, n_k)) \tilde{d}(T_i, n_k) \right) - N + 1.5 \right) .
\]
This vanishes because \( \delta \) is symmetric and the above \( \delta \)-terms are non-vanishing only if the weighted distances of \( n_o \) and \( n_j \) can be substituted.

Therefore, we obtain the slightly altered Hebbian update rules from a gradient descent, if we again disregard the summands (1.2), (1.3), (2.2), (2.3), (3.2), (3.3):
\[
\mathcal{W}_0(n_l) := \mathcal{W}_0(n_l) + \eta \cdot \alpha \cdot \sum_k \chi_s(T_i, n_k) \eta(nh(n_k, n_l))(a - \mathcal{W}_0(n_l))
\]
\[
\mathcal{W}_1(n_l) := \mathcal{W}_1(n_l) + \eta \cdot \beta \cdot \sum_k \chi_s(T_i, n_k) \eta(nh(n_k, n_l))(R_1 - \mathcal{W}_1(n_l))
\]
\[
\mathcal{W}_2(n_l) := \mathcal{W}_2(n_l) + \eta \cdot \beta \cdot \sum_k \chi_s(T_i, n_k) \eta(nh(n_k, n_l))(R_2 - \mathcal{W}_2(n_l))
\]
where as before \( a \) denotes the current label and \( R_i \) denotes the representation of the subtrees.

If we would like to perform an exact gradient descent, the specific adaptations of this method for the recursive SOM and SOMSD follow the same line as in the previous
A GENERAL FRAMEWORK FOR UNSUPERVISED PROCESSING OF SD

Note that we could use the alternatives of forward or backward propagation as described in [74]. Again the function $\text{rep}_S$ of SOMSD is not differentiable. However, we can approximate the function $\text{rep}$ up to every desired degree. Denote the indexes of the neurons by $i(1), \ldots, i(N)$. The representation function which computes the index of the minimum input can be approximated by the soft-min function

$$
\text{rep}_S(x_1, \ldots, x_n) \approx \sum_{i=1}^{N} \frac{\exp(-x_i/\gamma)}{\sum_i \exp(-x_i/\gamma)}
$$

where $\gamma > 0$ controls the quality of the approximation. This approximation is differentiable. The original function is recovered for $\gamma \to 0$ except for situations where the minimum is not unique. Note that tie breaks are here solved by an interpolation between the indexes of the winners instead of the choice of the first winner.

Note that for SOMSD the lattice of neurons has to be chosen a priori. It can be expected that a two-dimensional regular grid is appropriate only for very restricted data sets due to the more complex data structure. The experiments of [69] show that different structures are stored in different clusters in a regular grid map. The lattice can combine different structures only partially due to a combinatorial explosion of possible neighbors: already for sequences over a finite alphabet, the number of ways in which a sequence can be expanded by $i$ terms constitutes an exponential number in $i$. An even more rapid increase can be found for tree structures, of course. One could expect that lattices which mirror this behavior of an exponentially increasing neighborhood like the hyperbolic SOM [141] could be a good choice for huge data sets which cover a variety of different structures.

Energy Function of NG

The neural gas algorithm constitutes another alternative where the lattice structure is not given a priori but emerges according to the data structure. Given a data point, those neurons are considered as neighbors which have similar places in the ranking according to the distance from the data point. The cost function of NG adapted to structured data has the following form:

$$
\mathcal{E}_N(T) = \frac{1}{2} \sum_{i=1}^{d} \sum_{j=1}^{N} \eta(rk(i, j)) \tilde{d}(T_i, n_j).
$$

$rk(i, j)$ equals the number of neurons $k$ such that $\tilde{d}(T_i, n_k) < \tilde{d}(T_i, n_j)$. Again $\eta(x) = \eta_0 \exp(-x/\sigma_{nh}^2)$ is a scaling factor. If we put this cost function in the above general form, we find for the function $f$

$$
f(\tilde{d}(T_i, N)) = \frac{1}{2} \sum_j \eta(rk(i, j)) \tilde{d}(T_i, n_j)$$
7.5. LEARNING AS COST MINIMIZATION 147

The derivative with respect to \( \tilde{d}(T_i, n_t) \) yields

\[
\frac{1}{2} \sum_j \frac{\partial \eta(rk(i, j))}{\partial \tilde{d}(T_i, n_j)} \tilde{d}(T_i, n_j) + \frac{1}{2} \sum_j \eta(rk(i, j)) \frac{\partial \tilde{d}(T_i, n_j)}{\partial \tilde{d}(T_i, n_t)}
\]

where the second term yields the usual update rule for NG, i.e. the contribution \( \frac{1}{2} \eta(rk(i, l)) \) to the above formulas. The first term vanishes, which can again explicitly be computed using the identity

\[
\text{rk}(i, j) = \sum_k H(\tilde{d}(T_i, n_j) - \tilde{d}(T_i, n_k))
\]

Then the first term becomes

\[
\frac{1}{2} \sum_{j, k} \frac{\partial \eta(\text{rk}(i, j))}{\partial \text{rk}(i, j)} \delta(\tilde{d}(T_i, n_j) - \tilde{d}(T_i, n_k)) \tilde{d}(T_i, n_j) \left( \frac{\partial \tilde{d}(T_i, n_j)}{\partial \tilde{d}(T_i, n_t)} - \frac{\partial \tilde{d}(T_i, n_k)}{\partial \tilde{d}(T_i, n_t)} \right)
\]

This vanishes due to the properties of \( \delta \).

The Hebbian update rules hence result from a gradient descent, if we disregard the summands (1.2), (1.3), (2.2), (2.3), (3.2), (3.3):

\[
\mathcal{W}_0(n_t) := \mathcal{W}_0(n_t) + \eta \cdot \alpha \cdot \eta(\text{rk}(i, l))(a - \mathcal{W}_0(n_t))
\]

\[
\mathcal{W}_1(n_t) := \mathcal{W}_1(n_t) + \eta \cdot \beta \cdot \eta(\text{rk}(i, l))(R_1 - \mathcal{W}_1(n_t))
\]

\[
\mathcal{W}_2(n_t) := \mathcal{W}_2(n_t) + \eta \cdot \beta \cdot \eta(\text{rk}(i, l))(R_2 - \mathcal{W}_2(n_t))
\]

where as before \( a \) denotes the current label and \( R_i \) denotes the representation of the subtrees.

As before, adaptation for SOMSD is necessary if the precise gradients should be computed instead of this approximation. Since no lattice is given a priori, the space of indexes which constitutes \( \mathcal{R} \) in the original approach consists of points which do not possess any prior metrical information. The distance between indexes should be dynamically computed during the training algorithm according to the ranking with respect to the respective data point. Hence we again choose the representation function

\[
\text{rep}_N(x_1, \ldots, x_n) = (\text{rk}(x_1), \ldots, \text{rk}(x_N)) = \left( \sum_{j=1}^N H(x_1 - x_j), \ldots, \sum_{j=1}^N H(x_N - x_j) \right)
\]

where \( \text{rk}(x_i) \) denotes the rank of \( x_i \) if the values \( x_1, \ldots, x_N \) are ordered according to their size. Since this function is not differentiable, we have to approximate by a differentiable function. We can substitute \( H \) by a sigmoidal approximation \( \text{sgd}_\gamma(x) := \text{sgd}(x / \gamma) \rightarrow H(x) \) for \( \gamma > 0, \gamma \to 0 \) and \( x \neq 0 \) where \( \text{sgd}(x) = (1 + \exp(-x))^{-1} \).

Note that the same emphasis is laid on each position of the ranking in this formulation. It can be expected that the position of winning units is more important than the rank of neurons which are far away. Hence one could modify the formal representation of trees such that winning neurons are ranked higher. One possibility is to include an exponential weighting \( \text{rep}(x_1, \ldots, x_n) = (\exp(-\text{rk}_k(x_1)), \ldots, \exp(-\text{rk}_k(x_N))) \). This has
the effect that deviations for neurons which are close to the actual tree are ranked higher than neurons which are further away. Alternatively, we could use a simple truncation \( \text{rep}(x_1, \ldots, x_n) = (rk_k(x_1), \ldots, rk_k(x_N)) \) where

\[
rk_k(x_i) = \begin{cases} 
  rk(x_i) & \text{if } rk(x_i) < k \\
  k & \text{otherwise}
\end{cases}
\]

for some \( k \in \{1, \ldots, N\} \). This has the effect that the precise position of only the closest \( k \) neurons is of interest, the others are identified in the formal representation.

### 7.6 Properties of \( \text{rep} \)

The function \( \text{rep} \) which is used in order to obtain a formal representation of the trees is obviously a crucial part of the dynamics. It has to be made sure that the important information of the distributed representation of a processed tree is preserved through this function \( \text{rep} \). At the same time, \( \text{rep} \) should be efficiently computable and noise tolerant. We want to discuss several properties which seem of importance concerning the formal representations, i.e. \( \text{rep}, \mathcal{R}, \) and \( d_{\mathcal{R}} \). We start with a simple observation:

- The function \( \text{rep} \) should map to a bounded domain, i.e. it should be possible to uniformly bound the value \( d_{\mathcal{R}}(r_1, r_2) \) for all representations \( r_1 \) and \( r_2 \) which might occur during a recursive computation or as a component of a weight.

Note that unless in the standard SOM, this property is not automatically fulfilled if all labels of trees come from a bounded domain, since distances may blow up through the recursive processing. Assume, for example, that we drop the exponential function when computing the formal representation \( \text{rep} \) for the recursive SOM, we would obtain a blow up of the values during the recursive computation: the distances could accumulate during the recursive computation such that no universal bound could be found unless the height of trees to be processed is limited. Both, for SOMSD and the recursive SOM this behavior is prohibited due to the compactness of the respective domain \( \mathcal{R} \).

Another obvious demand on \( \text{rep} \) is the following:

- \( \text{rep} \) should be noise tolerant, i.e. if \( x \in \mathbb{R}^N \) is disrupted by noise to \( x + \eta \), the distance between \( d_{\mathcal{R}}(\text{rep}(x), r) \) and \( d_{\mathcal{R}}(\text{rep}(x + \eta), r) \) should be small for all possible \( r \in \mathcal{R} \).

This point is of particular importance, since small errors of the computation e.g. due to noisy inputs may blow up easily at recursive processing. Hence some focus on important parts of the distributed representation of trees should be introduced when computing \( \text{rep} \). Assumed that important information lies in the winner, then clearly SOMSD precisely extracts this point. The recursive SOM amplifies large values and smooths vanishing terms via the exponential function. Another fact makes insensitivity with respect to noise a crucial part of the design: the dimensionality of the input for \( \text{rep} \) is commonly high
dimensional since it coincides with the number of neurons of the map. The formal representations may be high dimensional, too, as in the case of the recursive SOM. Hence errors may accumulate simply due to the curse of dimensionality.

One can explicitly compute the sensitivity with respect to noise for concrete implementations of \( \text{rep} \): assume \( x = (x_1, \ldots, x_n) \in \mathbb{R}^N \) is an activation, \( r \) a formal representation, and \( \eta = (\eta_1, \ldots, \eta_N) \subset \mathbb{R}^N \) a vector of i.i.d. noise variables with zero mean and finite variance. We are interested in the term

\[
E(\|d_{\mathcal{R}}(\text{rep}(x), r) - d_{\mathcal{R}}(\text{rep}(x + \eta), r)\|) \quad (\star)
\]

where \( E \) denotes expectation with respect to \( \eta \). If \( \text{rep} \) is simply the identity and \( d_{\mathcal{R}} \) the Euclidean distance, one obtains

\[
(\star) = E(\|\sum (x_i - r_i)^2 - \sum (x_i + \eta_i - r_i)^2\|)
\]

\[
= E(\|\sum 2\eta_i (x_i - r_i) + \eta_i^2\|)
\]

\[
\geq \sum 2(x_i - r_i)E(\eta_i) + \sum E(\eta_i^2)
\]

\[
= N \cdot E(\eta_i^2)
\]

hence this representation clearly suffers from the curse of dimensionality. We obtain for the recursive SOM

\[
(\star) = E(\|\sum (\exp(-x_i) - r_i)^2 - \sum (\exp(-x_i - \eta_i) - r_i)^2\|)
\]

\[
= E(\|\sum \exp(-2x_i)(1 - \exp(-2\eta_i)) - 2\exp(-x_i)r_i(1 - \exp(-\eta_i))\|)
\]

\[
\leq \sum \exp(-2x_i)\|1 - E(\exp(-2\eta_i))\| + 2\exp(-x_i)|r_i| \cdot \|1 - E(\exp(-\eta_i))\|.
\]

The term \( E(\exp(-\eta_i)) \) approaches 1 if the support of the noise is concentrated on 0. Usually, only few terms \( x_i \) are close to 0 whereas the majority will be large corresponding to a unique winner for each tree of the network. Hence the curse of dimensionality is softened through the factors \( \exp(-x_i) \), but can possibly still be observed for high dimensional formal representations.

For SOMSD with \( \text{rep} \) as the index of the winner, \( (\star) \) obviously only yields a nonzero term, if the noise \( \eta \) changes the winner. For data where the winner is distinct, i.e. its activation is considerably smaller than the remaining entries, this probability is small for noise with limited support or Gaussian noise. However, even if the winner changes, it can be expected that the new winner lies in a close region, since the neurons which are neighbored to the winner tend to have small activations as well. Hence this representation is robust to noise. Moreover, the accumulation of errors through noise in recursive computation steps is obviously avoided in this approach due to the discrete values in \( \mathcal{R} \).

A further point of discussion concerns the representational capability of \( \mathcal{R} \). Note that the number of neurons is determined so as to correspond to the number of different data points we would like to recognize, or alternatively the granularity of the representation which we would like to obtain. However, the computation of each neuron goes through
the bottleneck of $R$ or the image of $rep$, respectively, i.e. the activation of different trees is internally represented by values in $R$ computed with $rep$. If two activation profiles cannot be distinguished by such values then the corresponding trees cannot be discriminated by the network. Hence we require the following:

- If two trees should not be identified through a computation, then their respective representation via $rep$ in $R$ has to be different.

The cardinality of $R$ and the image of $rep$ is infinite for the recursive SOM. For SOMSD $rep$ only maps to the number of indexes, hence SOMSD can differentiate between all trees which have different winners in the recursive computation. One can explicitly lower bound the cardinality of the image of $rep$, assumed a network should recognize all trees with binary labels of height at most $T$. We consider here an elementary definition of the notation of recognition:

**Definition 7.4** A neuron $n_i$ in a network recognizes a tree $t$ iff $\bar{d}(t, n_i) > \bar{d}(t, n_j)$ for all $n_j \neq n_i$. A network recognizes a set of trees $D$, if for each tree $t \in D$ a different neuron $n(t)$ can be found which recognizes $t$.

Note that the choice of the symbol ‘$>$’ is arbitrary and should depend on the respective setting. We could substitute the symbol ‘$>$’ in the above definition by ‘$<$’. Then obviously SOMSD could recognize every finite set of trees. Moreover, a number of neurons which equals the number of all different subtrees of trees to be recognized would be sufficient. Obviously, trees with real-valued labels can only be distinguished if an infinite number of internal representations is available. However, for finite labels (e.g. a sufficiently fine-grained representation of real labels) SOMSD uses almost the minimum cardinality for the image of $rep$ which will do for this purpose. We here assume that labels come from a binary set $\{0, 1\}$.

**Proposition 7.5** Assume a network recognizes all trees with labels in $\{0, 1\}$ up to height $T$. Then the cardinality of the image of $rep$ is lower bounded by the number of trees with labels in $\{0, 1\}$ of height at most $T - 1$.

**Proof.** Denote the number of trees with elements in $\{0, 1\}$ of height at most $T - 1$ by $h$. Assume a given SOMSD can recognize all such trees. Assume the cardinality of the image of $rep$ is at most $h - 1$. Then two trees with labels in $\{0, 1\}$ of height at most $T - 1$ exist, say $t$ and $t'$ such that $r = rep(\bar{d}(t, n_1), \ldots, \bar{d}(t, n_N))$ and $r' = rep(\bar{d}(t', n_1), \ldots, \bar{d}(t', n_N))$ are identical. Hence the activity profile of $t$ and $t'$ cannot be distinguished in further recursive computations. In particular, a tree which contains $t$ as a proper subtree cannot be distinguished from the same tree where $t$ is substituted by $t'$.

Note that SOMSD does not need the indexes for winners for trees with maximum height, since their activity profile is not used in further recursive computations.

We can expect from the standard use of self-organizing networks that important information of an activity profile of a tree is the location of the winner. SOMSD uses precisely this information, whereas the recursive SOM has access to all activations including the
winner. The TKM obviously does not use this information and is therefore restricted concerning its capacity. Since the respective winners are usually spread among the network e.g. according to the different tree structures, the network must have access to the activation of all neurons in recursive steps. Stated informally, we require the following:

- \( \text{rep} \) should be a global function which has access to (and effectively uses) information of the activation of all neurons.

One could alternatively try to design networks such that the recursive similarity of one neuron only depends on the activity profile of neighbored neurons. In such a local setting, the information about the global winner may be lost. Hence it can be expected, that such a network has only limited capabilities. A first investigation on the notion of locality and globality for the recognition of a set of trees is introduced in [74].

Commonly, information in self-organizing networks is detected through the metric structure of the data. In our case, \( d_E \) and \( d_C \) will commonly be metrics or scalar products. They are used for local comparisons of labels or formal representations, respectively. The computation for tree structures emerges through the recursive processing using \( d_C \), \( d_R \), and \( \text{rep} \). \( \text{rep} \) should take this fact into account. I.e. it should map similar trees to similar formal representations such that further use of \( d_R \) is justified. One problem consists of course in the fact that we have no notion for the similarity of trees. We did not fix a metric on trees; rather it emerges from the recursive processing and recursive comparisons of labels and representations using \( d_C \) and \( d_R \). A further discussion on this topic, in the context of topology preservation, is introduced in [74]. We can expect that important information lies in the winners for a tree. Trees are considered as similar in all standard ways of using the trained SOM if they have the same or similar winners. Hence at least the metrical information concerning the winner should be preserved through the function \( \text{rep} \) for appropriate processing:

- \( \text{rep} \) should map two vectors \( x \) and \( x' \in \mathbb{R}^N \) for winners lying at similar positions to formal representations which are similar with respect to \( d_R \). If \( x \) and \( x' \) have winners at very different positions, their formal representations should be less similar.

## 7.7 Discussion

We have proposed a general framework for unsupervised processing of structured data based on the main idea of recursive processing of the given recursive structured data. For this purpose, the dynamics of supervised recurrent and recursive networks is directly transferred to the unsupervised framework. Many special approaches like TKM, Rec-SOM, SOMSD, and even recurrent and recursive networks are covered by the framework. A key issue of the dynamics is the notion of internal representations of context, which enables networks to store activation profiles of recursively processed substructures in a distributed manner in a finite and fixed dimensional vector space. This allows the comparison of structured data of arbitrary size. The general framework allows us to formulate
training mechanisms in a uniform manner. Hebbian learning with various topologies such as VQ and NG topology can be immediately formulated. It turns out that unlike the case of unsupervised vector processing, Hebbian learning is only an approximation of a gradient dynamics of appropriate cost functions. We have formulated the cost functions for the general framework and we derived precisely computing of the gradients. Hebbian learning disregards the contribution of substructures in all cases and it is thus much less costly than the precise approaches. Moreover, this formulation proposes how to transfer different learning paradigms such as generalized vector quantization and variations to the recursive case [77, 147]: learning vector quantization (LVQ) [99, 98] constitutes a self-organizing supervised training method to learn a prototype based clustering of data with Hebb-style learning rules. The approach [147] proposes a cost function for variants of LVQ and introduces so-called GLVQ. This method constitutes a very stable and intuitive learning method for which additional features like automatic metric adaptation have been developed [77]. Since all these methods rely on cost functions which only depend on the squared Euclidean distance of patterns from the weights of neurons, they can immediately be included in the above framework for structured data, and above formulas can be used for calculating the corresponding derivatives.

Starting from this general formulation, a uniform investigation of properties of recurrent self-organizing maps is possible. We have already demonstrated the possibility with respect to the investigation of training algorithms. Further directions of research can take general properties of rep and the internal representation of trees into consideration which would help to design general criteria of the respective functions and uniform possibilities to evaluate the approaches. We have shown first steps into this direction concerning the noise tolerance of various representations and the representation capability. This substantiates the intuitive emphasis we give to the internal representation: the concrete choice of the internal representation is a crucial point for the success of the approaches. Actually, several proposals which can be found in the literature differ with respect to this point. The study of the property of the rep function can allow us to uniformly compare concrete approaches based on theoretical considerations even prior to training.
Chapter 8

Discussion and Related Approaches

The Artificial Intelligence, and in particular the Machine Learning area, are characterized by the existence of a large number of different approaches, founded on alternative computational paradigms (logic, statistics, etc.). As a result, these approaches usually differ on the representation languages, the mapping mechanisms, and the learning strategies they use. The processing of structured domains is becoming increasingly important in the Machine Learning area: a set of known methods is studied aiming at extending the class of data that can be processed, traditionally restricted to a flat form. The area has been termed with different names as “Relational Data Mining” or “Structured Domain Learning”.

We can consider the different approaches as characterized by different types of hypothesis space searched with different biases. Each method is more or less suitable according the specific task at hand. In particular, the language and the biases can be more or less appropriate to describe the relationships among the data of the application. A comparison among the various methodologies is a natural demand, and it can be the basis for further developments of models for SD.

In this Chapter, after in the preceding chapters we discussed the analysis and the extension of the neural networks approach to realize a SD-Recursive processing system able to handle structured data, we present a brief evaluation of alternative approaches for the processing of structured domains in ML. Some of them are characterized by the same recursive framework assumed in this thesis (a recursive transduction system), while others, which have origins in different paradigms of Artificial Intelligence, are characterized by a completely different approach. Other alternative approaches, more related to the specific field of computational chemistry applications, are instead postponed to the PART III of the thesis.

This review is not intended to be exhaustive. Instead we try to highlight selected previous work and promising new branches of research we believe to be useful for dealing with some of the issues that SD processing addressed. This provides an alternative way to discuss the characteristics of the proposed recursive connectionist methodology in a wider frame. With the sake of a critical mutual comparison we will give particular emphasis to the respective advantages and limitations.

Above all, we aim at arguing the importance of further analysis in order to integrate
the characteristics of the various approaches. This chapter, that concludes the PART II of the thesis (Models), should be considered a framework to introduce future developments (of models) and future research line. The work proposed in this thesis is shown, for future works, as a basis to study new hybrid approaches.

The chapter is organized as follows. In Section 8.1 we present the “Syntactic approach”, which illustrates the historical perspective before the new developments for the SD processing in ML were available. It is followed by the introduction of the “Symbolic approach”, based on the high representational power of the Inductive Logic Programming (ILP) language, that is especially relevant for its natural predisposition to deal with relational instances. Finally, the other approaches belonging to areas closer to the neural computing area will be summarized: in spite of their relevance in the ML area they often allows to extend the approach to SD in a more restrictive way.

8.1 Syntactic Approach

One of the main and traditional approaches to pattern recognition, is the syntactic one. Originally it was the natural alternative to the traditional decision-theoretic (or statistical) approach when there was a demand for structure-handling capability. The description of this method allows us to have a look at an approach widely used for the modeling and the description of structural relationships in pattern classes before other methods, in the ML area, were extended to deal explicitly with structures.

Syntactic pattern recognition is based on concepts belonging to the formal language theory. Fundamental for this approach is the decomposition of patterns into set of sub-patterns and primitives. The description of a pattern is done in terms of formal-language syntax of pattern components. The type of relationships among sub-patterns distinguishes the types of data structure used for pattern representation, which are, essentially, strings, trees and graphs. Pattern recognition is performed using mainly matching or parsing techniques. In the former the recognition can be done matching the representation of a pattern with a set of reference patterns; the single pattern can be considered as a pattern class. In the second method a mathematical model of grammar formally generates the language. Each primitive corresponds to a symbol permissible on some grammar. A grammar $G_i$ is made by rules that allow the generation of sentences that correspond to a set of patterns belonging to a uniform group. In a classification problem the language $L(G_i)$ generated by the $G_i$ would consist of sentences representing patterns of the same class $i$. The syntactic pattern recognition process is performed in terms of parsing. Given a sentence representing the input pattern, the problem is one of deciding in which language $L(G_i)$ the input pattern represents a valid sentence, so to assign it the class $i$. The possibility to have invalid sentences or a sentence belonging to more than one language corresponds to the presence of a rejection class.

On that ground, in this area, categorization and translation of syntactic structures is

---

1Useful surveys of the basic idea in syntactic pattern recognition appear for example in [38, 25].
based on the string language recognition capabilities of automata.

If patterns are tree or web representations the only exception to the methodology used for strings being that the recognition process is carried out by considering tree or graph grammars. This natural extension renders the syntactic approach the first unrestricted method capable to deal with structures.

The learning from a set of patterns is introduced by the means of "grammatical inference" from a finite set of instances. The problem is clearly under-specified so conditions and constraints must be imposed to recover a suitable source grammar for the data set. For example, according to Occam’s razor principle, the “simplest” grammar, e.g. on the number of rewrite rules or their lengths, that explains the training data is adopted. Various techniques were developed to infer various, often specific and limited, classes of grammars [61].

In matching methods, a generalization allowing learning is obtained by using weighted distance measures for attributes. Apt measures can be defined for strings. Clearly, similarity measures that reflect the prior knowledge on a specific application can leads to successful model based on this approach.

The generality and flexibility of the syntactic approach is constrained, in comparison with others ML approaches, by some important limitations. The main one is the difficulty of handling pattern deformations caused by noise or errors. The application of the syntactic approach to real-world problems may be restricted due to this drawback. Also, the use of discrete symbols to represent pattern primitives, as inherited from formal language theory, results inconvenient in handling numerical representation of pattern characteristics. Finally, the pattern representation and the syntax rules for describing complicated pattern structures are often found to be so complex that there is a lack of efficiency in performing the pattern matching or the parsing process.

Various ML methodologies deal naturally with the above problems and were often found able to improve the generality, flexibility and the quality of the results of the syntactic approaches when applied to the same class of problems. An illustrative example is given by recursive neural networks used for natural language grammatical inference. This model has natural error-handling and numerical-handling capabilities and it has been demonstrated to be Turing equivalent [148]. Various applications have been developed for induction of simpler grammars [42] [59] [174] [136] (e.g. on learning Tomita languages) as for induction of large regular grammars [58].

Nevertheless some ideas of the syntactic approach, primarily the structure-handling capability, may be used both to improve the other ML approaches and/or to develop combined approaches. As example see the possibility to extract the grammar learnt by a recurrent network (e.g. [136]).

Finally we mention the connection between ML techniques for structures, like recursive neural networks, and tree automata as pioneeringly developed by [113] [62] e.g. for deterministic bottom-up tree automata.

The relevance of the syntactic approach gives us additional motivations to overcome its limitations by the study of ML approaches with comparable structure-handling capabilities.
8.2 Symbolic Approach

Traditional key to AI is the ability to manipulate symbols. In many cases it is useful to describe the language of the learned model using symbolic formalisms: the hypothesis functions can be expressed as a set of if-then rules.

Well known methods in ML can learn set of propositional rules (see for example [130]): For instance, decision trees algorithms can be used to learn a set of rules expressed by the paths from the root to the leaves of the decision tree. Genetic algorithms can encode each rule set as a bit string and uses genetic search operators to explore this hypothesis space.

In order to represent relationships between data items the language has to be extended to first-order logic (FOL). In the following, we focus on ILP systems that are models able to learn relations in the form of logic theories.

Useful review for ILP can be found in [115] [131] [135] [41] [130].

8.2.1 Inductive Logic Programming

Inductive Logic Programming (ILP) [131] is the relatively young branch area of ML which deals with the induction of hypothesized predicate definitions from examples and background knowledge. Logic program language is used as a single representation language for examples, background knowledge and hypotheses.

The ILP approach allows the use of a more expressive descriptive language than the one used in propositional learning, involving first-order rules that contain variables. The representational power of the first order logic versus the propositional logic, used by others traditional ML approaches, is the key feature of ILP. For this reason ILP can deal with problems in which the hypothesis that must be represented involves relational assertions that can be conveniently expressed using first-order representations. ILP systems have already been applied in academic settings with some success to a range of difficult problems. These include protein structure prediction, drug structure-activity prediction, finite element mesh design, satellite fault diagnosis, circuit design, automobile traffic-flow analysis, and intelligent software agents for the Internet, lexical analysis. Some of the results have even been acknowledged by experts as having identified new “knowledge”.

Moreover the fact that knowledge is represented as rules over certain predicates makes the results of learning easier to be interpret by human than almost any of the other approaches in ML. This, along with the flexibility to include background knowledge, which can even include other propositional algorithms, allows a form of data analysis and decision-support that is, in principle, unmatched by first-generation methods.

Most important from our point of view is that the observations are expressed in the same language as the explanations, i.e. first-order logic. Therefore we have a natural way to represent and to handle structured information by a relational language. The hypothesis space of ILP models supports, at different levels of detail, the relationships between data.

Let us give an important example in the area of chemical applications. The SD is composed by chemical graphs. Each compound can be represented by relevant properties
and its structure as made in [107] (see also [41]). Basically, in such approaches, the structure is described at atom and bond level: the relations atom and element describe the atoms and the relations bond and connection describe the bond between atoms and bond-type the type of the bond. Some groups of atoms, e.g. benzene rings, are not explicitly decomposed in atoms and bonds. The fragment C-Cl of a given compound C-Cl₃-H, may be described as in the following:

atom('compound','compound_1').
element('compound_1',cl).
atom('compound','compound_2').
element('compound_2',c).
...
bond('compound','compound_1_2').
connected('compound_1','compound_2','compound_1_2').
connected('compound_2','compound_1','compound_1_2').
bond-type('compound_1_2',-).

The characteristic of the ILP methodology expressed so far should motivate our interest in the area. The study of comparison and integration with the neural networks approach deserves further (future) research.

**The method**  Briefly, the representational language used in ILP is composed by clauses. A clause is any disjunction of literals whose variables are universally quantified, and literals \( L_i \) are predicates of first-order logic. Theory is a finite set of clauses. The goal of ILP is to search an appropriate theory that fits the given examples.

We will consider learning first-order Horn theories, i.e. learning a disjunctive set of rules. This can be viewed as a method to automatically infer a PROLOG program from examples. Formally a Horn clause is a clause of the form \( H \lor \neg L_1 \lor \ldots \lor \neg L_n \) which is equivalent to the following, using a rule notation: IF \( (L_1 \land \ldots \land L_n) \) THEN \( H \). The literal \( H \) corresponds to the target concept.

These rules can be used as mapping mechanism in order to classify new data instances and/or used to improve the analysis of the background knowledge on the problem.

**Introduction to ILP algorithms**  In order to present the main aspects of ILP algorithms we focus on concept learning, i.e. binary classification. The target concept to be learned distinguishes positive and negative examples by the values 1 and 0 respectively (belonging or not to the category or concept).

The structure of the hypothesis space is viewed according to the more-general-than relationship: given hypotheses \( h_g \) and \( h_s \), i.e. boolean-valued functions, \( h_g \) is more general than or equal to \( h_s \) (or \( h_s \) is more specific than or equal to \( h_g \)) if and only if any instance that satisfies \( h_s \) (s.t. \( h_s(x) = 1 \)) also satisfies \( h_g \). Strictly more-general-than relationships is obtained when \( h_s \) is not equal to \( h_g \).
Learning is viewed as an ordered search in the hypothesis space organized on the more-general-than relationship and according, roughly, to two main strategies: the sequential covering and the inverse resolution algorithms.

The sequential covering strategy, thanks to its simplicity, allows to present in the following the core of the ILP methodology, according to our comparison aim. It extends the approaches of traditional algorithms for propositional learning, as CN2 [33], to deal with first-order logic. Some analogies are also possible with the decision-tree learning.

The main structure of the sequential covering algorithm, shared by many implementations, is quite simple for concept learning (binary classification task) and shortly summarized in the following sketch:

- Let \( P \) the set of examples labeled as positive
- While \( P \) is not empty do
  - Generate a new rule \( R \) that covers a large number of positive examples
  - Add \( R \) to the list of learned rules
  - Remove positive examples covered by \( R \) from \( P \)
- Return the set of rules

A disjunctive set of rules is learnt adding one rule at time; their composition covers the full set of positive examples. Because this algorithm performs a greedy search, without backtracking, it is not guaranteed to find the best or the smallest set of rules that covers the training examples. The strategies to learn a single rule carry out a heuristic beam search through the hypothesis space using general-to-specific (e.g. FOIL [139]) or specific-to-general scheme (e.g. GOLEM [133]). The selection of literals that will be added is guided by an information gain heuristic. More in general different evaluation functions, based on statistical measures, can be used to estimate the utility of adding new literals to the rule.

For example FOIL (Quinlan, 1990) [139] is a sequential covering algorithm, in which the language for clauses is PROLOG without functions. The search for a single rule is of the type general-to-specific (starts with the most general rule which asserts that the target concept is ever true), adding a single new literal to the rule preconditions (body of the clause) in order to obtain a more specific rule, with beam of width one. The strategies to generate the new rules is of the “generate and test” type, i.e. the search of successor hypothesis are generated on the basis of the syntax of the hypothesis representation and then tested on the training data to evaluate its performance.

The inverse resolution strategy uses the inverse of the deductive resolution operator for deductive reasoning to generate new candidate hypotheses. An example of this strategy is given by CIGOL algorithm [132]. It uses the inverse resolution to construct clauses and an example-driven scheme so that individual training examples constrain the generation of hypothesis. The specific hypothesis is corrected to improve its performance.

Interesting applications were developed for instance using GOLEM and PROLOG algorithms to deal with structure-activity relationships (QSAR) problems [97] [156].
8.2.2 Discussion and Comparison: Advantages and Problems

We can summarize some aspects of the ILP with the aim of comparison with other approaches:

- natural processing of symbolic and structured data;
- flexibility to include background knowledge: it allows the expert involvement in human background knowledge refinement;
- comprehensibility of the hypotheses: the results of learning are explicit rules. ILP addresses directly the important aim of knowledge discovery associated to the modeling one. This allows easy and further use of the discovered knowledge.

Current limitations are briefly discussed, with respect to the neural computing approach, in the following:

**Efficiency:** in ILP the search of possible hypotheses is performed in a large expressive space of the hypothesis. Even though and clever branch-and-bound or greedy search strategies are employed, the number of hypotheses generated and evaluated by these approaches may still be huge. Moreover, because of the lack of any type of compression of the information, the efficiency is related to the complexity of the input space, e.g. any single hypothesis is verified against all the examples. It is worth to note that concept learning problems are known to be NP-hard even in a propositional setting [86]. So, learning algorithms in ILP must cope with inherently computational complex problems. All these problems are exacerbated due to the combinatorial growth of relations expressible by a structured input pattern.

**Selection of language bias:** the expressiveness of the first-order language imposes to restrict the set of clauses or theories that are permitted. Constraints to the search for theories are imposed to deal with an inherently exponential search problem. An a priori accurate selection is difficult as the solution is application-dependent. Examples of restrictions that may be put on the language are: to restrict to function-free clauses, to restrict to non-recursive clauses, to put an upper bound on the number of clauses in the theory, to put an upper bound on the depth, or on the number of literals or variables in a clause, etc. Some methods have been studied to formally specify the language constrains as a declarative bias. However, the evaluation of the representation capabilities changing the bias is not easy. In neural networks, the complexity (and expressivity) of the model is directly related to the number of units of the model (the VC-dimension of the models is proportional to the number of network units). So, neural networks allow changing the complexity of the model varying the number of units while sharing the same framework for both the model and the learning algorithms. The same relationships have no correspondence in ILP.
Type of knowledge extracted: the rules do not necessarily cover all the relationships expressed in the instance set by the source of data. Generally, the learning algorithms terminate as soon as the positive examples are correctly classified. No smoothed information is expressed in the solution to obtain an interpolation capability.

Numerical Data: Handling numbers, as reals, and arithmetical computations is difficult, at least possible but rather inefficient. This deficiency is implicitly inherited by the use of a plain logic language. Especially, most of the methods are usually not suited to deal with the case of simultaneous occurrence of both categorical and numerical entities. This contrasts with the numerical representational characteristic of neural networks that naturally copes with general non-linear numerical-valued function approximation.

Noise Tolerance: Generally symbolic methods are not suitable to deal with incomplete data and uncertainty. This fact may limit the application to real-world tasks which require to deal with noisy and incomplete data. Again, in contrast, the neural networks approaches show good properties of tolerance to the lack of data integrity.

Specifically, for instance, ILP is not directly suited for regression problems. So far only few ILP algorithms can predict numerical values. This kind of learning problem is called relational regression in [40]. For instances, some approaches were proposed by Karalic (FORS) [95] and by Kramer [104, 41] but the problem is still open. Kramer for example proposed a structural regression tree algorithm that integrates the statistical method of regression trees into ILP. The algorithm constructs a decision tree. The node of tree contains conjunction of literals and the leaf nodes have a numerical value. In FORS (First Order Regression System) the technique is to break up the domain region of an input-output relation by the use of an induced set of logic formula and to approximate the input-output relation by local linear mappings, i.e. introducing simple linear equations to compute the output attribute $y$ in the clauses. However, such algorithms assign only constants or local linear equations to the leaves of the regression tree leading to a limited way to learn continuous target spaces.

The area of relational regression is another interesting example of the great deal of work on processing of relational data with flexible tools. Another reason of interest is that such field shares our interest for QSAR applications.

The proposed counterpart of ILP approach to the processing of SD, i.e. the recursive neural networks, inherited the characteristics of neural computing approach to naturally cope with some of the previous issues (see also Section 2.2). Besides characteristics already discussed above, we would focus on arguments for SD learning. In particular, the shape assumed by the recursive neural networks space of hypothesis for SD learning is continuous. Hence, using an error differentiable function, we can use simple gradient based techniques to perform parameters optimization on the basis of the training data.
Concerning the capability to deal with SD, as previously seen, ILP seems to address with single relationships of the SD in a natural manner. The final rules can contain even single relationship provided by the examples. On the other hand neural networks generate a smoothness global encoding of structures: the recursive encoding allows for processing local and topological information with a homogeneous overall view. The internal distributed code of the input structures developed by the model is adapted to the task at hand (see Section 5.2).

The major issue under investigation in ILP is often the trade-off between complexity of the algorithms and expressiveness of the representation languages. The recursive neural networks approach offers a suitable balancing between the two for the processing of SD. In fact, Recursive neural networks allow to deal with SD retaining the efficiency of the training techniques developed for propositional learners.

The main drawbacks of neural networks, in this context, are due to the use of a propositional language (that can limit the expressiveness) and to the black-box problem, i.e. the difficulties to make explicit and easily interpretable the knowledge learned by the neural network model.

**Crossing the Boundary**

We would mention the emerging challenge in the ML community to cross the boundaries between relational and attribute-value learning. Some topics are the propositionalization of the relational formalism, the multi-instance learning and the case-based reasoning.

Among such approaches, the propositionalization approach is an example of the attempt made in the area of ILP to reduce the burden to deal with complex hypothesis spaces, to make good and efficient use of background knowledge, and to improve the performances of the learning systems. This is also one of the main effort made for studying the boundaries between relational and attribute-value learning.

In fact, a way of combining symbolic processing exploiting the wide range of existing propositional learner has been considered in the area of Relational Data Mining [41] to transform a relational representation into a propositional representation (i.e. a flat representation). This type of transformation is called *propositionalization* [105]. It involves the construction of structural features from data.

A form of propositionalization is already presented in Section 2.3: the propositionalization scheme can be manually devised by the expert in application, providing the distinguished features that can properly represent the domain.

The approach mentioned here refers to an automatic generation of features by the propositionalization system that can be applied to different problems. The process can involve the construction of structural features from the relational background knowledge. Basically, the methodology entails the use of an ILP system (or DM techniques, e.g. searching frequent itemset) to generate features (propositionalization phase). The aim is to generate a representation of the data that does not hamper the performance of the learner. Then, a regression ML model is used to predict output values (mapping and learning phase).
The method can be grouped as methods that completely decouple the generation of features from the mapping phase (model construction) or grouped as methods that allow the learner to partially influence the preprocessing phase. Various researchers in Relational Data Mining and ILP area have proposed ways to derive propositional features from relational problems and then they have used these features successfully in attribute-value learners. A review of this line of research can be found in [106].

**Final Remarks on Relating the Approaches: a Language Issue** We can consider a different way to compare the two approaches, ILP and RNNs, raising the following issue: the relationship between the language underpinning the neural computing approaches and the language used in ILP.

There is a long research line devoted to study the neural networks as symbolic inference systems. A survey attempting a classification of connectionist inference systems according to their capability to implement variable binding is in [24]. However, connectionist learners are mostly effective in the case of simple propositional languages. Actually, the neural networks presented throughout the thesis are propositional learners.

Central to our purpose of SD processing is the function $\tau_E$ that allows us to perform the recursive encoding of the structured data. Clearly, the function $\tau_E$ can in principle be expressed in the logic programming language. Nevertheless some measures are needed to treat the numerical information. This argument supports the following assertion: the hypothesis expressed by the recursive neural networks is a specific type of hypothesis among the set of hypotheses that a logical language can express.

However, the class of recursive hypotheses is by itself powerful, quite compact and elegant. In FOL terms it allow us to realize a set of “parametric recursive clauses”. Hence, it is possible to realize Parametric and Adaptive SD-Recursive processing systems, and we exploited this possibility by recursive neural networks. The recursive neural networks training algorithms allow the tuning of parameters to the training data.

More important, the universal approximation theorem ([73], see Section 5.1) clearly shows the powerful of such approach. A major limitation is due to the propositional language underlying the neural networks: neural computing models are constrained to exclude variables. However, in practice, logical expressions, i.e. terms in FOL, can be easily represented as tree/DOAGs: for example the logical term $f(f(x, y), g(z))$ can be represented as a tree (in the p-rep form seen in Section 3.1) where vertexes are labeled by the function symbols (note that the constants, considered as function of arity 0, are leaves of the tree). Hence, instances of logical terms can be processed by RNN.

The main message is that although the recursive neural networks use propositional knowledge (without variables) they are not limited to use a flat attribute-value representation to handle the input domains.

A further interesting argument is the following: the fact that logic language can express recursive hypothesis does not assure that the ILP learning models can acquire rule expressed in such form. Besides the discussed problem in dealing with continuous num-
bers, the ILP models are by need constrained in the set of clauses or theories that are permitted: language bias often entails the restriction to non-recursive clauses.

Such topics deserve further research and some emerging approaches, for instance the propositionalization methods, clearly show the tendency to study hybrid solutions.

Concerning these proposed topics, we believe that even a simple comparison based on language issues, aimed at expressing the two approaches in their reciprocal language, can be useful. In fact, besides the dissemination merits, such approach can be used as a framework to study the possibility for bridging between the two apparently distant symbolic and connectionist worlds.

8.3 Propositional Approaches

The area of ML is very large. We propose in the following a short list of methods that can be further investigated in the framework of SD processing. Some of them deserve our specific interest for future developments.

In contrast to the approach based on first-order rules (Section 8.2), the methods presented in the following are classified as propositional approaches.

Probabilistic approaches Probabilistic approaches involve Bayesian learning algorithms [130] and probabilistic graphical models [93], e.g. Boltzmann machine, hidden Markov models (HMM). HMM in particular is regarded as one of the most significant approaches for probabilistic sequence modeling with successful applications, for instance in speech recognition and computational biology [5]. The use of HMM for SD processing is introduced in [52]. The extension to bi-directional models for sequence processing (see Chapter 6) is in [6, 7].

The advantages of models such HMMs reside in their capabilities to model stochastic dependences among variables with a solid statistical basis. They allows to reason on the probability distribution of the variables.

In general, besides the specific models, Bayesian reasoning provides a very general probabilistic framework that has been exploited to unify the inference approaches and that can be extended to new models.

Practical difficulties concerning these approaches are:

- requirement of initial knowledge of many probabilities, or their empirically estimation;
- significant computational cost of the inference process, often linear in the number of candidate hypotheses.

It should be noted that HMMs were extended to deal with SD (performing probabilistic transduction) [52]. This approach is based on a recursive processing; hence, it can be described under the same general recursive formalism presented in Chapter 3. For instance, the unfolding process generates a Bayesian network with shared conditional
probability tables. Moreover, a proper constrained (re)parametrization of the model allows us to use the simple and relatively efficient neural networks techniques for learning (see [8] for a first example of such approach).

Some works extend methods for learning in Bayesian networks to handle relational representations [56].

**Distance Based Methods** A very popular approach to ML, instance based learning (lazy learners) [130], can be studied to address the processing of SD. For example we can mention that the use of editing distance between graphs has been considered in $k$-nearest neighbor algorithms (see also Section 8.1 and Section 7.2). Case-based reasoning methods use more complex, symbolic representation for instances, e.g. CADET [157].

The problem is still to define a general relational distance measure (useful in applications) and to develop learning models based on this distance that can be successful in applications.

An interesting approach has been developed in [35]. The authors introduced a method, implemented in SUBDUE system, for discovering (repetitive) substructures in structural databases using the minimum description length principle.

**Support Vector Machines** Support vector machines (SVMs) [164] [26], which are able to deal with high-dimensional feature spaces, are emerging as a major technique within the ML area. Initially this method has been studied in the framework of concept learning. SVMs are based on the construction of the optimal separating hyperplane for patterns. For linear machine training, the optimal separating hyperplane is the one with the largest minimal margin, where the margin is the distance between the hyperplane and the closest positive or negative examples. A criterion for choosing the optimal hyperplane is included in the cost function and the problem becomes equivalent to solve a linearly constrained quadratic programming problem. The support vectors consist of a small subset of the training data extracted by the algorithm; they are the closest points to the decision boundary and they represent the critical elements of the training set for learning. The SVM is an approximate implementation of structural risk minimization induction principle, minimizing both the $R_{emp}$ and the bound on risk $R$ seen in Section 2.1. Accordingly, the SVM can provide very good generalization performance. The technique was generalized to non-linear separating surfaces and successively to real-valued functions. In fact, the implicit projection to high-dimensional feature space, performed through kernel functions obeying to specific condition (Mercer condition, [164]), permits to solve linear problems in the feature space corresponding to solution of non-linear problem in the input space. Hence, the main idea to extend the SVM applicability is in the “kernel trick” ([150]), which permits the computation of inner products in high-dimensional feature spaces. The definition of a kernel function corresponds to the definition of a similarity measure on data.

First SVM models worked only for flat domains. However, recently there is an emerging effort to extend the model for dealing with SD (e.g. [55, 80, 102]). In fact, through
the use of appropriate kernel (the “kernel trick”) the original data can be mapped in a flat feature space that the SVM can subsequently process, i.e. the appropriate definition of the kernel can allow to embed the analyzed data types in linear spaces. The choice of the appropriate kernels is a critical aspect, strongly related to the problem at hand and to the knowledge available to the kernel designer (the problem is related to the problem of a proper “a priori” preprocessing of the SD seen in Section 2.3). Since the kernel expresses the prior knowledge that is available about the task, its choice should be solved on a per case basis. Nevertheless, for some problem this knowledge is sufficient to develop an ad hoc kernel that allows to exploit the generalization capability of the SVM. Examples have been devised in the text categorization and Natural Language Processing (NPL) areas using string and tree kernels ([166, 34], and see for example [150] for a short review).

A systematic comparison of the kernel approach to SD processing with other approaches, including the recursive one, is needed. For instance, we have considered an initial set of applications [123] to compare the recursive approach with tree kernels methods. In particular, we have considered methods characterized by the mapping of an input tree into feature values that account for the occurrence number of its subtrees, i.e. a representation that, in analogy to the bag-of-words representation used in text processing, can be considered as a bag-of-subtrees representation (e.g. [166, 34]). Beside the specific and preliminary results, along the lines of the current chapter, we would briefly empathize general characteristics of the two approaches.

The kernel-based methods retain all the advantages above mentioned for SVM. In particular, since the space exploited by the kernel methods may have very high dimensionality, the drawbacks depending on the expressivity of a low dimensional vectorial representation of SD (as discussed in Section 2.3) are reduced. However, this does not assure that the chosen kernel considers the proper features (e.g. the local and global characteristics of the structured data) with respect to the task at hand; moreover a too high sparse representation can make the learning task very difficult. As above discussed, this is related to the critical aspect of the kernel design: the mapping performed by the kernel correspond to the a priori definition of an encoding function for the SD.

On the other hand, an adaptive SD-Recursive processing system, and in particular the RNN, maps the structured data in low-dimensional space through an adaptive encoding function (the function $\tau_E$). The code space of RNN can be seen as a “focused feature space” developed by the model on the basis of the training set. Hence, the similarity measure on data has not be chosen prior to learning, thus leading to a more general approach able to automatically encode the structural information depending on the computational problem at hand.

Finally, to show the potentiality of the future research in this field, we mention some interesting developments that allow the combination of kernel methods with other techniques, overcoming some limitations of the kernel approach. In the work presented in [91, 90] the authors proposed a construction, called the Fisher kernel, to combine kernel methods and probabilistic generative models, e.g HMMs. The method allows us to combine the features of HMM models, including the handling of variable length sequences, with the features of SVMs. Since Fisher kernel takes in account underlying probabilistic
models, it can provide a similarity measure which respects the process that generated the data. This is an interesting way to consider adaptive processing of SD by kernel machines.

Because of all these characteristics, the extension of SVM approach to SD and the study of its integration with the recursive approach seem very promising and they will be an interesting source of our future works.
Part III

Applications
“A relation exists between the physiological action of substance and its chemical composition and constitution, understanding by the latter term the mutual relations of the atoms in the structure.”
Crum Brown, 1868

In this part we present a novel approach to a fundamental problem in computational chemistry. For various reasons Chemistry and Biochemistry seem to be the right domain to test the computational capabilities of the proposed models (in the class of SD-Recursive processing systems) for processing of structures. First, such fields offer a set of real-world problems with strong humanitarian, scientific and industrial impulses where prediction models can be a useful tool to help the rational design of new molecules, e.g. new materials and new drugs. Then, the domain is organized in set of structured data, molecules, which are naturally represented as labeled graphs using vertexes for atoms, or group of atoms, and edges for bonds between atoms. The considered task is the modeling of the correlation between such chemical structures and their physical, chemical or biological activity where the micro-mechanism of interaction between compounds or between active molecules and the bio-receptor that determined the properties of interest, are not known in advance or very difficult to devise. The research area is referred using the acronym QSPR Quantitative Structure-Property Relationship, studying physical properties, or QSAR Quantitative Structure-Activity Relationship studying biological or pharmacological activities. Moreover, the data set is usually gathered experimentally, and thus is partial, noisy, and incomplete. Finally, must be observed that information carried by the compound are a combination of local and global aspects, and that apparently small chemical modifications of a ligand can change dramatically its pharmacological/chemical properties. The information are also a combination of structural and numerical inputs. The targets values are expressed by real number leading to the use of regression models.

The viewed capabilities of the connectionist paradigm to deal with these issues make the recursive neural networks a suitable candidate tool to perform the prediction in QSPR and QSAR tasks.

It is important to note that these applications are characterized both by a complex source of data, because the relationship to be approximated represent a set of biochemicals interactions, and often by a scarce amount of available data. The reason of using few data is related to limitations of traditional approaches, which in general are able to work only for homogeneous class of compounds, or, for innovative research areas, to the scarcity of experimental data for the activity under analysis. The importance of a good accuracy of the model, rather than his efficiency, is therefore a strong feature of QSAR applications.

The aim of the following set of applications, which should be considered demonstrative, is to show the advantages of the adaptive recursive methods in QSPR/QSAR analysis. The ultimate goal is to show how some real-world applications could have strong benefits using suitable ML approaches and that such problem can be a source of original ideas for new methods and models.
Structure of the Part  This Part is organized as follows:

In Chapter 9 we start introducing the basics of the current approach and we present an overview of the proposed applications. Then we introduce traditional approaches details for QSPR and QSAR analysis. The main aim is to propose a general framework for the area, unifying the traditional and the new approach based on recursive neural networks, as developed in Section 9.3. The basic concepts of recursive neural model are reviewed, in a more intuitive form, to collocate them in the QSPR/QSAR perspective allowing the discussion of the approaches characteristics in an uniform frame, and to make as much as possible independent the PART III. The Chapter 9 is concluded by a discussion on the general issues, that are shared in the various applications, for the representation of the compounds.

In Chapter 10 we present two QSPR applications. In the first one we face the prediction of boiling points of alkanes, showing the first attempt made to compare the model with previous approaches at the state of the art in the area. In the second one we present the result of the QSPR analysis aimed to predict thermodynamical properties of a set of compound selected in collaboration with the Chemical department of the University of Pisa.

In Chapter 11 we present examples of QSAR applications developed in collaboration with the Chemical Pharmaceutical department of the University of Pisa. The first one concern the analysis of benzodiazepines (a group of 1,4-benzodiazepin-2-ones characterized by a strong pharmacological interest) with the aim to compare the new approach with the traditional “Hansch” QSAR analysis. This well-know problem, characterized by the presence of prior knowledge on the domain, allows us to investigate some qualitative aspects. The internal representation developed by the recursive model trained for the task is analyzed considering the characteristics of the compound affecting the activity. Moreover, we use this data set to present result to evaluate the efficacy of the learning strategy used for the model. In the last experiment, we propose an application of our model as prediction tool for a new class of molecules: the QSAR analysis is completed by the prediction step for a theoretical library of compounds allowing to perform the first test of the model in the drug discovery process.

Finally, we introduce the characteristics that have lead to the development of an integrated software tool aimed at supporting the development of applications of RNN in SD.
Chapter 9

A New Approach to QSPR/QSAR Analysis

9.1 Introduction (Overview)

The possibility of relating some significant aspects of molecular structures to any particular behavior of a selected class of chemical compounds offers a big challenge in many fields of research, such as Chemistry, Biochemistry, Pharmaceutical Chemistry, etc. The assessment of such relationships represents the starting point for the prediction of required properties of new molecules. For instance, the ability of a model to predict specific properties of molecules allows the researchers to rationally design new compounds optimizing the requirement of both human and financial resources. For this reason the achievement of good predictive models constitutes a big task for both the basic and the applied research.

Many mathematical models were developed in the past years with the aim of analyzing relationships between molecular structures and target properties such as chemical reactivity or biological activity. The earliest methods all imply a non-direct correlation of the molecular structure to the target property. In these models some physico-chemical properties were used as molecular descriptors. They should be better classified as property-property or property-activity relationships models. The major problem in correlating some molecular properties (reflecting different structural aspects of molecules) to other kinds of properties (typically chemical reactivity or biological activity) is represented by the need to find a set of complete and relevant molecular descriptors.

The problem of identifying such proper descriptors, which initially had led to the use of physico-chemical properties [79, 78, 54], subsequently has been faced by the use of a wide class of numerical descriptors, more specifically oriented to the representation of molecular geometry/shape and atom connectivities (topological indexes) [72, 144, 119, 11]. Although these last methods use chemical graphs as versatile vehicles for representing structural information, the chemical graphs need to be encoded into the vectorial (or matricial) form required by the technique used to solve the regression problem. Of

\footnote{An introduction to QSAR analysis explaining its relevance for rational drug design is in Section 2.3.3.}
course, this encoding process is going to remove some structural information which may be relevant. Moreover, the \textit{a priori} definition of the encoding process has other several drawbacks. For example, when the encoding is performed by using topological indexes, they need to be properly designed by an expert through a very expensive \textit{trial and error approach}. Thus this approach needs an expert, which may not be available, or may be very expensive, or even may be misleading if the expert knowledge is not correct. Finally, changing the class of chemical compounds under study, or the computational task, will of course mean that all the above steps must be performed from scratch. More general vectorial representation of graphs, with uniqueness properties, may be very difficult to map on the target values.

A completely different approach is possible facing directly the processing of structured domain in the machine learning systems. While algorithms that manipulate symbolic information are capable of dealing with highly structured data, they very often are not able to deal with noisy and incomplete data. Moreover, they are usually not suited to deal with domains where both categorical (symbols) and numerical entities coexist and have the same relevance for the solution of the problem.

Neural networks are universally recognized as tools suited for dealing with noisy and incomplete data, especially in contexts where numerical variables play a relevant role in the solution of the problem. In addition to this capability, when used for prediction tasks, they do not need a formal specification of the problem, just requiring a set of examples showing samples of the function to be learned. Unfortunately, neural networks are mostly regarded as learning models for domains in which instances are organized into \textit{static} data structures, like records or fixed-size arrays, and thus they do not seem suited to deal with structured domains. Recurrent neural networks, that generalize feedforward networks to sequences (a particular case of dynamically structured data) are perhaps the best known exception.

Throughout the previous chapters following the line developed in [155] and [52], we have studied a generalization of recurrent neural networks for processing sequences to the case of directed graphs (SD). While the details of recursive neural networks (RNN) are presented in \textsc{Part II}, the basic idea behind this generalization, i.e. the concept of \textit{unfolding}, will be recalled and applied to the chemical application setting.

The capability of RNN models analyzed in the previous chapters, are particularly useful when dealing with prediction tasks in Chemistry, where data are usually gathered experimentally and the compounds can naturally be represented as labeled graphs. Basically, each node of the graph is an atom or a group of atoms, while edges represent bonds between atoms. So neural networks for processing of structures seem to have the computational capabilities to deal with chemical domains. The prediction model can face one fundamental problem in Chemistry: the prediction of the physical-chemical properties, chemical reactivity or biological activity of molecules, leading to \textit{Quantitative Structure-Property Relationship} (QSPR), or \textit{Quantitative Structure-Activity Relationship} (QSAR) studies. Recursive neural networks face this problem by simultaneously learning both how to represent and how to classify structured patterns. The specificity of the proposed approach stems from the ability of these networks to automatically encode the structural
information depending on the computational problem at hand, i.e., the representation of the molecular structures is not defined a priori, but learned on the basis of the training set. This ability is proved in the following by the application of Recursive Cascade Correlation (Section 4.3) to various, and radically different QSPR/QSAR problems. In fact, we consider two QSPR problems, the prediction of the boiling point for a group of acyclic hydrocarbons (alkanes), the prediction of thermodynamical properties of the solvation process, and two QSAR problems, the prediction of the non-specific activity (affinity) towards the benzodiazepine/GABA_A receptor by a group of benzodiazepines (Bz), and the prediction of A1 adenosine receptor ligands affinity toward the receptor.

It must be stressed that the generality and flexibility of a structured representation, allows one to deal with heterogeneous compounds and heterogeneous problems using the same approaches. This advantage is not at the expense of predictive accuracy, in fact our results [20] [21] compare favorably versus the traditional QSAR treatment, for the analysis of benzodiazepines, based on equations [65]. It is also competitive with results on QSPR problems (such as, the prediction of the boiling point of alkanes) where the a priori analytical knowledge allows the use of suitable ‘ad hoc’ representations as input to standard neural networks [30].

Successive studies on the internal representation developed by the recursive neural networks (realized by a Cascade Correlation algorithm) applied to QSAR studies of benzodiazepines were conducted using principal component analysis [128]. This study allows us to deal with the issue of the relationship between the developed neural numerical codes and the qualitative aspects of the QSAR problem. The results show that the recursive neural network is able to discover relevant structural features just on the basis of the associations between the molecular morphology and the target property (affinity). In particular the characteristics of many substituents affecting the activity of benzodiazepines, already highlighted by previous QSAR studies, were correctly recognized by the model. This is a further step towards the assessment of the model as a new tool for the rational design of new molecules.

**Structure of the Chapter** The Section 9.2 allows to introduce details of the traditional approaches to QSPR/QSAR analysis with emphasis in methods based on the graph topology and on neural networks models. In Section 9.3 we propose a unifying schema for QSPR/QSAR problem basing on graph transduction and we “immerse” in such formulation the proposed approach based on recursive adaptive models. General considerations concerning the representation of molecules are drawn in Section 9.4.

### 9.2 Background: Approaches to QSPR/QSAR

The basic assumption of QSPR/QSAR is that there exists a direct relationship between the chemical structure of a compound and its physical properties or its biological activity with respect to a receptor. Moreover, it is assumed that this relationship can be quantified.
A typical way of representing the information about a chemical structure is to resort to chemical graphs, i.e., graphs where each node corresponds to an atom (or functional group of atoms) and each edge corresponds to a chemical bond. The main problem in traditional approaches to QSPR/QSAR analysis is to find a good numerical representation capable of retaining the chemical and topological information present in chemical graphs and to relate them to the target property or biological activity. The need for a numerical representation is due to the use of mathematical models, e.g., multi-linear regression, to quantify the relationship of specific structural features of a compound with the target value.

Different approaches to QSPR/QSAR can be distinguished according to the way these numerical representations are defined and/or devised. Basically, two different methods have been proposed for the definition of numerical descriptors:

- using known *physico-chemical parameters*[^3] [111], such as polarization, molar refractivity, hydrophobicity, etc.; these parameters basically measure physical and chemical properties of atoms or groups of atoms, such as substituents, giving a partial description of the molecular structure;

- using *topological indexes*, which code specific morphological properties of the molecular graph.

These descriptors are then related to the target property or biological activity by more traditional techniques, such as multiple linear regression, principal component analysis (PCA), partial least square (PLS) [112], or more recently developed techniques, such as genetic algorithms [142, 151], neural networks [27, 4] and support vector machine (e.g. [36, 88]). In the following, we briefly review topological indexes and neural networks techniques for QSPR/QSAR since they are directly related to the approach proposed in our work.

### 9.2.1 Topological Indexes

Topological indexes have been developed to represent structure features (graph invariants) on the basis of graph theory. To this end, molecular structure is expressed in numerical form suitable for manipulation in algebraic equations. In this approach, the focus of attention is on geometrical and topological features rather than chemical ones. For example, topological indexes typically convey information about the “shape” and “size” of chemical compounds.

The topological index based methodology for QSPR/QSAR is typically defined through a 3 step procedure:

[^2]: The group contribution methods will be detailed in Chapter 10.
[^3]: Historical remark: a systematic approach to the treatment of these relationships was mainly introduced by C. Hansch et al. in the 60s [79, 78] with the development of equations able to correlate the biological activity to physical and chemical properties of biologically active compounds.
1. the chemical compound is represented by using a simplified structure derived from the molecular skeleton by taking off both hydrogen atoms and the typology of bond, atoms and groups;

2. the above representation is encoded by numerical indexes that represent specific topological features;

3. one or more meaningful indexes are selected, according to the problem at hand, and used as input to regression models, such as linear regression.

Specifically, concerning step 2, the count of atoms, or bonds, or couple of adjacent bonds, are among the simplest descriptors. These descriptors were used since the very beginning to evaluate the qualitative relationship between structure and property. Unfortunately, they are not powerful enough to allow the discrimination of different molecular topologies. For example, they are not able to fully capture the branching structure of a given compound. The theory of topological indexes tries to remedy this representational gap by defining more complex and powerful descriptors able to retain specific additional structural information depending on the property to be predicted. Due to the different possibilities of relationship between topological characteristics and properties, a considerable number of topological indexes have been developed. Among these there are adjacency or distance matrix based indexes [144, 145, 11] and indexes based on information theory [119]. A systematic classification of topological indexes based on the encoded morphological characteristics can be found in [72].

As an example, here we discuss the derivation of the chi connectivity indexes family. The key concept in chi indexes is the decomposition of the molecular graph into fragments (subgraphs) of different size and complexity. A chi index is calculated as the weighted count of a given type of subgraph. The order and type of the index depend on the size and complexity of the selected subgraph, respectively. The order $m$ of the index is the number of graph edges in the considered subgraph, whereas the type $t$ refers to the particular arrangement of the edges (e.g., Path (P), Cluster (C), Path/Cluster(PC), and Ring (R); see Figure 9.1) in the subgraph.

In order to compute a chi index of a molecule, each vertex of the corresponding molecular structure needs to be represented by a numerical value, the so called delta value:
where $|S(i)|$ is the degree of the $i$th vertex. The *delta value* is then used to define the contribution of single subgraphs to the overall computation of the index.

For example, in the *chi* index of order zero ($^0\chi$) the selected subgraph is a single vertex. Zero degree denotes the absence of edges in fragments containing only single vertexes. By defining the contribution term $c$ for a given subgraph $s$ as

$$^0c_s = (\delta_s)^{-1/2},$$

the index can be computed as the sum of these terms for all subgraphs (vertexes) in the graph:

$$^0\chi = \sum^0c_s.$$  

In first order *chi* index ($^1\chi$), edges are taken in account as graph fragments. In analogy with Equation 9.2 and 9.3, $c$ and $\chi$ are defined as:

$$^1c_s = (\delta_i\delta_j)^{-1/2},$$

$$^1\chi = \sum^1c_s,$$

where an edge $s$ connects vertex $i$ with vertex $j$, and the sum is over the set of edges.

Eventually, indexes of higher order ($m > 1$) exploit more complex types ($t$) of graph fragments, such as paths and clusters:

$$^mc_s = (\prod^m(\delta_i)^{-1/2},$$

$$^m\chi_t = \sum^mc_s,$$

where the product is over all delta values in each singular fragment of order $m$, and the sum is over all fragments of order $m$ and type $t$.

Examples of computations of $^3\chi_p$ are shown in Figure 9.2. Note that, for each chemical structure, several distinct topological indexes can be computed. The choice of how many indexes and which specific one is computed depends both on the type of chemical compounds and on the computational task.

For different classes of molecules, the use of topological indexes in linear regression has been successful, showing that they are able to retain topological information which is directly correlated with the target property. In effect, the usefulness of the topological approaches for certain groups of problems, mostly for alkanes and Hydrocarbons, is beyond doubt. However, they cannot be recognized as general solution methods for many other QSPR/QSAR problems. For example, it remains under discussion both how to expand the definition of the indexes for families of compounds containing heteroatoms and multi-bond types, and a unique numerical characterization for each molecule in large data sets.

Examples of properties studied in this context are solubility, boiling point, density, molar volume, molecular refraction, partition coefficients and thermodynamic properties.
Figure 9.2: Examples of computations for $^{3}X_{p}$ topological indexes. There are three possible paths composed of three edges in each represented structure. For every vertex the delta values are reported and we calculate the $^{3}c_{3}$ value for each fragment. It can be noted that different structures lead to different indexes values.
Some QSAR applications have been developed for toxicity analysis, anesthetic potency, hallucinogenic activity, enzyme inhibition, flavor, odor and taste, antimicrobial and antibacterial activity, carcinogenicity. Some examples of the above applications are reported in [72].

### 9.2.2 Neural Networks

Recently, feedforward neural networks have been used in QSPR and QSAR, since they constitute a more powerful regression tool than multiple linear regression. In fact, the nonlinearity of neural networks can be exploited to better correlate topological indexes and physico-chemical parameters to target properties and activities [3, 30, 1, 39].

The molecule is typically represented by a vector of features (flat representation), which may be both topological indexes and physico-chemical parameters. As we will see in the following, a bit more sophisticated representations have also been proposed. The vectorial representation is then fed in input to a Multilayer Perceptron (MLP) for classification or prediction. Different neural networks approaches basically differ in the way the molecule is represented.

The most typical approach is the one where a feedforward neural network is trained on physico-chemical parameters or numerical topological indexes (see [27, 175] for reviews). This approach has been used for tasks where an high degree of nonlinearity is required for the development of the QSAR/QSPR analysis, as in the case of QSAR of a class of benzodiazepines [3, 137]. An example of a QSPR study which fits in this context has been the prediction of the boiling point and other properties of alkene [118], where besides using $\chi$ indexes, other ‘ad hoc’ topological indexes are defined in order to deal with double bonds which occur in the unsaturated hydrocarbons family and to distinguish enantiomers of alkenes.

Some authors tried to preserve more information on the molecular structure through a vectorial or matricial representation of the chemical graph. An approach of this type is used in [30], were the authors present a study on the prediction of the boiling points of alkanes. In this case, the input to the MLP is given by a vectorial code obtained by encoding the chemical graph with suppressed hydrogens through an “N-tuple” code (see Figure 9.3). Each component of the vectorial code, which in this case is of dimension 10 since only alkanes with up to 10 atoms are considered, represents the number of carbon bonds for each atom. The uniqueness of the code is guaranteed by keeping a lexicographic order.

This representation for alkanes is particularly efficient for the prediction of the boiling point since it is well known that the boiling point is strongly correlated with the number of carbon atoms and the branching of the molecular structure. Of course, while the obtained predictions are very good, the same representation could be useless for a different class of compounds. An example of matricial representation can be found in [43], where Elrod et al. used simplified and fixed size connectivity matrices ($5 \times 5$) to partially represent the structure of a substituent.
It must be noted that both vectorial and matricial representations have a fixed dimension and thus they cannot fully encode structures of larger size. Moreover, smaller structures need to be represented by inserting zeros into the void positions. The first problem could be solved by enlarging the size of the representation, however this would increase the number of free parameters of the MLP, thus decreasing the probability to obtain a good generalization capability, since the data sets are typically very small in size and usually it is not easy to extend them to compensate for the increased number of parameters. The latter problem leads to the underutilization of parameters.

Another problem of vectorial and matricial representations, however, must be recognized as the uniqueness of representation. In fact, in order to avoid that the same compound is represented in different ways, a procedure for the assignment of a unique enumeration for the vertexes of the chemical graph must be devised. In [43], the problem is partially solved by numbering the atoms (excluding hydrogens atoms) in a substituent according to a breadth first visit of the corresponding chemical graph starting from the point where the substituent is attached (see Figure 9.4). Any additional ambiguity is ignored. Whereas this approach seems to work for the family of compounds examined in this study, there is no guarantee to obtain a unique representation for a larger set of compounds. In order to obtain a unique representation for each compound, it would be necessary to devise a canonical enumeration procedure. Unfortunately, such procedure should solve the more general problem of isomorphism between graphs, which is known to be computable in polynomial time only for some classes of graphs.

Finally, a template based approach has been explored as well [114]. The basic idea of this approach is to have a neural network which mimics the chemical structure of the input compound. A common or given template in the family of compounds to be examined is
individuated a priori and then used as topology for the neural network. Specifically, each input unit in the network corresponds to an atom (or to a functional group of atoms) into a specific position of the molecular template and it allows to signal the presence (or absence) of a chemical entity in that position of the template. Moreover, the network is not fully connected, since connections are present in correspondence with bonds represented into the template (see Figure 9.5 for an example). Of course, because of the common template, this approach can be applied only for a set of very homogeneous compounds.

Finally, we can mention recent approaches based on support vector machines (e.g. [36, 88]). Although such model can show improved behavior with respect to the prediction performance, basically the approaches use a preprocessing phase to extract structural descriptor from molecules according to the methods presented in this section.

A proposal to deal with the problem moving from a “flat” representation of chemical data to a richer representation based on structured domain is introduced in the following.
9.3 A General Framework Toward the Novel Approach

In this section we present a unified framework, based on functional transductions, to summarize the traditional QSPR/QSAR approaches and to introduce the new one based on neural network for structures. In this formalism we can prove that recursive neural networks (RNNs) can be a suitable tool for the automatization of two fundamental parts of the QSAR analysis, i.e. the encoding and mapping phases. Moreover, reviewing the different approaches in a general perspective should help to analyze the advantages of the new approach.

Without loss of generality, for the sake of a simpler exposition and due to their relevance, we mainly focus the following explanations and examples on QSAR studies. However, though QSPR deals with general properties instead of activity, the following considerations are valid both for QSPR and QSAR analysis.

The aim of a QSAR study is to find an appropriate function $\mathcal{T}(\cdot)$ which, given a molecule structure, predicts its biological activity, i.e.:

$$\text{Activity} = \mathcal{T}(\text{Structure}). \quad (9.8)$$

More generally QSPR assumes that any molecular property, such as physical-chemical properties, can be related to the structure of the compounds, i.e.:

$$\text{Property} = \mathcal{T}(\text{Structure}). \quad (9.9)$$

The function $\mathcal{T} : \mathcal{I} \to \mathcal{O}$ is therefore a functional transduction from an input structured domain $\mathcal{I}$, constituted by molecular structures\(^4\), to an output domain $\mathcal{O}$, such as the real number set. In Equations 9.8 and 9.9 the term “structure” stresses the importance of the use of global information about molecular shape, atom connectivities and chemical functionalities as understood in the QSPR/QSAR studies.

The function $\mathcal{T}(\cdot)$ is a complex object which can be described as the sequential solution to two main problems: i) the representation problem, i.e., how to encode molecules through the extraction and selection of structural features; ii) the mapping problem, i.e., the regression task usually performed by linear or non-linear regression tools (e.g., equational modeling, and feedforward neural networks).

According to this view, $\mathcal{T}(\cdot)$ can be decomposed as follows

$$\mathcal{T}(\cdot) = g(f_{\mathcal{I}}(\cdot)) \quad (9.10)$$

where $f_{\mathcal{I}}(\cdot)$ is the feature representation (or feature extraction) function from the domain of the chemical structures to the descriptor space, while $g$ is the mapping function from the descriptors space to the biological activity space. This corresponds to the traditional QSPR/QSAR approaches, as summarized in Figure 9.6 for the QSAR, where chemical features are represented by a suitable set of numerical descriptors (function $f_{\mathcal{I}}$), which

\(^4\)Note that this is not yet a structured domain composed by mathematical graphs, as the domain $\mathcal{G}$ introduced in Chapter 3.
Figure 9.6: Outline of the traditional QSAR approach. Structural features of the molecule are represented through different numerical descriptors. The numerical descriptors can be obtained by using different approaches. Their number and type depend on the QSAR task at hand. The encoding process on the whole defines the $f_I$ function. A regression function ($g$) is then applied to the numerical descriptors to obtain the predicted biological activity.

are then used to predict the biological activity (function $g$). As seen in Section 9.2, the representational problem is faced by using different approaches such as the definition and selection of physico-chemical or geometrical and electronic properties, the calculation of topological indexes, or an explicit vector based representation of molecular connectivity (see the examples in Section 9.2). The question mark in the picture shown in Figure 9.6 stresses that the number and type of descriptors used to represent the chemical compound depend on the specific QSAR problem at hand. The exact number and type of descriptors used for a specific study are decided by an expert in the field.

In more detail, the encoding process requires the solution of two subtasks. The aim of the first one is to explicitly represent the relevant structural information carried by molecules, while the second one is to codify this structural information into a numerical representation. For example, when considering topological indexes, first of all a molecule is represented by the molecular graph skeleton, and then invariant properties of the molecular graph skeleton are used to define and compute a numerical formula. Thus, the function $f_I$ can be understood as the following composition

$$f_I() = \tau_E(\tau_R()) ,$$

(9.11)

where $\tau_R$ extracts a specific structural aspect from the molecule (i.e., the solution to the first subtask), and $\tau_E$ computes a numerical value from the structure returned by $\tau_R$ (i.e., the solution to the second subtask). Examples of $\tau_E$ are the connectivity indexes ($\chi$), or the hydrophobic, electronic, polar and steric properties.
We could sort the traditional approaches on the basis of the evolution toward the use of more direct representations of the molecular structures. Summarizing the methods seen in Section 9.2, we can mention models based on physico-chemical properties [3, 1, 137, 39], which may be obtained as combinations of fragment contributions, on topological indexes [118, 30], or matricial [43] graph representations, and finally a template-based approach [114]. This last model uses a neural network that partially mimics the chemical structures of the analyzed compounds by means of a common molecular template, statically defined for all the compounds.

However, in traditional QSPR/QSAR, both $\tau_R$ and $\tau_E$ are defined *a priori*, i.e., they do not depend on the regression task. Therefore they are designed through a very expensive trial and error approach in order to adapt them to the regression problem required by the QSPR/QSAR study. So, even if the chemical graph is clearly recognized as a flexible vehicle for the rich expression of chemical structural information, the problem of using it in a form amenable directly to QSPR/QSAR analysis is still open.

In this Thesis we propose a complete different approach. In fact, using the recursive neural networks *(PART II)* we realize the $\tau_E$ function through an adaptive mapping, thus allowing the automatic generation of numerical descriptors which are specific for the regression task to be solved.

In fact, the methods presented in PART II provide a SD-Recursive processing system able to implement a function graph transduction $\mathcal{T}_G : \mathcal{G} \rightarrow \mathcal{O}$. We can exploit such function in the QSPR/QSAR context considering the function $\tau_R : \mathcal{I} \rightarrow \mathcal{G}$ to formalize the representation of the “molecular structure” in the input domain $\mathcal{G}$. The functions $\mathcal{T}$ of Equations 9.8 and 9.9 can be obtained as the following composition:

$$\mathcal{T} = \mathcal{T}_G(\tau_R()).$$

(9.12)

The main aspect of such approach is its ability to take directly as input the graph generated by $\tau_R$ and to implement adaptively both $\tau_E$ and $\tau_R$ (that compose $\mathcal{T}_G$).

In order to exemplify the above concepts, in Figure 9.7, we show the outline of the proposed approach assuming that a given molecule is represented by $\tau_R$ as a labeled tree in $\mathcal{G}$. The definition of an appropriate function $\tau_R$ for the specific set of molecules studied in this thesis, based on simple rules according to the standard conventions used in chemistry, is discussed in Section 9.4 and in the following chapters. The goal of such rules is to obtain an unique structured representation of the chemical fragment as labeled ordered trees. For instances, the tree in Figure 9.7 is obtained representing the shared nucleus of the molecular group as the tree root and the atoms and bonds of the molecule substituents as vertexes and edges of the tree. This tree-structured representation is then processed by a recursive neural network. The output of the recursive neural network constitutes the regression output, while the internal representations of the recursive neural network (i.e., the output of the hidden units) constitute the neural implementation of the numerical descriptors returned by $\tau_E$. It must be stressed, at this point, that the recursive neural network does not need to take as input a fixed-size numerical vector for each input graph, as it happens with standard neural networks typically used in QSAR studies, because it is
Figure 9.7: The new QSAR scheme using recursive neural networks is shown: the molecule, after a structural coding phase driven by ad hoc rules ($\tau_R$), is directly processed by the recursive neural network through the adaptive encoding function $\tau_E$. The internal representation developed by the recursive neural network is then used by the regression model implemented by the output part of the neural network (function $g$) to produce the final prediction (activity).

We may observe that the main difference between the traditional QSAR scheme shown in Figure 9.6 and the proposed new scheme reported in Figure 9.7 is due to the automatic definition of the $\tau_E$ function obtained by training the recursive neural network over the regression task. This implies that no a priori selection and/or extraction of features or properties by an expert is needed in the new scheme for $\tau_E$.

To fully grasp the method underpinning recursive neural networks within the context outlined in Figure 9.7, it is crucial to understand how the encoding function, i.e., $\tau_E$, is computed for each input graph. For the sake of exposition, in the following we assume that $\tau_R$ returns labeled trees, where each label associated with each node of the tree is a symbol representing, for example, the atom type or a molecular group. Since $\tau_E$ will be realized by a recursive neural network, these symbols need to be represented as numerical vectors. For example, a bipolar localist representation can be used to code (and to distinguish among) the types of chemical objects. In a bipolar localist representation each component of the vector is assigned to one entity and it is equal to 1 if and only if the representation refers to that entity; otherwise it is set to -1; e.g., assuming that the fluorine atom (F) is associated with the i-th component and the chlorine atom (Cl) is associated with the j-th component, the fluorine atom is represented by the following vector $[-1, -1, ..., -1, 1, -1, ..., -1, -1]$, while the chlorine atom is represented by $[-1, -1, ..., -1, 1, -1, ..., -1, -1]$.  

---

5The following arguments are an intuitive explanation of the mathematical concepts presented in Section 3.2 adapted to the chemical domain.
Figure 9.8: The coding process: a code is progressively generated for each node by using the code already produced for its descendants. Nodes colored with different gray levels are used to denote the time when the code of each node is used as state information for the current node: e.g., the code for node 2 is generated by using the codes generated for nodes 3 and 4 (in addition to the numerical label attached to node 2).

\[[-1, -1, \ldots, -1, 1, -1, \ldots, -1, -1].\]

The computation of \(\tau_E\) is a progressive process which starts from the leaves of the input tree and terminates at the root of the tree, where a numerical code for the whole tree is generated. Specifically, this coding process starts at the leaf level by producing step by step a code for each visited leaf node and by storing these codes as state information for each corresponding leaf. Successively, the internal nodes are visited, from the frontier to the top of the tree. For each currently visited node its numerical label and the codes already computed for its children (stored in the state), are used to compute the code for the current node. Since this computation is performed in the same way for all the nodes in the tree, the generated codes are all constrained to be of the same size. Finally, the code computed for the root of the tree is used as the numerical code for the whole tree. Note that for leaf nodes the process starts with a null state because there is no previous information from descendants.

In Figure 9.8 we exemplify the above visit on an input tree where the labels are not explicitly represented. First the leaves (nodes 4, 5, 6 and 7) are visited and the corresponding codes are generated. Then node 3 is visited and a code for it is produced taking into account its label and the codes generated for its children, i.e., nodes 5, 6, and 7. Successively, a code is computed for node 2 using the codes computed for (the subtrees rooted in the) nodes 3 and 4, and the label of node 2. Finally, the root node 1 is visited and the code for it, corresponding to the code for the whole tree, is generated. The different gray levels used to fill in the tree nodes convey information about the time when the code of each node is used as state information for the current node.

Note that the way the encoding function acts on a specific tree, such as the tree in Figure 9.8, is specified in terms of how the encoding function acts on the subtrees of each node. In this sense the encoding is “recursive”.

Concerning the regression function \(g\), it takes as input the code generated by \(\tau_E\) for the root of each input tree and returns the desired value associated with the tree.
9.3.1 Recursive Neural Network Model in QSPR/QSAR

A synthesis of the functions and domains for QSPR/QSAR applications extended on the basis of the SD-Recursive processing system presented in Chapter 3, can be given in the following schema:

\[
\mathcal{T} \xrightarrow{\tau_B} \mathcal{G} \xrightarrow{\tau_E} \mathbb{R}^m \xrightarrow{g} \mathbb{R}
\]

(9.13)

where \( \mathcal{G} \) is the set of labeled direct ordered acyclic graphs (DOAGs) with supersource and bounded out-degree (\( K \)), e.g. labeled rooted \( K \)-ary trees. The descriptor (or code) space is chosen as \( \mathbb{R}^m \) while the output space, for our purpose, is defined as \( \mathcal{O} = \mathbb{R} \).

We choose a uniform and quite simple neural realization for each step of \( \tau_E \) through the definition of a recursive neural network model. In order to process each node the recursive neural network uses the information available at the current node: \( i) \) the numerical label attached to the node, \( ii) \) the numerical code for each subgraph of the node (state information).

As a result, if \( K \) is the maximum out-degree of DOAGs in \( \mathcal{G} \), the recursive neural network, for each step of \( \tau_E \), gets input from the space

\[
\mathbb{R}^m \times \underbrace{\mathbb{R}^m \times \cdots \times \mathbb{R}^m}_{k \text{ times}}
\]

and produces a code in \( \mathbb{R}^m \).

The details on neural network realizations are in Chapter 4. Here, we would like to review the unfolding concept and then better clarify the approach instancing it to the chemical domain. In particular we show how the encoding process is specialized for each input molecule.

The neural encoding process of an input graph can be represented graphically by replicating the same recursive neurons (through the input graph) and connecting these replicas according to the topology of the input graph. We obtain in this way the so called encoding network. Examples of encoding networks for \( m = 1 \) are shown in Figure 9.9. The examples involve two substituents (-CH₂ and -CH₂-CF₃) for the benzodiazepine class of molecules studied in this work. More complete examples are in Figure 9.10, based on the same substituents, where two neurons are involved (\( m = 2 \)) and a representation of the numerical vectors with \( n = 3 \) for the encoding of the symbol is reported. For the sake of simplicity, the labels shown here represent only the three different atoms involved in these examples (i.e., H is represented by \([1,1,1,1]\), C by \([-1,1,1,1]\), and F by \([-1,-1,1,1]\)).

The encoding network is a feedforward network that mimics the topology of the molecular graph. For each input graph a corresponding encoding network is built up. There is a correspondence between graph nodes and units of the encoding network; however, the template used to encode the molecular graph is not fixed \textit{a priori} as happens in the template-based approach used in [114]. Notice that the weight matrices are shared by different encoding networks (see Figure 9.9), since the same recursive neurons are used to
Figure 9.9: Examples of encoding networks (left side) for the chemical fragments -CH$_3$ and -CH$_2$-CF$_3$ with $m = 1$. The fragments are assumed to be represented by the chemical trees shown on the left side of the figure. Each encoding network is obtained by replicating (unfolding) the recursive neuron for each node in the chemical trees (as shown by the multiple occurrences of the weights). The black squares represent void pointers which are encoded as null vectors (in this case, the void pointer is equal to 0). The labels, here represented as symbols, are supposed to be encoded through suitable numerical vectors. The output of each encoding network is the code computed for the corresponding chemical fragments.
Figure 9.10: Examples of encoding networks with \( n = 3 \) and \( m = 2 \) (left side) for the chemical fragments \(-\text{CH}_3\) and \(-\text{CH}_2\text{-CF}_3\). The labels of the chemical trees represent the atom types: H is represented by \([1, -1, -1]\), C by \([-1, 1, -1]\) and F by \([-1, -1, 1]\). Void subgraphs are encoded by the null vector \( \mathbf{x}_0 \). The output of each encoding network is the code computed for the corresponding chemical fragments (i.e., \( \mathbf{x}_{\text{CH}_3} \) and \( \mathbf{x}_{\text{CH}_2\text{-CF}_3} \), respectively).
“visit” the nodes of different input graphs. This is a consequence of the use of a stationary model.

The neural network output for a given molecular graph is obtained by completing the corresponding encoding network with the neural realization of $g()$. Such completed network is trained on the regression task. Thus, both the weights of the hidden recursive neurons and the weights of the output neuron (realizing $g()$) are trained simultaneously on the training set. As a result of this joint training, the encoding of the molecular graph is adaptive, since it is computed on the basis of the specific regression task.

There are different ways to realize the recursive neural network (see Chapter 4). In the present work we choose to use a constructive approach that allows the training algorithm to progressively add the hidden recursive neurons during the training phase. The model is an (recursive) extension of Cascade Correlation based algorithms [47, 46] (Section 4.3). The built neural network has a hidden layer composed of recursive (hidden) units. The recursive hidden units compute the values of $\tau_E$ (in $R^m$) for each input DOAG, as shown in Figure 9.9 or in Figure 9.10. The number of hidden units, i.e. the dimension $m$ of the descriptor space, is automatically computed by the training algorithm, thus allowing an adaptive computation of the number and type of (numerical) descriptors needed for a specific QSPR/QSAR task. In the Recursive Cascade Correlation for structures (RCC) model, in order to realize the function $g$, we use a single standard linear output neuron. A complete description of the Cascade Correlation for structures algorithm and a formulation of the learning method and Equations has been shown in Section 4.3.

In summary, the hidden layer of a recursive network produces a numerical vectorial code (i.e., its internal representation) that represents the input molecular graph. In terms of QSPR/QSAR studies, we can imagine that each hidden recursive neuron calculates an adaptive topological index on the basis of the information supplied to the model (i.e., the training set). The outputs of the hidden units are arranged into a vector of these topological indexes and used as input for a linear regression model realized by the output unit (the $g()$ function), as shown in Figure 9.7. It is important to stress that these topological indexes are automatically developed by the neural network, since they arise from the training process as a function of the relationship between structures and corresponding values of the target property. They are developed, for this reason, independently from the domain knowledge.

The advantage of this new approach is that it allows us to describe and to process a molecular graph in a way that considers both the graph topology (connectivity) and the atom types (or the chemical functionalities). The use of a neural network to realize the encoding and regression functions allows the production of a flexible prediction model. However, the use of a “black-box” approach to implement the encoding and the regression functions raises, especially for QSAR, the following issues:

- chemical meaningfulness of the numerical descriptors produced by the recursive neural network;
- relationship between the developed numerical codes and the qualitative aspects of the QSAR problem.
Those issues were partially addressed in the following by studying the internal representations developed by the recursive neural network trained on a specific family of benzodiazepines. Examples of such results are reported in Section 11.1.4 (and in [128]).

A complete answer to these issues would allow the extraction of the knowledge learned by the neural network, posing the basis for a full understanding by human experts of the model and therefore permitting the assessment of the model as a new tool for the rational design of new molecules.

### 9.4 Representational Issues

A specific type of representation of the molecular structure is required for the model presented here. The choice of the representation defines the function $\tau_R$ introduced in Figure 9.7. Since the functions $\tau_E$ and $g$ are automatically developed by the model, in the new QSPR/QSAR scheme the specification of function $\tau_R$ is the only one available for the designer’s tuning.

Molecular structural formulas have already been treated in literature as mathematical objects (graphs) according to chemical graph theory. In our case, a representation of molecular structures in terms of DOAGs is required. The candidate representation should contain the detailed information about the shape of the compound, the atom types, the bond multiplicity, and the chemical functionalities, and finally it should retain a good similarity with the representations usually adopted in Chemistry.

When the molecular structure is represented as a DOAG, the main representational problems which are encountered are: (i) how to represent cycles, (ii) how to give a direction to edges, and (iii) how to define a total order over the edges.

An appropriate description of the molecular structures analyzed in this work is based on a labeled tree representation. To exemplify, we can consider two paradigmatic instances of the problem described in the following chapters, one for QSPR analysis, and one for QSAR analysis.

The first example is given by alkanes. In this case each carbon-hydrogens can correspond to a node of the tree, the root of the tree can be determined by the first carbon-hydrogens group according to the IUPAC nomenclature system, cycles are absent and the total order over the edges can be based on the size of the sub-compounds.

In the case of benzodiazepines, the major atom group that repeats unchanged throughout the class of analyzed compounds (common template) constitutes the root of the tree. When other repeating atom groups do exist in all the analyzed molecules, single atoms, belonging to these groups, do not require to be explicitly represented. Each atom that requires to be explicitly represented or each repeating atom group corresponds to a node of the tree. Each bond that requires to be explicitly represented corresponds to an edge. A label is associated with each node. Here, these labels are just used to discriminate among

---

6 An alternative representation, which the model was able to deal with, would have been to explicitly represent each atom in the major atom group. However, since this group is repeated for all the compounds, no additional information is conveyed by adopting this representation.
9.4. REPRESENTATIONAL ISSUES

different atoms (or atom groups) and do not contain any physico-chemical information. The use of DOAGs for the molecular description implies the loss of only minor structural information. At the present level of development of the model, cycles are usually treated as repeating atom groups, for which a single label is used. When different types of cycles are present at corresponding positions of the molecular structure throughout the class of analyzed compounds, different labels are used to describe them.

The representational scheme described above basically solves all the representational problems (i)-(iii). In fact, with reference to the benzodiazepines data set, concerning the first problem, since cycles mainly constitute some common shared template of the benzodiazepines compounds, it is reasonable to represent them as a single node where the attached label codifies information about their chemical nature. The second problem was solved using the major common template as the root of a tree representing a benzodiazepine molecule. For the third problem, the total order over the edges follows a set of rules mainly based on the size of the molecular fragments. Finally, the degree of each vertex conveys information concerning the bond multiplicity.

Rules that allows us to define the function \( \tau_R \) according to the above ideas will be specified in each section describing the different tasks. This should prove the flexibility of such rules to adapt at drastically different problems, allowing the designer to exploit the high abstract and graphical description offered by the structured domain. In particular the designer can fix the amount of molecular structure that need to be explicitly described (including prior knowledge in such representation). Defining the information represented by the vertexes the designer can easily specify the “granularity” (or the levels of abstraction) of the structured representation. A further flexibility it is offer by the label encoding. Specifically, giving a numerical code for each vertex label of the domain it is possible to specify the degree of similarity among the different type of vertexes (assuming some prior knowledge): orthogonal vectors can be used when we like to distinguish labels without taking any assumption for properties shared among the vertexes. Besides, the label can be also used to convey other information corresponding to designer selected features.

It is finally worth to note that, since the model is adaptive and can modify its behavior, specifically the encoding process, according to the training data (i.e. to the task), the arbitrariness that can be results from the rules (e.g to solve problems (i) and (iii)) can be partially or totally compensated by the learning process.

---

7We distinguish different principal heterocycles or cycles that appear as substituents using different labels.

8This is further point showing the flexibility of the approach. Actually, the representations used in our applications does not relay at all in the a priori coding of molecular features. i.e. no physico-chemical descriptor was used by our model.
Chapter 10

QSPR Experiments

10.1 QSPR Analysis of Alkanes

The analysis of alkanes for the prediction of the boiling point has been introduced in Section 2.3.3 as a paradigmatic example of QSPR problem. Since it is frequently used to test QSPR models, the problem is study here for comparison aims with respect to known approaches. The prediction task is well characterized for this class of compounds, since the boiling points of hydrocarbons depend upon molecular size and molecular shape, and vary regularly within a series of compounds, which means that there is a clear correlation between molecular shape and boiling point. Moreover, the relatively simple structure of these compounds is amenable to very compact representations such as topological indexes and/or vectorial codes, which are capable of retaining the relevant information for prediction. For these reasons, multilayer feedforward networks using ‘ad hoc’ representations yield very good performances.

In order to perform a comparison with our method, we decided to use as reference point the work described in [30] which uses multilayer feedforward networks. The data set used in [30] comprised all the 150 alkanes with 10 carbon atoms. Cherqaoui et al. use a vectorial code representation of alkanes obtained by encoding the chemical graph (tree) with suppressed hydrogens through an “N-tuple” code, as seen in Section 9.2.2. Each component of the vectorial code, which in this case is of dimension 10, represents the number of carbon bonds for each carbon atom (see Figure 9.3). The last components are filled by zeros when the number of atoms of the compound is less than 10. The uniqueness of the code is guaranteed by keeping a lexicographic order.

This representation for alkanes is particularly efficient for the prediction of the boiling point since it is well known that the boiling point is strongly correlated with the number of carbon atoms and the branching of the molecular structure. However, the same representation could be useless for a different class of compounds and different tasks.
10.1.1 Representation of Alkanes

We observe that the hydrogen suppressed graphs of alkane molecules are trees and they can be represented as ordered rooted trees by the following minimal set of rules:

1. the carbon-hydrogens groups (H, C, CH, CH₂, CH₃) are associated with graph vertices while bonds between carbon atoms are represented by edges; this is equivalent to consider a hydrogen suppressed graphs, i.e. to disregard the information about the hydrogen atoms as typically made for this task;

2. the root of the tree is defined as the first vertex of the main chain (i.e., the longest chain present in the compound) numbered from one end to the other according to IUPAC rules [87] (the direction is chosen so to assign the lowest numbers possible to side chains, resorting, when needed, to a lexicographic order); moreover, if there are two or more side chains in equivalent positions, instead of using the IUPAC alphabetical order of the radicals names, we adopt an order based on the size of the side chains (i.e., depth of substructure);

3. the orientation of the edges follows the increasing levels of the trees;

4. the total order on the subtrees of each node is defined according to the depth of the substructure; we impose a total order on the three possible side chains occurring in our data set: methyl < ethyl < isopropyl.

Examples of representations for alkanes are shown in Figure 10.1.
The complete lists of the compounds, according with our representation, along with the target and the output results are reported in [21].

10.1.2 Experimental Results (Alkanes)

As the target output for the networks we used the boiling point in Celsius degrees normalized into the range \([-164, 174\) °C] (scaled by a factor of 100 in the simulations). A bipolar localist representation encoding the atom types was used. Indeed, for alkanes, the only distinguished atom is the hydrogen, that is present only in one instance to correctly saturate the methane compound. This first strategy has been changed for the experiments presented in Chapter 6 adopting a uniform labeling of carbon groups.

For the sake of comparison, we tested the prediction ability using exactly the same 10-fold cross validation (15 compounds for each fold) used in [30]: no ensemble techniques were adopted to compute our results. We repeated the procedure for four times. Learning was stopped when the maximum absolute error for a single compound was below 0.08 (i.e. 8 °C).

The obtained results for the training data are reported in Table 10.1 (°C) and compared with the results obtained by different approaches, i.e., the results obtained by Cherqaoui et. al. using ‘ad hoc’ Neural Networks, two different equations based on connectivity
10.1. QSPR ANALYSIS OF ALKANES

Figure 10.1: Example of representations for alkanes.

(χ) topological indexes, and multi-linear regression over the vectorial code for alkanes. The results obtained on the test set are shown in Table 10.2 and compared with the MLP results obtained by Cherqaoui et al. For completeness we have reported the cumulative results from a set of several trials of our model in row 3 of Table 10.2. It must be pointed out that the results are computed by removing the methane compound from the test set (for the MLP and RCC in Table 10.2), since it turns out to be an outlier. Particularly, from the point of view of our new approach that considers the structure of compounds, methane (CH₄) is so structurally small that it does not represent a typical element in the class of alkanes.

In Figure 10.2 the residual errors for each compound are reported. The complete list of data is reported in Appendix of the [21]. Examples of training and test curves for two different instances of Cascade Correlation networks, trained over the same fold, are shown in Figure 10.3.

10.1.3 Discussion

The behavior of the model for the prediction of the boiling point of alkanes demonstrates the ability of the model to be competitive with respect to ‘ad hoc’ techniques. In fact, the obtained results compare favorably with the approach proposed by Cherqaoui et al. bearing in mind that the vectorial representation of alkanes retains the structural information which is known to be relevant to the prediction of the boiling point. Due to the
Figure 10.2: Outputs obtained by Cross-Validation by MLP (a) and RCC (c) versus the desired target for test data. Zooms in the range [100,180] of the plots are shown in (b) and (d), respectively. The values are expressed in °C.

Figure 10.3: Mean training and test error for two different instances of Recursive Cascade Correlation networks trained over the same fold. The mean error is plotted versus the number of inserted hidden units.
### 10.2 Prediction of Thermodynamic Properties

The prediction of the biological activity of a drug requires knowledge of the interaction mechanism between the molecule and the specific site of action of the receptor but is determined also by the physicochemical parameters regulating the absorption into the cells. The transport of a molecule from the aqueous medium of the extracellular region to the cytoplasm is ruled by passive diffusion across the lipid bilayer of the membrane. The lipophilicity of solutes, as expressed by their partition coefficient between water and a suitable apolar organic solvent, has often been considered a key parameter to model passive diffusion. In this respect, experimental evaluation and rationalization of thermo-

Table 10.1: Results obtained for alkanes on training data set by Recursive Cascade Correlation (RCC), by Cherqaoui et. al. using ‘ad hoc’ neural networks (MLP), by using topological indexes and by using multi linear regression. The data are obtained by a 10-fold cross-validation with 15 compounds for each fold. The correlation coefficient (R) and the standard deviation of error (S) are reported.

<table>
<thead>
<tr>
<th>Model</th>
<th>#Units</th>
<th>Mean Abs. Error</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCC (Mean)</td>
<td>110.7</td>
<td>1.98</td>
<td>0.99987</td>
<td>2.51</td>
</tr>
<tr>
<td>Best MLP</td>
<td>7</td>
<td>2.22</td>
<td>0.99852</td>
<td>2.64</td>
</tr>
<tr>
<td>Top. Index 1</td>
<td></td>
<td></td>
<td>0.9916</td>
<td>6.36</td>
</tr>
<tr>
<td>Top. Index 2</td>
<td></td>
<td></td>
<td>0.9945</td>
<td>5.15</td>
</tr>
<tr>
<td>MLR</td>
<td></td>
<td></td>
<td>0.9917</td>
<td>6.51</td>
</tr>
</tbody>
</table>

Table 10.2: Results obtained for alkanes on test data set by Recursive Cascade Correlation (RCC) and by ‘ad hoc’ neural networks (MLP). The data are obtained by a 10 fold cross-validation with 15 compounds for each fold. The last row of the Table is computed over four different cross-validation evaluations.

<table>
<thead>
<tr>
<th>Model</th>
<th>Mean Abs. Error</th>
<th>Max Abs. Error</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best MLP</td>
<td>3.01</td>
<td>10.42</td>
<td>0.9966</td>
<td>3.49</td>
</tr>
<tr>
<td>Best RCC</td>
<td>2.74</td>
<td>13.27</td>
<td>0.9966</td>
<td>3.5</td>
</tr>
<tr>
<td>Mean RCC</td>
<td>3.71</td>
<td>30.33</td>
<td>0.9917</td>
<td>5.43</td>
</tr>
</tbody>
</table>
dynamic solvation properties in two not miscible solvents are very important. Indeed, these quantities can be used to calculate the corresponding partition properties and to define a lipophilicity scale. Since a long time a thermodynamic investigation of the solvation of organic molecules in different solvents has been carried out. The collected information were rationalized mainly by using group or bond contribution methods [28] or empirical correlation based on solute molecular descriptors such as usual in QSPR analysis. The choice of a suitable group of descriptors for the modeling of examined systems represents a fundamental issue of these correlation procedures.

The recursive neural networks overcome this limit, allowing us to deal with a quite general and flexible representation of the compounds in terms of structures, which is able to convey much more information than standard vectorial representations.

In this Section, we report the use of recursive neural networks (RCC model) to describe the standard free energy of solvation, $\Delta_{\text{solv}}G^\circ$ [28], in water of a set of linear monofunctional organic compounds. Solvation properties were selected as target because of both the availability of a large number of literature data (which allows an effective training of the neural network) and the modelistic implications of these properties. We also present and discuss the rules used to represent the examined molecules as object in a structured domain in the form of labeled directed ordered trees. The results obtained by RCC are compared with those obtained by applying a standard group contribution method (GCM, presented in Section 10.2.1) to the same set of molecules. This comparison is meaningful since both approaches deal with a representation of the molecules directly inferred by the molecular structure alone. Moreover the GCM method has proved one of the most reliable in the prediction of thermodynamic properties of molecules in solution at infinite dilution and can be considered in some sense as the reference method in the solvation field.

This work represents the first step of a research project concerning the application of RNNs to the analysis of a wide number of thermodynamic properties in a QSPR/QSAR approach, carried out by collaboration with the Chemical Department of the University of Pisa [15].

10.2.1 Group Contribution Methods

Group contributions schemes have been extensively employed as a practical method for predicting a plethora of thermodynamic and related properties such as free energies and enthalpies of formation, enthalpies of vaporization, boiling points, vapour pressure of pure liquids, and so on. The basic idea underlying these approaches is that a solute molecule acts as a number of fragments contributing to the investigated property as independent of each other. These fragments can be constituted by individual atoms or simple atomic groups and usually coincide with portions of alkyl chains or with the functional groups characterizing the different classes of organic compounds. In particular, several schemes of group additivity have been developed in order to rationalize and predict thermodynamic functions of solvation and partial molar properties at infinite dilution of a wide variety of compounds in solvents of different features. The general form of equation commonly
employed in these additivity schemes is:

$$Y = \sum_j n_j B_j$$  

(10.1)

where $Y$ is the thermodynamic property of interest, and $B_j$ is the contribution to the property of the $j$-th group present $n_j$ times in the solute structure. The usual procedure employed for determining the values of the group contributions $B_j$ is based on a linear multiparameter regression. Though these procedures are essentially empirical, at least as regards free energies of solvation, some molecular basis can be found in the Ben-Naim’s theory [12]. Such theory, studying the chemical potential of the molecules, show as the free energy of solvation can be approximated by a sum of contributions each related to the solvation of an individual fragment of the solute molecule. Hence, this method is recognized as the best method to describe the solvatation property starting from the molecules characteristics.

The group contribution methods can be seen as an instance of the framework introduced in Section 9.3. The function $\tau_R$ models the extraction of specific structural aspects from the molecules through the selection of the set of the relevant fragments, i.e. fragments that the experts known that contributes to the investigated property, along with the frequency of their occurrence. The function $\tau_E$ returns the fixed-width vector where each component is the numerical descriptors $n_j$ (frequencies of the $j$-th fragments in the input compound), and the position $j$ univocally identifies each group. The mapping is performed by the regression function $g$ according to Equation 10.1, where the $B_j$ are the free-parameters. More general schema have been studied to include constant terms and some other indexes accounting for peculiar structures, still in form of numerical indexes, that for polyfunctional compounds (i.e. compounds covering multiple substitutions of functional groups) acts as corrective terms.

10.2.2 Molecular Structure Representation

As introduced so far, in our approach based on recursive neural networks we move from the use of a flat representation of the compounds, which characterizes also the group contribution methods. For the data set and the task of prediction thermodynamic properties the description of the molecules is easily obtained from the structure formula by means of 2D graphs. The experts in this field observed that, although the structured representation conveys much more information than a vectorial representation of the compounds, the overall procedure should lead to a loss of information with respect to a 3D representation. However, part of the lost information can be recovered by introducing an order among the atomic clusters individuated as the ”groups” constituting the molecule, thus matching, in some sense, the well known Newmann projections employed to assign the absolute configuration of a chiral molecule.

The set of studied compounds is quite variegate and it includes: alkanes, alkenes, alkines, alcohols, ethers, thioalcohos, thioethers, aldehydes, ketones, acids, esters, amines, amides, halogenoalkanes, nitriles, nitroalkanes.
The chemical experts designed a set of rules for this task, aimed at obtaining a unique structured representation of each molecule in terms of labeled ordered rooted trees, according to the following criteria:

1. each molecule is partitioned in the following groups: \( \text{CH}_3, \text{CH}_2, \text{C}, \text{H}, \text{C} = \text{C}, \text{C} = \text{C}, \text{OH}, \text{O}, \text{C} = \text{O}, \text{NH}_2, \text{NH}, \text{N}, \text{SH}, \text{S}, \text{CN}, \text{NO}_2, \text{F}, \text{Cl}, \text{Br}, \text{I}; \)

2. each group corresponds to a node of the tree and each bond between them corresponds to an edge;

3. we decided an ‘ad hoc’ priority scale (see Appendix 10.2.5) for chemical groups and the root of the tree is placed on the group with the highest priority;

4. the orientation of the edges follows the increasing levels of the trees.

5. the total order on the subtrees of each node is defined according to the same priority scale for chemical groups used to select the root.

More details on the set of rules are reported in Appendix 10.2.5. It can be observed that we chose to divide the CH group into C and H. In such a way we maintain the same approach in describing the C-H bond independent of the hybridisation of the carbon atom. It is noteworthy that, by using the total order of the subtrees, we were able to build up different representations for the cis and the trans isomers of alkenes (see point 2 of Appendix 10.2.5) and for the R and the S enantiomers of an optically active molecule (point 2 of Appendix 10.2.5).

A numerical label is associated with each node. The labels discriminate among different groups of atoms and do not contain any physicochemical information. For labeling we basically use a bipolar localist representation, i.e. we use a 1-of-n coding scheme for categorical variables. The labels are represented by vectors of width 20, with one or a few specific bits turned on (+1) and all the others turned off (0). Sharing bits between different labels allows us to represent similarity between chemical groups. On the other hand, two orthogonal vectors (i.e. bits turned on in different positions within the vector) represent groups of different chemical nature. In particular we stated that:

- all the H-groups have the same numerical label.
- \( \text{CH}_3, \text{CH}_2, \text{C} \) have similar numerical labels.
- \( \text{NH}_2, \text{NH} \) have similar numerical labels, but orthogonal to N.
- \( \text{OH}, \text{O} \) and \( \text{SH}, \text{S} \) have orthogonal labels.
- \( \text{F}, \text{Cl}, \text{Br}, \text{I} \) have similar numerical labels.

The similarity or the orthogonality between groups, decided according to their chemical features, is the maximum of ”chemical information” transferred, for this task, to the RNN as input data.

Examples of representation which comply with the above rules are shown in Figure 10.4 for two chemical compounds, 2-methyl-2 propanol and 1-methoxy-propane.
10.2. PREDICTION OF THERMODYNAMIC PROPERTIES

![Molecule and Chemical Tree of 2-methyl-2-propanol and 1-methoxy-propane]

Figure 10.4: Example of representations for 2-methyl-2-propanol and 1-methoxy-propane.

10.2.3 Experimental Results

The first step of the method validation aim is to investigate if the general method can compare with specific solutions in the field, in particular in the cases (data sets) that yield very good performances for traditional methods. This is goal is carried out in the current task using the same data set used for the group contribution method, i.e., a set composed by a set of linear monofunctional organic compounds, for which both theoretical background and empirical results shown the high reliability of the group contribution method. Initially, we used two splitting of data to compose the test set: the first one is used to test the performance of the RCC on a set of data that yields the best performance for group contribution methods, and the second is composed selecting a “fear” test set to compare the prediction performance of the two approaches. The sampling of data has be done according to general criteria aimed at maximizing the molecular diversity, while including samples of the main classes of compounds in the test set. A third set of data is provided to study the prediction behavior of the two approaches in the cases that are considered
“difficult” by the experts in the applications fields.

Specifically, we studied the $\Delta_{\text{soln}} G^\circ$ in water of 178 linear molecules, as described in [28], containing: alkanes, alkenes, alkines, alcohols, ethers, thioalcohols, thioethers, aldehydes, ketones, acids, esters, amines, amides, halogenoalkanes, nitriles, nitroalkanes.

In the first preliminary experiment, the molecules were divided in a training set (145 molecules), and a test set (33 molecules) for the training and the validation process respectively. The target values are in kJmol$^{-1}$ (in the range $[-40, 12]$ kJmol$^{-1}$).

Four trials were carried out for the simulation. The initial connection weights used in each simulation were randomly set. Learning was stopped when the RCC algorithm inserted 100 hidden units. The obtained results for the training and test data are reported in Table 10.3, and summarized through a simple ensemble averaging method (i.e. computing the mean output over the four trials). Statistics on the mean absolute error (Mean Abs. Error), maximum absolute error (Max Abs. Error), the correlation coefficient (R) and the standard deviation of error (S), as defined in regression analysis, are reported.

<table>
<thead>
<tr>
<th></th>
<th>Mean Abs. Error (kJmol$^{-1}$)</th>
<th>Max Abs. Error (kJmol$^{-1}$)</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR</td>
<td>0.05</td>
<td>0.27</td>
<td>0.9998</td>
<td>0.077</td>
</tr>
<tr>
<td>TS</td>
<td>0.41</td>
<td>1.58</td>
<td>0.9983</td>
<td>0.587</td>
</tr>
</tbody>
</table>

Table 10.3: Results obtained on Training (TR) and Test set (TS) by RCC over 4 experimental trials by a ensemble averaging method.

Regarding the evaluation of such results, we can observe that the training values are reproduced in the range of real measured experimental error ($0.1$ kJmol$^{-1}$) and the test values are predicted with an error below $0.45$ kJmol$^{-1}$. For a standard group contribution method ([28]), this value is the mean error on the “training set” composed of all the 178 molecules. It is worth to note that such error values are largely below 1%. The results suggest that the neural networks recursive approach can potentially achieve results at the state of art in the fields.

For the second splitting of the data set the test set has been randomly chosen so to compare the two methods on a fear basis. The target values of the training set is in this case in the range $[-28, 12]$ kJmol$^{-1}$. The training set has 139 compounds and the test set has 33 compounds. The results of the RCC is compared with those obtained by applying the group contribution method to the same set of compounds.

The main statistics computed over all the simulations for the training set and test set are reported in Table 10.4. Specifically, the results obtained by RCC and by group contribution method are reported: the mean absolute error (Mean A. Err.), the maximum absolute error (Max A. Err.), and standard deviation of error (S), as defined in regression analysis, are reported for the two approaches, respectively. The statistics for Recursive Cascade Correlation are summarized over 8 trials by an ensemble averaging method.

Regarding the evaluation of the results in Table 10.4, we can observe that the test
Table 10.4: Results obtained on Training (TR) and Test set (TS) by RCC (over 8 experimental trials by an ensemble averaging method), and by group contributions method (GCM).

The performances of RCC is very close to the group contribution methods, that is a state of the art results for the prediction of this property [28]. The result on the training set reflects the inherent flexibility of the RCC model which abstracts, through the adaptive process, much more information from the data set with respect to the standard group contributions method. On the other hand, the RCC would need a rather wider data set in order to achieve a further improvement in its predicting ability.

Finally, a small set of data (7 compounds taken from the whole data set) has been selected to test the prediction performance of the trained models. This “prediction set” contains compounds that are not well represented in the training set, allowing to compare the predictive power of the two approaches at the sampling limits of the training data. The models are trained on the training set used for the second splitting of data.

The results for each compound are reported in Table 10.5. For each compound we report the target (experimental measured property), the error of the group contribution method (GCM) and the mean error over the 8 trials (predicted value) of RCC.

Table 10.5: Results obtained for each compound of the prediction set by RCC and by group contributions method (GCM).

Concerning the results of Table 10.5, note that the methane compound cannot be handle by a standard group contribution method (lacking for the definiton of a group H bounded to a sp$^3$ C), hence the first results in Table 10.5 is not available (N.A.) for
group contribution method. RCC had satisfactorily resolved this case. The prediction error for the acetamide compound results quite high: it should be noted, however, that such target value is out of the target values range in the training set (\([-28, 12\) kJmol\(^{-1}\]) and that the response of the model is close to the minimum known value. A comparison with the group contributions method is not possible in case of acetamide due to the lack of other occurrences of the amide group in the data set. The compounds in row 2, 3 and 4 contains conjugated groups: the results show that RCC, trained with this data set, is not able to outperform the group contribution method for the conjugation cases. RCC outperforms the group contribution method on haloalkenes (last two row in Table 10.5).

10.2.4 Discussion

The current experiments allow to show the ability of RCC model to achieve results that are comparable to the best known systems to describe the solvation property starting from the molecules characteristics, i.e. the group contributions method. We used a set of molecules for which the group contributions method achieves the best performance, according to the background chemical theory. The main aim was to investigate if a very general methodology, such as the one proposed, can be compared, in term of performance, with a method working for homologous classes of compounds achieving a low error in prediction (note that the error is below the threshold of 1% and it is very close to the measurement experimental errors for such properties). Moreover, the last experiments show the potentiality of the new approach to deal with compounds that group contributions methods cannot properly handle. These preliminary steps were useful to validate the new method. However, more extensive experiments are planned to refine the comparison with group contribution methods: we would extend the study to polyfunctional compounds where it is known that the theoretical limitations of the group contributions method are stronger and the method requires the introduction of corrective terms. In fact, the ultimate goal is the extension of the applicability of the prediction methods to compounds and problems not covered by the standard methods.
10.2.5 Appendix

In the following we provide the rules for the representation of the molecular structure used for experiments of Section 10.2, including alkanes, alkenes, alkines, alcohols, ethers, thioalcohols, thioethers, aldehydes, ketones, acids, esters, amines, amides, halogenoalkanes, nitriles, nitroalkanes.

1. each molecule is partitioned in the following groups: CH\_3, CH\_2, C, H, C=C, C≡C, OH, O, C=O, NH\_2, NH, N, SH, S, CN, NO\_2, F, Cl, Br, I;

2. each group corresponds to a node of the tree and each bond between them corresponds to an edge;

3. we decided an 'ad hoc' priority scale for chemical groups and the root of the tree is placed on the group with the highest priority;

4. the orientation of the edges follows the increasing levels of the trees.

5. the total order on the subtrees of each node is defined according to the same priority scale for chemical groups used to state the root.

1. **Priority scale:**

   (a) Molecules containing only carbon and hydrogen atoms.

   i. Alkanes: The groups of the longest chain are numbered beginning with the end that is closest to an alkyl substituent and the root is fixed on the first group. If two (or more) alkyl substituents are present at equal distance from the two ends of the longest chain, the elements along the substituent chains are ranked until a point of difference is reached at which a distinction in priority is possible. The substituent of lowest priority is hydrogen (E.g. \(-\text{CH}_2\-\text{CH}_3 > -\text{CH}_3; \text{CH}_2\-\text{C}(\text{CH}_3)_3 > -\text{CH}_2\-\text{CH}_3\)). The groups of the longest chain are then numbered beginning with the end closest to the higher priority branched chain.

   ii. Alkenes and dienes: the main chain is the longest one that includes the functional group C=C. Numbering of the chain starts at the end farther from the double bond and the root is set on the first group. In dienes with both terminal double bonds the root is set on the apical hydrogen atom of the more substituted double bond.

   iii. Alkines and enines: the root is set on the triple bond.

   (b) Molecules containing heteroatoms (N, O, S, F, Cl, Br, I).

   i. A functional group containing a heteroatom has higher priority than any other functional group and the root is set on it.
ii. In molecules with different functional groups, the C=O has the highest priority. This choice allows to give a similar representation to ketones, aldheydes, carboxylic acids, esthers, alkanoyl chlorides, anhydrides, and amides. As far as the other groups are concerned the priority is assigned on the basis of the heteroatom. The priority decreases going to the right in the group of the periodic table (N>O>F) and down in the period (O>S; F>Cl>Br>I). Fixed the heteroatom, the priority among the groups follows the order: (a) nitrogen N>NH>NH2>NO>NO2; (b) oxygen OH>O; (c) sulphur C=S>S>SH=S>O.

iii. If two or more identical functional groups are present the root is set on the inner one in order to minimize the depth of the structure.

iv. In polyhaloalkanes the root is fixed on the highest priority halogen atom bound to the carbon bearing the highest number of the same halogen (E.g. Froot-CF2-CF2Cl). In polyhaloalkenes the root is fixed on the highest priority halogen atom bound to the sp2 carbon bearing the highest number of the same halogen (E.g. Froot-CF=CF-CF3).

2. Edges order:

(a) The edges starting from a node are ordered following the groups priority rules. If two (or more) substitutes in a node have the same priority the elements along the substituent chains are ranked until a point of difference is reached at which a distinction in priority is possible.

(b) In a double bond the edges are numbered starting from the cis position with respect to the root and follow the order cis>trans>gem. When the double bond stereoisomery is cis, absent or not specified, positions 1 and 2 are occupied according to the priority rules of the groups. If the double bond stereoisomery is trans the higher priority group occupies position 2.

(c) In a triple bond the edges are ordered following the groups priority rules.

(d) We can distinguish R and S enantiomers by ordering the edges of the asymmetric carbon according to the priority rules. The second enantiomer is obtained by changing the order of two edges.
Chapter 11

QSAR Experiments

11.1 QSAR Analysis of Benzodiazepines

Due to the strong therapeutic interest [65] and to the multiplicity of SAR studies of this class of compounds, benzodiazepines (Bz) were chosen as the starting application domain for QSAR analysis. At this stage, a group of 1,4-benzodiazepin-2- ones, previously studied by Hadjipavlou-Litina and Hansch [65] through traditional QSAR equations, was selected for testing our model, the evaluation of the method being the initial step of its application. The task is the prediction of the non-specific activity (affinity) towards the Bz/GABA<sub>A</sub> receptor. The affinity can be expressed as logarithm of the reciprocal of the drug concentration C (mol./liter) able to give a fixed biological response<sup>1</sup>. The data set analyzed by Hadjipavlou-Litina and Hansch (see Table 2 of [65]) is characterized by a good molecular diversity, and this last requirement makes it particularly significant for QSAR analysis. The total number of molecules analyzed was 77. A list of the compounds (used to compose one of the training data set) is reported in the in Table 11.6 in the Appendix of the Chapter.

All the molecules present a common template consisting of the Bz nucleus (in three compounds the A ring of the Bz nucleus consists of a thienyl instead of a phenyl group) and they differ in a variety of substituents at the positions shown at the left side of Figure 11.1.

11.1.1 Representation of Benzodiazepines

The labeled tree representation of a Bz is obtained by the following minimal set of rules:

1. the root of the tree represents the Bz nucleus;

2. the root has as many subtrees as substituents on the Bz nucleus, sorted according to the order conventionally followed in Chemistry (standard IUPAC numbering of molecules);

<sup>1</sup>In order to characterize the fixed response, the drug concentration able to give half of the maximum response (IC<sub>50</sub>) is commonly used.
substituent positions);

3. each explicitly represented atom (or any other common atomic group) of a substituent corresponds to a node, and each explicitly represented bond\(^2\) to an edge; the root of each subtree that represents the substituent is the atom directly connected to the common template, and the orientation of the edges follows the increasing levels of the trees;

4. different atoms (or any other common atomic group) are represented by different labels, and each node in the trees has a label associated;

5. the total order on the subtrees of each node is hierarchically defined according to: 
   i) the subtree’s depth,  
   ii) the number of nodes of the subtree,  
   iii) the atomic weight of the subtree’s root.

In the analyzed data set different labels are used for the following atoms: C, N, O, F, Cl, Br, I, H. Moreover we use a different label for each of the following atomic groups: bdz (Bz nucleus), bdztg (Bz nucleus where the A ring is a thienyl group instead of a phenyl one) and ph, py, cya, naf, respectively, for fragments of Phenyl, 2-pyridyl, Cyclohexenyl, Cyclohexyl and Naphthyl. For labeling we basically use a bipolar localist representation, as shown in Section 9.3, extended with a coding of the vertex arity.

An example of representation for benzodiazepines, which complies with the above rules is shown in Figure 11.1 (compound #60 in Table 11.6 in the Appendix of the chapter) (see also Figure 9.7). In Figure 11.2 the representation of a specific substituent is shown.

### 11.1.2 Experimental Results (Benzodiazepines)

For the analysis of the benzodiazepines data set four different splittings in disjoint training and test sets of the data were used (Data set I, II, II, and IV, respectively). Specifically,
the first training set (with a test set of 5 compounds) has been chosen as it contains the same compounds used by Hadjipavlou-Litina and Hansch. The second data set is obtained from Data set I by removing 4 racemic compounds from the training set and one racemic compound from the test set. This allows the experimentation of our approach without the racemic compounds which are commonly recognized to introduce ambiguous information. The test set of Data set III (5 compounds) has been selected as it simultaneously shows a significant molecular diversity and a wide range of affinity values. Furthermore, the included compounds were selected so that substituents, already known to increase the affinity on given positions, appear in turn in place of H-atoms, which allows the decoupling of the effect of each substituent. So, a good generalization on this test set means that the network is able to capture the relevant aspects for the prediction. The test set of Data set IV (4 compounds) has been randomly chosen so to test the sensitivity of the network to different learning conditions. The training set III, with the used numbering of the molecules, is reported in Table 11.6 in the Appendix.

As target output for the networks we used $\log(1/C)$ (scaled by a factor of 10 in the used data set). Six trials were carried out for the simulation involving each one of the different training sets. The initial connection weights used in each simulation were randomly set. Learning was stopped when the maximum error for a single compound was below 0.4. This tolerance is largely below the minimal tolerance needed for a correct classification of active drugs.

The main statistics computed over all the simulations for the training sets are reported in Table 11.1. Specifically, the results obtained by Hadjipavlou-Litina and Hansch, as well as the results obtained by the null model, i.e., the model in which the expected mean value of the target is used to perform the prediction (Section 4.4), are reported in the first and second row, respectively. For each data set, statistics on the number of inserted hidden
units are reported for the Recursive Cascade Correlation network. The mean absolute error (Mean Abs. Error), the correlation coefficient (R) and the standard deviation of error (S), as defined in regression analysis, are reported in the last three columns, respectively. Note that Mean Abs. Error, R and S for Recursive Cascade Correlation are obtained by averaging over the performed trials (six trials) (and not using a committee); the minimum and maximum values of the mean absolute error over these six trials are reported as well. Concerning the evaluation of the i-strategy (see Section 5.4) we have reported for the Data set II the results obtained when using Recursive Cascade Correlation without i-strategy on the number of training epochs (nis), by using the i-strategy (is), and by an empirically tuned version of the i-strategy (tis).

The results for the corresponding test sets are reported in Table 11.2. In case of small test data sets the correlation coefficient is not meaningful so we prefer to report the maximum absolute error for the test data (Max Abs. Error), calculated as the average over the six trials, and the corresponding minimum and maximum values of the maximum absolute error obtained for each trial.

In Figures 11.3 and 11.4 we have plotted the error of the network versus the desired target for data set I and III. Moreover, for the sake of comparison, in Figure 11.3 the error obtained using an equational approach [65] on data set I is reported as well.

Each point referring to the neural networks models in the plots represents the average error, together with the deviation range, as computed over the six trials (i.e., the extremes of the deviation range correspond to the minimum and maximum output values computed over the six trials for each compound).

<table>
<thead>
<tr>
<th>Training Set</th>
<th>Mean #Units (Min-Max)</th>
<th>Mean Abs. Error (Min-Max)</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLH</td>
<td>0.311</td>
<td>0.847 (0.390)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null model</td>
<td>0.580</td>
<td>0</td>
<td>0.702</td>
<td></td>
</tr>
<tr>
<td>Data set I nis</td>
<td>29.75 (23-40)</td>
<td>0.110 (0.099-0.120)</td>
<td>0.99979</td>
<td>0.127</td>
</tr>
<tr>
<td>Data set II is</td>
<td>15.3 (13-17)</td>
<td>0.100 (0.076-0.114)</td>
<td>0.99978</td>
<td>0.130</td>
</tr>
<tr>
<td>Data set II tis</td>
<td>34.0 (27-38)</td>
<td>0.087 (0.080-0.102)</td>
<td>0.99982</td>
<td>0.117</td>
</tr>
<tr>
<td>Data set III</td>
<td>19.7 (18-22)</td>
<td>0.087 (0.072-0.105)</td>
<td>0.99985</td>
<td>0.098</td>
</tr>
<tr>
<td>Data set IV</td>
<td>16.5 (13-20)</td>
<td>0.099 (0.078-0.132)</td>
<td>0.99976</td>
<td>0.131</td>
</tr>
</tbody>
</table>

Table 11.1: Results obtained for benzodiazepines on training data set I by Hadjipavlou-Litina and Hansch (HLH, first row), by a “null model” (second row) and on all the training data sets by Recursive Cascade Correlation. Results obtained for different learning settings (i-strategy) are reported for training data set II. The mean absolute error, the correlation coefficient (R) and the standard deviation of error (S) are reported.
### Table 11.2: Results obtained for benzodiazepines on test data set I by Hadjipavlou-Litina and Hansch (HLH, first row), by a “null model” (second row) and on all the test data sets by Recursive Cascade Correlation. Results obtained for different learning settings (i-strategy) are reported for training data set II. The mean absolute error, the mean of the maximum of the absolute error, and the standard deviation of error (S) are reported.

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Data #</th>
<th>Mean Abs. Error (Min-Max)</th>
<th>Mean Max Abs. Error (Min-Max)</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLH</td>
<td>5</td>
<td>1.272</td>
<td>1.750</td>
<td>1.307</td>
</tr>
<tr>
<td>Null model</td>
<td>5</td>
<td>1.239</td>
<td>1.631</td>
<td>1.266</td>
</tr>
<tr>
<td>Data set I</td>
<td>5</td>
<td>0.720 (0.611-0.792)</td>
<td>1.513(1.106-1.654)</td>
<td>0.842</td>
</tr>
<tr>
<td>Data set II nis</td>
<td>4</td>
<td>0.757 (0.703-0.810)</td>
<td>0.991 (0.839-1.142)</td>
<td>0.792</td>
</tr>
<tr>
<td>Data set II is</td>
<td>4</td>
<td>0.662 (0.501-0.807)</td>
<td>0.839 (0.661-1.088)</td>
<td>0.683</td>
</tr>
<tr>
<td>Data set II tis</td>
<td>4</td>
<td>0.546 (0.444-0.653)</td>
<td>0.727 (0.523-0.973)</td>
<td>0.579</td>
</tr>
<tr>
<td>Data set III</td>
<td>5</td>
<td>0.255 (0.206-0.325)</td>
<td>0.606 (0.433-0.712)</td>
<td>0.329</td>
</tr>
<tr>
<td>Data set IV</td>
<td>4</td>
<td>0.379 (0.279-0.494)</td>
<td>0.746 (0.695-0.763)</td>
<td>0.460</td>
</tr>
</tbody>
</table>

Figure 11.3: Output of the models proposed by Hadjipavlou-Litina and Hansch (left) and for the Recursive Cascade Correlation network (RCC) (right) versus the desired target; both models use the same training and test sets (data set I). Each point in the right plot represents the mean expected output for Recursive Cascade Correlation network, together with the deviation range (minimum and maximum values), as computed over six trials. The tolerance region is shown on the plots.
11.1.3 Evaluation of the $i$-strategy

To complete the model evaluation we present in the following a brief analysis of the learning behavior, especially considering the problem of overfitting, by discussing some learning curves for the recursive neural networks.

Due to the small number of training examples we considered various learning strategies in order to avoid or mitigate the overfitting problem. We fully described the adopted strategies in Section 5.4: basically we control the gains of the sigmoids, and the increase of the weight values through an incremental strategy on the number of training epochs for each new inserted hidden node.

The results reported in Table 11.1 and Table 11.2 confirm the effectiveness of the $i$-strategy for learning. In fact, the improvements observed for Data Set II show a significant reduction both in the mean number of hidden units and in the mean absolute error for the test set. The improvement in generalization does not seem to be directly correlated with the mean number of hidden units, since when using the tuned version of the $i$-strategy (tis), the mean number of hidden units increases with respect to the standard $i$-strategy (is) while the generalization error decreases.

In addition, we can also observe the dynamical behavior of the models showing the learning curves with respect to the use of the $i$-strategy. In Figure 11.5, plot (a) shows typical learning curves for the training set with and without adopting the $i$-strategy. The training error with $i$-strategy is higher than the training error without $i$-strategy for the first inserted hidden units; however, with the increase in the number of hidden units this relationship is inverted. Moreover, plot (b) in Figure 11.5, which reports learning curves
for the test set, clearly shows that overfitting does occur when training does not use the \textit{i-strategy}, while it does not occur when using the \textit{i-strategy}. The global result is that, using the \textit{i-strategy}, the better training error shown in Figure 11.5 (a) is combined with a better generalization performance.

![Graph](image)

**Figure 11.5:** Mean training and test error for two different instances, using or not using the \textit{i-strategy}, of Cascade Correlation for structures networks trained over the data set I. The mean error is plotted versus the number of inserted hidden units. The \textit{i-strategy} allows to reach a lower training error using less units (a). The test error decreases as a function of the number of hidden units only when using the \textit{i-strategy} (b).

Concerning the other learning parameters, an initial set of preliminary trials was performed in order to determine an admissible range. However, no effort was done to optimize these parameters.

### 11.1.4 Internal Representation Analysis

In order to understand the degree to which the proposed model is able to capture relevant domain knowledge from the training data, we investigated the internal representations, i.e. the output of hidden units, developed by the neural network trained with the selected set of benzodiazepines.

The outputs of hidden units correspond to the encoding values generated for each compound or molecular fragments in the data set. Some of these fragments exactly correspond to the substituents attached to the main common template; other fragments are part of the substituents and do not have any chemical meaning.

Since the information about the morphological characteristics of the chemical compounds is directly given in input to the model as labeled trees, it is possible to perform a direct analysis of the computed values for these numerical codes associated to each compound and its subcomponents.

For this investigation we performed a Principal Component Analysis (PCA) of the internal representations. Due to the relatively large dimensionality of the representational
space (typically around 20-30 hidden units are inserted by the training algorithm), we studied 2-D plots of the first two principal components. The aim was to show, as a first approximation, the relative distance and position of internal representations and how they cluster within the representational space of the model. We expect the configurations of the points in the plots to approximately describe the knowledge learned by the neural network from the training data.

From previous SAR studies some positions of the Bz nucleus are recognized to be the ones where substituents play significant roles in determining the biological activity also in relation to their specific chemical characteristics: positions 1, 7 and 2' ([65] and references therein). Within the class of compounds analyzed the above mentioned positions appear to be widely sampled.

In brief, the most important characteristics required for substituents at position 1 concern lipophylicity and steric hindrance, while the ones required for substituents at position 7 and 2' (or 2' and 6'), mostly concern the electronic effect. Lipophylicity ($\pi = \log P$) and electronic effect of the substituents (Hammett $\sigma$ constant) constitute the most popular physico-chemical descriptors employed in the traditional equation based Hansch approach [79, 78]. Substitutions at positions 6, 8, and 9 are known, instead, to decrease the affinity.

What we were interested in finding, through the analysis of the first two principal components was the presence of clusters possibly containing molecules grouped according to a classification amenable to the two above mentioned descriptors. As a first approach we reduced the relevant molecular descriptor to very simple entities, in order to make the analysis as clear as possible. From this perspective we collected into a unique class the lipophylicity ($\pi$) and steric hindrance descriptors, and only considered an on-off classification (molecules with a hydrogen atom or molecules with substituents, mostly lipophylic, at position 1). In the more detailed analysis reported in [128] we reduced the scale of the substituent effect values (Hammett $\sigma$ scale) to a few sub-classes corresponding to the effect that the substituents produce on well known chemical reactions (electrophilic substitution in aromatic compounds). But for the results reported here we again considered an on-off classification, i.e. presence or absence of halogens atoms (F, Cl, Br, I). In fact halogens atoms strongly affect the $\sigma$ values.

We then focused our interest on the analysis of the molecules on the basis of the substituents at position 2' or 2' and 6'. We considered three cases: (i) molecules with only one halogen substituent (at position 2'), (ii) molecules with two halogen substituents (at the symmetrical 2' and 6' positions) and (iii) molecules bearing no-halogen substituents at these positions.

The principal components of the internal representations developed by the Recursive Cascade Correlation (outputs of recursive hidden neurons) were analyzed for all the six experiments on data set III mentioned in Section 11.1.2.

Two representative plots of the first two principal components are shown in Figures 11.6 and 11.7. They show the biologically active molecules analyzed (compounds associated to a target) and the relevant molecular fragments.

Analysis of the plot shows that molecules and fragments are clustered on the basis
Figure 11.6: Principal component analysis of training compounds used in the experiment I derived from 28 output values of hidden neurons. Compounds characterized by $R_1=H$ (left side of the plot) and compounds bearing a substituent at position 1 (lower side of the plot) are grouped by contour lines. The circled sub-cluster on the right side includes compounds where the A ring of the Bz nucleus is a thienyl group instead of a phenyl one. See Table 11.6 in the Appendix for compound numbering.

of both morphological differences and specific chemical features, that can not be inferred directly by the observation of the molecular graph, rather only by the association of molecular structures and targets.

The plot (see Figure 11.6 for experiment I) appears to be split in two big clusters: all the substituents or molecular fragments approximately fall into its triangular upper right side, while all compounds to which a target is associated (molecules) approximately fall into its triangular lower left side. The plot obtained on the basis of experiment II (see Figure 11.7) appears to be split in two big groups as well, although with slight differences.

The group containing compounds associated to a target is divided, in turn, in two sub-groups, highlighted in the plot shown in Figure 11.6 by contour lines. On the left side we find all the molecules bearing a methyl substituent or other alkyl groups at position 1 of the Bz nucleus (the alkyl groups may be substituted in turn and may show bigger steric hindrance and/or different chemical features). In a central region of the plot we find all the molecules that bear no substituents at position 1. The little sub-group on the right side
Figure 11.7: Principal component plot of training compounds used in the experiment II derived from 30 output values of hidden neurons. Compounds characterized by $R_1=\text{H}$ (left lower side of the plot) and compounds bearing substituent at position 1 (right lower side of the plot) are grouped by contour lines. See Table 11.6 in the Appendix for compound numbering.
of the plot contains compounds characterized by thienyl, instead of the phenyl, for the group A ring of the Bz nucleus.

A similar plot for PCA of the internal representations developed in experiment II is reported in Figure 11.7, where the cluster of compounds bearing substituent at position 1 is located at the left lower corner.

Both the biggest clusters contain molecules divided in turn into smaller homogeneous sub-clusters on the basis of the presence of substituents at the other significant positions of the Bz nucleus previously mentioned.

In Figure 11.8 we observe that each of the two big clusters identified in the previous plots is sub-clustered on the basis of which kind of atom or atomic group is present at position 7. Compounds characterized by the presence of a halogen atom at position 7 are marked by little boxes, while little crosses are used to mark the remaining compounds. The sub-groups so identified only partially overlap; mostly it is possible to find regions of the plot where molecules characterized by one or another kind of substituent prevail. The corresponding plot for results obtained from experiment II is reported in Figure 11.9.

The plots shown in Figure 11.10 and 11.11 allow us to focus the analysis on the pres-
Figure 11.9: An expanded view of circled areas of plot reported in Figure 11.7. Compounds characterized by $R_7 = \text{Halogen}$ are marked by little boxes; compounds where $R_7$ is not a Halogen are marked by times signs.
Figure 11.10: An expanded view of the circled areas of the plot in Figure 11.6. Compounds characterized by \( R_{2'} = \text{halogen} \) are marked by boxes; compounds bearing halogen atoms both at position \( 2' \) and \( 6' \) are marked by plus signs in boxes, and compounds where \( R_{2'} \) and \( R_{6'} \) are not a halogen are marked by times signs. Compounds bearing halogen atoms at positions \( 2' \) or \( 2' \) and \( 6' \) appear to be located at the (left) upper side of each group.
Figure 11.11: An expanded view of circled areas of plot reported in Figure 11.7. Compounds where $\text{R}_2$ = Halogen are marked by little boxes ($\Box$); compounds bearing halogen atoms both at position $2'$ and $6'$ are marked by plus sign in boxes and compounds where $\text{R}_x$ and $\text{R}_y$ is not an Halogen are marked by times signs. Compounds bearing halogen atoms at positions $2'$ or $2'$ and $6'$ appear to be located at the left upper side of each group.
halogen atoms which are deactivating and orto-para directing substituents.

It is noticeable that the differences in analogous plots showing the results obtained from distinct experiments (corresponding to different realizations of the model) only consist of rotations and/or translations of the clusters with respect to each other (e.g. as we can observe from the comparison of plots reported in Figure 11.6 and 11.7), i.e. the molecules are still homogeneously clustered on the basis of the substituent effects.

In this regard it has to be pointed out that the substituent effects on the target molecular properties (e.g. affinity) combine to each other in very complex ways. Nevertheless the well defined clustering observed in most of the PCA plots suggests that each single effect may be easily extracted by the model, in its different realizations corresponding to the six experiments, offering a quite direct analysis of the structure-property relationships. The analogies found in PCA of different experiments appear to be particularly significant. It shows that the capability of the model in extracting structural features, which are significant for the target correlation, is quite independent from the different realizations of the model itself.

11.1.5 Discussion

Regarding the evaluation of the performance of the proposed model for the treatment of benzodiazepines, from the comparison with the results obtained by the traditional equational treatment, we can observe a strong improvement in the fitting of the molecules included both in the training set and in the test set. The experimental results suggest a significant improvement over traditional QSAR techniques. Good results were obtained also for Data set III, where the most poorly predicted compound is the one bearing hydrogen atoms in place of substituents which play an important role in determining affinity. Finally, the soundness of the proposed model was confirmed by the experimental results obtained for Data set IV, where the only compound which showed the maximum variance through the trials contains a Naphthyl group as C ring which never occurs in the training set. This explains the high variance observed in the prediction.

The ability of recursive neural networks to automatically discover useful numerical representations of the input structures at the hidden layer is the key feature of the adaptive solution to the QSAR task. By analyzing these representations through Principal Component Analysis, as expected, we found that the global distribution of molecules and fragments in the plots of the two first principal components reflects the expected capability of the model in detecting homogeneous structural features that can be directly observed on the basis of the molecular morphology. However, the most remarkable aspect is that the distribution reflects its ability in detecting the similar characteristics of the substituents not directly related to the molecular morphology, such as electronic effects produced by halogen atoms. It has to be recalled here that halogen atoms are represented and distinguished, with respect to each other, only by four different (orthogonal) labels, which do not contain any evident information regarding their very homogeneous electronic properties.
The PCA analysis of the Bz data set shows the capability of the RNN model of discovering the relevant structural features just on the basis of the association between the molecular morphology and the target property. In particular we can observe that the characteristics of many substituents affecting the activity of benzodiazepine were correctly recognized by the model, i.e. the numerical code developed by the RCC is effectively related to the qualitative aspect of the QSAR problem. This is a further step towards the assessment of the model as a new tool for the rational design of new molecules.

11.2 QSAR Analysis of 8-Azaadenine Derivates

In this Section, we face the design of novel molecules belonging to the class of adenine analogues (8-azaadenine derivates), which present a widespread potential therapeutic interest, in the new perspective offered by recursive neural networks for Quantitative Structure-Activity Relationships analysis.

After the preliminary step of assessment of the RNN as new approach to the QSAR analysis carried out in the previous sections both by performance comparison with traditional QSPR/QSAR methods and studying the relationship with qualitative aspects of the QSAR problem, we propose an application of our model as prediction tool for new molecules.

The set of compounds, selected by researcher of the Chemical Pharmaceutical Department of Pisa University, are a class of adenine analogues (8-azaadenine derivatives), many of which act as antagonists of the human A1 adenosine receptors. Just as few examples of their widespread potential therapeutic interest, we mention that they may be exploited as potassium sparing diuretics, with kidney-protective properties, or in therapy of degenerative diseases such as the Alzheimer’s one [134]. We expect that the design of new antagonists take a big advantage from the use of predictive tools such as the one supplied by the QSAR models. In fact, a reliable quantitative prediction of the activity before the compound is made is of great interest to reduce the cost of drugs development.

The generality and flexibility of the structured representation allows us to deal with a class of compound characterized by a quite high morphological complexity, e.g. tautomeric forms.

The main aspect of this work is the completion of the QSAR analysis followed by the prediction step for a small theoretical library of compounds. The molecules, which a significant activity had been predicted for, have been subsequently synthesized and biologically essayed. The prediction results are will be compared with the values obtained ‘a posteriori’ through the biological essay. Some qualitative analyses are also conducted for chiral substituent and the tautomeric forms.

11.2.1 The Task: QSAR for 8-Azaadenine

The class of molecules studied for our analysis is made up of 8-azaadenine derivatives. The selected target property was the affinity toward the receptor expressed as binding
constant (Ki) values. The binding constant is usually measured in nanomolar (nM) units, which is a concentration unit. It is a measure of the strength of the binding between the ligands (in our case antagonists) and the receptor that constitutes the target biological macromolecule. Lower Ki values correspond to more active ligands. The strength of the binding is one of the requirements needed to be satisfied in order to obtain a drug exploitable in therapy. It is measured 'in vitro', while the whole molecular properties needed are subsequently measured 'in vivo'. Usually the QSAR studies are focused in 'in vitro' data, since the 'in vivo' data bases are not large enough for QSAR analysis.

We face the QSAR task according to the following steps:

(a) Composition of data set: the 117 molecules, of known Kis, were selected among the ones synthesized and biologically essayed in the Medicinal Chemistry group which we co-operate to (already published compounds). These data set present some novel characteristics: i) chiral compounds are included and the target values refer to the racemic mixture; pure enantiomers are described by the same structural formula, but present specular arrangements of the atoms; the biological activity may differ significantly from each other; ii) eight compounds undergo a tautomeric equilibrium. This means that the morphology of these molecules can be associated to two different structures. In general, we cannot indicate a priori if only one tautomer or both of them, and, if so, in which ratio, are effective on the receptor. In fact the environment surrounding the ligand in its bound state contributes to determine the tautomeric ratio. For each compound undergoing tautomeric equilibrium two different representations were used which the same value of the target property was associated to (see Section 11.2.2). The total number of structures becomes 125.

(b) Training of the RCC model using 91 molecules (plus 5 tautomeric forms) and, in parallel,

(c) the analysis and validation steps, performed with a test set composed by around 20% of the total number of data (26 molecules, plus 3 tautomeric forms).

(d) Design of a small virtual library of new compounds (“prediction set”). In our preliminary attempt this set was built with the following guidelines: i) taking into account new suggestions from qualitative SAR; ii) assaying the predictive power of the neural model at the sampling limits of the training data; iii) maximize the molecular diversity; iv) minimize the development cost for the synthesis laboratories and maximize the rapidity of such synthesis.

(e) We have used the RCC model obtained from the step (b) and (c) to compute the target property (Ki) for data of the prediction set. The compound designed were then synthesized and biologically essayed. We report the synthesis of the quantitative results in Section 11.2.3.

(f) We have drawn some qualitative results on specific compounds belonging to the prediction set, as reported in Section 11.2.3.
11.2.2 Molecular Structure Representation

Basically, the definition of an appropriate function $\tau_R$ for the specific set of molecules follows the line discussed in Section 11.1 aimed to obtain a structured representation of each compound as labeled ordered tree. The shared nucleus of the molecular group (template) is the root of the whole molecular tree. The other molecular fragments are represented at atom-bond level. A specific attention is given to tackle the tautomeric aspect by a proper definition of the shared nucleus and by the multi-representation of such cases. An example of representation for an 8-azaadenine compound is shown in Figure 11.12 for the first tautomeric form; the template is labeled “AZA”. Substituents at positions 2, 6 and 9 are responsible for the diversity of the library. The numbers used to indicate the substituent position of the nucleus are inherited from the IUPAC nomenclature of the endogenous ligand adenosine. For the second tautomeric form the substructure in position 7 and 9 are exchanged. The two tautomeric forms correspond to the same target value.

Concerning the label attached to each node, a bipolar localist representation encoding the types of the chemical objects has been used. For example, the label for the N atom would be something like $[-1,-1,...,-1,1,-1,...,-1,-1]$.

![Figure 11.12](image)

Figure 11.12: Example of an 8-azaadenine (left side) with the template-nucleus (shadowed) and its representation as labeled tree (right side).

11.2.3 Experimental Results

Due to the low number of training data and to avoid overfitting, several expedients were used, as explained in Section 5.4. As a result we could continue learning, adding new hidden unit in RCC, without overtraining the model. No significant increase in the test error was encountered in our experiments. However, the large tolerance in the precision of the target values in the data set (experimental values) and the limitations in the sampling quality of the sub-components of the structures present in the small training set do not allow to expect an optimal generalization performance.

As target output for the networks we used $\log(K_i)$ normalized into the range $[0,1]$. An initial set of preliminary trials was performed in order to determine an admissible range
11.2. QSAR ANALYSIS OF 8-AZADENINE DERIVATES

for the learning parameters. However, no effort was done to optimize these parameters. Five trials were carried out for the simulation using different random initialization for the connection weights.

The main statistics computed over all the simulations for the training set are reported in Table 11.3. Statistics on the number of inserted hidden units are reported, as well as on the expected mean absolute error and the standard deviation of error (S). Note that the mean absolute error and S are obtained by averaging over the single trials; also the minimum and maximum values of the mean absolute error over these five trials are reported. The maximum error is 1%, as fixed by the criteria to stop the learning.

<table>
<thead>
<tr>
<th>#Units Mean(Min-Max)</th>
<th>Mean Error(Min-Max)</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.6 (20-29)</td>
<td>0.0020 (0.0009-0.0028)</td>
<td>0.0026</td>
</tr>
</tbody>
</table>

The results for the corresponding test set are reported in Table 11.4, adding a column for the maximum absolute error. For the test data set we also report on the second row the statistics for the committee model (based on a simply ensemble averaging method) obtained using the mean expected output as computed over the five trials.

<table>
<thead>
<tr>
<th>Model</th>
<th>Mean Error (Min-Max)</th>
<th>Max Error (Min-Max)</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCC</td>
<td>0.1124 (0.0932-0.1230)</td>
<td>0.3006 (0.2512-0.3561)</td>
<td>0.1402</td>
</tr>
<tr>
<td>Committee</td>
<td>0.0650</td>
<td>0.2227</td>
<td>0.1201</td>
</tr>
</tbody>
</table>

Our main interest is on prediction set: the results for four significant compounds are reported in Table 11.5. For each compound we report the target (experimental measured activity), the mean output over the 5 trials (predicted value), the error value, and the correspondent values of the target and the output re-expressed in the original Ki scale.

Note that the differences between the predicted and the experimentally measured activity were below the usually encountered experimental error. Three compounds (number 2, 3, and 4) in Table 11.5 are predicted to be quite active ligand and the experimental results have confirmed this prevision.

We have tested other new compounds to extract qualitative information. Since we enclose in the virtual library some pure enantiomers, although in the training data mostly
Table 11.5: Results Obtained for Each Compound of the Prediction Set by RCC.

<table>
<thead>
<tr>
<th>#</th>
<th>Target</th>
<th>Output</th>
<th>%Error</th>
<th>Target Ki</th>
<th>Output Ki</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.7312</td>
<td>0.7486</td>
<td>-1.74</td>
<td>354</td>
<td>439.4</td>
</tr>
<tr>
<td>2</td>
<td>0.5516</td>
<td>0.5584</td>
<td>-0.67</td>
<td>38</td>
<td>41.3</td>
</tr>
<tr>
<td>3</td>
<td>0.5443</td>
<td>0.5266</td>
<td>1.77</td>
<td>34.7</td>
<td>27.8</td>
</tr>
<tr>
<td>4</td>
<td>0.5054</td>
<td>0.5745</td>
<td>-6.90</td>
<td>21.4</td>
<td>50.4</td>
</tr>
</tbody>
</table>

Racemic mixtures were considered, we tested both enantiomer forms (R and S). In particular the compounds number 1 and 3 in Table 11.5 are pure enantiomers of type R. The correspondent S enantiomers are less active ligands. We found in both cases that the S enantiomers were badly predicted, and this observation suggests that the model has learnt to predict the values of the R enantiomers also from the racemic mixtures sampled in the training data set. This result is compatible with what already known from biological data.

Concerning the evaluation of results, we note a gap between the test and training set accuracy, even though no overtraining is encountered. As mentioned above, high tolerance in the target values and sampling incompleteness can explain these results. However, a simply committee technique, like ensemble averaging method, allows to obtain a mean error of 6.5% for test set data, that is a result at the state of the art in QSAR analysis. Moreover we could obtain very reliable results for prediction set. This suggests us to focus future works on the data set and the theoretical library enlargement in order to obtain a more rational and complete sampling of structure instances.

We have to point out, respect to the accuracy, that the precision, which the binding constant can be experimentally obtained with, is quite low. It affects obviously the quality of the training data. Nevertheless also the precision in the prediction doesn’t need to be high: predictions that can be successfully exploited in drug design may be affected for example by an error up to 10 times the binding constant value, when its value is below 50 nM, or up to 3 times when its value is about 150-200 nM. It fact the need is usually to design ligands with binding constant below few hundred nM. Our results (errors) are largely below the above tolerance needed for a correct classification of active drugs.

With regard to the ability to discriminate between different tautomers shown by the model we observed that both in the training and in the test set the mean error calculated for each group of tautomers has the same order of the mean error calculated for the molecules not undergoing tautomeric equilibrium. Even though the number of enantiomers is limited, we can infer that both tautomeric forms may be active on the receptor.

Summarizing, we can conclude that the good fit with the experimental data and the qualitative results confirm the ability shown by the model of capturing relevant features of the whole molecular morphology.
11.2.4 Discussion

After the preliminary steps of assessment of the recursive approach as new approach to the QSPR/QSAR analysis performed in the previous chapters, the current experiments focus directly at the prediction of new molecules.

The present experiments allow us to test the potentiality of the connectionist approach for the challenge task of a rational design of new drugs by predictive models. Our results are a further step of assessment of the approach aimed to enter deeply in the drug discovery process by the proposed approach.

11.3 Toward a Software System

Practical applications of the model, carried out by the collaboration with experts in various fields, place demand on improving the automatization of the applications process. Moreover, it is required an effort to streamline the process. The experiments we performed lead to us a certain experience in the design of QSPR/QSAR application using RNN models. The recursive neural network (RCC) simulator is a powerful basis to develop these applications. However, the treatment of the application data and results is a relevant task. The basic idea is that a suitable processing of the data and of the results can add great values to the practical usability of the proposed approach and can provide a more powerful way to improve the performance in specific applications. On these bases, we started to develop a software system aimed at reducing the burden of creating a complete cycle of prediction from scratch.

In particular, the flexibility shown by the proposed model to face heterogeneous data and problems gave us the opportunity to extended the applications to complex problems considering more and more large data sets. A suitable software tool can guide the applications and it can allow for fitting the general methodology to the specific characteristics of the task at hand.

**The design cycle.** The design of a predictive system usually entails the repetition of a number of different activities: data collection, representation and preprocessing of data, model training, postprocessing and evaluation of the results. The process is a cycle since its phases may call for repetition of some process steps to obtain satisfactory results. Specifically, dealing with SD poses new problems that standard tools designed for flat domains cannot handle.

The application process can be roughly represented in Figure 11.13, where the components of the whole cycle are represented in a simple chain of activities. The central component is the model simulator based, for our purpose, on the RCC model. The simulator is fed by structured data and target information to build the model. The trained model is the predictive model applicable to new data.

In the QSPR/QSAR framework, the data sets are composed by structured representation of molecules. The first element of the design process should be a Data Base Manage-
The facilitation of the input of data is the relevant goal of this phase. From the information stored in the DB, sets of structured data can be managed allowing the researcher to assess the amount of molecular structure that needs to be explicitly described.

The preprocessing phase terminates creating the data sets that will be used as input data to the model simulator. The preprocessing phase also should include statistical and graphical tools that allow preliminary investigation of the data under analysis. The problem of conversion among data formats can also be considered. In fact, the possibility to manage various data formats is the key element to enable the access to preexisting chemical DB, the exchange of data with preexisting QSPR/QSPR software, and the creation of benchmarks based on standard formats.

Finally, the postprocessing step should allow to properly integrating and visualizing the results provided by the simulator. Task to be considered are the performance evaluation, including the Cross Validation technique, and the realization of model committees. Statistical tools and graphical interfaces are entailed. The tool should allow us to carry out quantitative and qualitative analysis. Such results should be interpretable by the QSPR/QSAR experts in order to evaluate the performances and to discover new knowledge on the domain.

An integrated tool. The first step toward the realization of a QSPR/QSAR prediction system based on recursive neural networks has been the software tool SPRINT (Structures PRocessing INtegrated Tool). The system is fully described in [13]. The major aim has been to provide the RCC simulator with tools that implement the components of the design cycle described above and to integrate them in a support system. We briefly present in the following the characteristics of such system. Although most of the procedures entailed in the design cycle described above have been realized, we focus on discussing some specific aspects that characterize our approach.

The first point concerns data. The preparation of data is a compulsory, time consuming, labor intensive step of the ML applications. Usually, this phase accounts for large
part of the cost of developing a predictive system. The need to cope with SD exacerbates the problem. Although different tools exist to manage DM applications, the same is not true for SD learning. This is one of the main reasons to develop our specific tool.

Hence, the first task of the system is to facilitate the collection and the transformation of data. Various solutions are implemented in the proposed system.

A DBMS is used to collect the data. Our previously available data sets, based on the “p-rep” representation presented in Section 3.1 are accessible in their original text format. A parser tool allows to check the correctness of the data and it helps in reducing the input errors.

An easier way to collect data is based on graphical insertion of structures in the database by the composition of predefined fragments. A graphical interface facilitates the direct creation and control of the 2D structure of the molecules. The user is guided in the composition of new structures by allowing the insertion of predefined structure fragments.

The compositionality is a specific characteristic of the system. The compositionality allows the definition of sub-structures that can be used in the creation of complex structured objects. As a result, the practical insertion of new structures is highly facilitated and accelerated. Moreover, through the possibility to define the granularity of the information it is easy for the designer to decide the information content of each vertex of the structure.

In fact, a set of rules can be defined to automatically transform the structure in the database by means of the substitution of a single vertex with a sub-structure or, on the opposite, of a sub-structure into a single vertex. The rules can be combined in a set. Through the use of the transformation rules the designer can decide the level of information conveyed by the structures. This allows to automatize the managing of one important aspects of the representation problematics introduced in Section 9.4 concerning the $\tau_R$ function. In other words these tools support the exploiting of the flexibility offered by a structured representation.

A simple example of rule that can be specified is the following:

$$A(B,B,*) \rightarrow C(*)$$

where we represent trees in the p-rep form (see Section 3.1) and we use the symbol ‘*’ to represent any structure that can occur in that position. All the instances of structure, containing substructures that match the substructure on the left side of the rule, will be transformed by the rule application.

The inverse rule can be given as:

$$C(*) \rightarrow A(B,B,*)$$

In chemical setting this type of rules can be used to aggregate chemical substructure into a single group or to expand a group into its explicit structure, e.g.:

ethyl-group $\leftrightarrow$ CH$_2$-CH$_3$
Another type of transformation allows the aggregation of different elements into a group that represents an abstract semantic related to the specific task, e.g. the atoms Cl, Br and F can be grouped as “alogen” atoms.

Globally, by the use of these simple rules the management of structures results simplified and automatized.

The problem of the structure granularity can be considered from different points of view.

As introduced in Section 9.4, in QSPR/QSAR applications, each chemical compound (or fragment) can be described at different levels of structural detail. At the maximum detail level the compound (or fragment) is represented by the graph of atoms and bonds. In some cases, the designer can recognize a group of atoms, e.g. representing a functional group, which can be represented by a single element (vertex) of the structure. This may be useful both to reduce the complexity of the input pattern and to encode specific knowledge on the domain. The flexibility of the structured representation allows us to consider different levels of such granularity, and the proposed rule method allows the transformation of the structure according to the designer specifications.

This is also a way to consider hybrid approaches to extract structured features (usually performed to obtain flat representation of the SD) retaining a structured representation of the relationships among such features. From this point of view, we offer a tool to specify a level of granularity that can best fit the task at hand.

From the point of view of the background knowledge, the possibility to control the amount of molecular structure that is explicitly described allows the designer to encode information known for the problem at hand (e.g. the chemical functionality of a chemical fragment). Specifically, the transformation rules allow to consider background knowledge in the form of aggregation of several facts into a single fact.

The preprocessing procedures provide also tools for generating the data set, i.e the splitting of data to generate training and test sets and for the Cross Validations methods. Statistics include the analysis of the frequencies of fragments and sub-structure in the data set, query on data base, and plots of target distribution.

Postprocessing procedures allow to compute simple statistics and to obtain graphical representations of the simulation results.

In particular, for instance, the results are composed to realize simple averaging committees of different models. The graphical visualization of the PCA plots can be used to investigate the internal representations developed by the recursive neural networks (encoding space) (see for example Section 11.1.4).

The system is written in Java (100% pure Java). Using Java the system can be used for a wide range of computer platforms allowing the designer to use the preferred one. The system provides a uniform Human-Computer graphical interface for preprocessing and postprocessing. The DBMS mechanism allows remote connection to the information. The database is based on MySql, freeware software, which provides a well-known and efficient DBMS.
The major aspect of the system is the integration of the different procedures in the same tool, so to create a development environment that would help the design of QSPR and QSAR applications. The system is currently evaluated by the research partners. In a more general perspective, the project represents a first step toward the complete implementation of the SD-learning cycle through software tools.

Appendix

In the following the training set for benzodiazepines data used in data set III are reported. We report in the tables the numbers associated to compounds (not their fragments) as used in figures 11.6, 11.7, 11.8, 11.9, 11.10, 11.11.

Note that the C ring, located at position 5, is a phenyl group in all the analyzed compounds except in compounds 47, 108, 109, 111 and 113 where it is replaced by 2-pyridyl, Cyclohexenyl, Cyclohexenyl, Cyclohexyl and Naphthyl, respectively (marked by * in Table 11.6).
<table>
<thead>
<tr>
<th>#</th>
<th>R1</th>
<th>R3/R6</th>
<th>R7</th>
<th>R8/R9</th>
<th>R2</th>
<th>R6</th>
<th>Log 1/C</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>-CH3</td>
<td>-CN</td>
<td>-F</td>
<td>-F</td>
<td>7.52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-CH3</td>
<td>-CH2</td>
<td></td>
<td>-F</td>
<td>7.68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-CH3</td>
<td>-CN</td>
<td>-F</td>
<td>7.74</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>-CH3</td>
<td>-CN</td>
<td>-F</td>
<td>7.90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>8.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>8.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>-N2</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>8.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>-NO2</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>8.43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>-CH3</td>
<td>-I</td>
<td>-F</td>
<td>8.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>-CH3</td>
<td>-Br</td>
<td>-F</td>
<td>8.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>8.70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>8.72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>8.74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>8.82</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>9.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-F</td>
<td>8.82</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-F</td>
<td>9.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-F</td>
<td>9.15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>9.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>9.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>9.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-Cl</td>
<td>9.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>-Cl</td>
<td>-Cl</td>
<td>-F</td>
<td>9.75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>9.92</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>9.93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>9.93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>9.93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>-CH3</td>
<td>-Cl</td>
<td>-F</td>
<td>9.93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>-CH3</td>
<td>-CN</td>
<td></td>
<td>6.84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>-CH3</td>
<td>-CN</td>
<td></td>
<td>6.84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>-CH3</td>
<td>-CN</td>
<td></td>
<td>6.84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>-CH3</td>
<td>-NO</td>
<td></td>
<td>7.02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>-CH3</td>
<td>-NO</td>
<td></td>
<td>7.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>-CH3</td>
<td>-NO</td>
<td></td>
<td>7.48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>-CH3</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>-Cl</td>
<td>-Cl</td>
<td></td>
<td>7.54</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 12

Conclusions and Future Works

There is a common line underpinning the methods studied in this thesis for learning in structured domains. The basic idea is to recursively compose the codes computed for each vertex of a given structure. Thankful to this idea it was possible to build encoding models that are isomorphic to the input data structure. The recursive control of the information flow can be combined with the use of machine learning techniques. The presence of free parameters of the model, easily realized by neural networks, allowed us to create adaptive models that can be used to implement learning in structured domains. This basic idea of recursivity is very natural in Computer Science. Moreover, it has been explored for a long time to study neural computing models able to generate implicit representation of time or other serial phenomena (recurrent neural networks) and then extended to hierarchical structure processing (recursive neural networks).

However, the learning of structure transductions can be a very difficult task. The solution must cope with the richness of information conveyed by structured data and their variability. We have shown through the dissertation how the property of the approach, concerning the form of computation, the parsimony (model complexity), combined with the characteristics of the neural computing methods, make the RNN a suitable tool to deal with SD for ML tasks.

The main advantage of the proposed approach is strictly related to the learning capability of the models that realize the processing system. In fact, by using the proposed adaptive processing scheme (adaptive transduction), the specific encoding procedure is learned on the basis of the training data. Specifically, the proposed approach allows the extraction of the topological information of structures directly from the examples of the computational problem at hand.

The exploitation of such basic idea lead to a variety of research topics aimed at investigating the models capability and the real performances in applications. Throughout the thesis new models, theoretical properties and applications have been studied to explore the potentialities of the recursive approach in ML framework.

In particular, the basic strategy relies on some strong assumptions. One of these assumptions is the assumption of causality. We have shown that this constraint can be partially relaxed extending the computational capability of the models to contextual pro-
The new CRCC model, by exploiting a contextual flow of information, provides an innovative encoding strategy with a theoretical basis that allows to understand the information that the model can extract from data: the shape of the context in CRCC is expressed as a function of the state variables in the model, and of topological characteristics of the input structures. We have proposed a compact formulation of such context information, which allows a direct comparison with respect to causal models. Then, we have formally shown that the new model can compute some kind of transductions that cannot be computed by pure causal models. Although based on specific hypotheses, such investigation provided new insights in the contextual processing of structured domains.

Another research topic concerns the transfer of unsupervised learning theory to the recursive approach. To make it possible we proposed a unifying framework both for the various recursive SOM variants and for their further extensions to structured data. This comprehensive theoretical framework is based on the recursive dynamics idea plus a few simple “ingredients” that can be instanced for different models. Moreover the properties of learning and the representation capabilities of various models have been investigated. Specifically, results on relationships between Hebbian learning and stochastic gradients methods were drawn: for structured domains, Hebbian learning does not in general correspond to gradient descent, but it can be seen as an efficient approximation.

We finally dealt with the assessments of the approach in real-world tasks with a double aim. First, we showed, through empirical investigation, that the adaptive recursive approach can be effective in real-world applications. Second, we exploited the functional transduction framework to develop a novel methodology to deal with QSPR/QSAR analysis based on the direct and adaptive processing of the structure of chemical compounds. The input of the RNN directly consists of a structured representation of the molecules; the encoding and the regression functions are learned together by the model. Hence, the proposed models are able to learn a map between the input structured domain and the activity/property output space discovering numerical codes for the chemical structures which are optimized with respect to the prediction task.

This allowed us to overcome a relevant drawback of the traditional approaches: in the new approach we can relax the need for a prior selection of features, which, in traditional approaches, need to be devised and optimized through a trial and error procedure by experts in the fields of application.

Applying the method to a set of radically different QSPR/QSAR problems, we have demonstrated that the application of recursive neural networks to QSPR/QSAR tasks allows the treatment of different computational tasks by using the same basic approach.

The generality is not to be paid in terms of predictive accuracy. The achieved results are at the state-of-art in different tasks, and they are competitive or improved with respect to ’ad hoc’ methods. This evaluation is further supported by the qualitative results on the analysis of the internal representation developed by the models: in our QSAR experiments for benzodiazepines, where the relevance of the molecular features are already known, we found that RNNs can capture significant topological aspects and chemical functionalities of the compounds just on the basis of the association between the molecular morphology and the target property. Hence, the code developed by the model is effectively related to...
the qualitative aspects of the QSAR problem.

Both the performance and qualitative results allow us to conclude that, although the method presented here is at an early stage of its development, the recursive neural network models are able to supply a well-suited tool for QSPR/QSAR analysis.

More generally, the presented applications can be seen as a paradigmatic and useful analysis of the wider problem of processing structured domains in Machine Learning.

A further advancement of the application objectives has been approached through the development of an integrated set of software tools supporting the QSPR/QSAR analysis based on RCC model. The focus was on the representation phase, with the aim of exploiting the abstract compositional descriptions allowed by a structured representation of the chemical compounds.

Summarizing, this investigation of the recursive approach to SD in ML represents a critical contribute that would stimulate further research toward new more powerful approaches. The adaptive recursive method provides a flexible method for solving SD problems characterized by a suitable balancing between generality and prediction performance. We have shown that the recursive method allows the propositional learner to deal with SD by providing a hypothesis space able to represent relationships among data, an apt learning algorithm, and a method to deal efficiently with structure variability. Thus, although some specific problems can be solved by different approaches with high performance results, we have at least shown that the recursive idea provides elegant and quite general solutions to SD processing in Machine Learning.

**Future Works**  

According to our general aim, i.e. the investigation of the Structured Domain Learning, there are several possible suggestions for new research topics.

Firstly we consider possible developments of the methodology.

CRCC (Contextual RCC) is a new topic that deserves further research. For example, we plan to study the extension and generalization of the approach up to consider a framework for the contextual processing of SD that can, for example, lead beyond the Cascade Correlation family of models.

The general framework for unsupervised learning of SD can be further exploited to study basic properties and to introduce and to implement new models in a uniform setting: the extension to structured data of various self-organizing map variants is certainly an interesting project for further research on concrete ML applications.

A different topic concerning analysis, not introduced up to now in this thesis, concerns the study of the basic properties of the transition function of the recursive models from a dynamical system theory viewpoint. Iterated Function Systems, a special type of dynamical system responsible for the creation of fractal images, can be used to explain the behavior of parametric SD-Recursive processing system (prior to learning). For instance, the study of fractal coding for SD can extend the analysis conducted in [159, 160] for sequence domains where interesting analogies with variable memory length Markov models (VLMM) have been analyzed.
As outlined in Chapter 8 a major source of suggestions arises from the comparison of different approaches to structured domain learning. Major emphasis was given, in particular, to the compared study of recursive neural network approach versus ILP and related Relational Data Mining approaches. Among the current cutting-edge methodologies, the model based on kernels/SVM seems to play a major role: we plan to analyze the kernel machines, with kernel designed for structured data, with respect to recursive neural networks (e.g. by empirical comparison on specific tasks) and to combine, as possible, the advantages of the two approaches. A very simple idea to combine recursive models and SVMs is to use a SVM to perform the post-processing of the internal representations of recursive neural networks; i.e. the output function $g()$ of the recursive neural network (e.g. a RCC) can be realized by a SVM.

Another interesting topic aimed at improving the predictive performance, concerns the design of systems exploiting committee and ensemble techniques.

The ultimate aim should be the design of new hybrid methods and learning algorithms for SD processing. The combination of different approaches, in the light of a neural computing basis, can be a suitable solution to the problem of integration of symbolic and sub-symbolic learning methods, i.e. connectionist-symbolic integration.

Concerning the applications, it is worth to note that there exists a strong demand for screening of chemical and biological data by ML techniques. Hence, both improvements of the current methodology and of the tools to support the applications (pre and post processing) are needed.

The scope of these applications can be further expanded and exploited for various problems in the area of Cheminformatics and Bioinformatics. In fact, new orders of drug discovery are emerging from the methodologies of managing and analyzing biological, clinical and chemical data.

Among the various possibilities we can briefly mention some ongoing researches.

We plan to extend the applications developed so far considering the extension of the classes of compounds both for QSPR (e.g. for the thermodynamical properties prediction task) and QSAR tasks (e.g. extending the database of A1 and A3 adenosine receptor ligands, containing the affinity data for both receptor types).

The prediction of chemical, physical, and biological activity of monomeric and polymeric chemical compounds is another challenging interdisciplinary research objective. The generality and flexibility of the structured representation allow us to analyze this class of molecules looking at the 2D graph of the monomers that constitutes the polymers. Such studies can be exploited in the study of polymers for tissue engineering applications.

Finally, the flexibility of the approach can also be exploited to capture relevant information such as topological or functional description of the data that can characterize various tasks in the Bioinformatics area. The evaluation of pharmacokinetic properties, for instance regarding individualized evaluation of drug effects, should consider the integration of the different Cheminformatics and Bioinformatics approaches to drug discovery.


