Universidade Federal de Pernambuco
Centro de Informática

Pós-graduação em Ciência da Computação

NEURAL NETWORKS FORECASTING AND CLASSIFICATION-BASED TECHNIQUES FOR NOVELTY DETECTION IN TIME SERIES

Adriano Lorena Inácio de Oliveira

TESE DE DOUTORADO

Recife
December, 2004

Universidade Federal de Pernambuco
Centro de Informática

Adriano Lorena Inácio de Oliveira

NEURAL NETWORKS FORECASTING AND CLASSIFICATION-BASED TECHNIQUES FOR NOVELTY DETECTION IN TIME SERIES

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutor em Ciência da Computação.

Orientador: *Prof. Silvio Romero de Lemos Meira, PhD*

Recife
December, 2004

AGRADECIMENTOS

Agradeço a Deus pela saúde, inspiração e motivação ao longo do trabalho. Agradeço também às diversas pessoas que apoiaram direta ou indiretamente o desenvolvimento deste trabalho, seja no âmbito acadêmico ou pessoal.

Pelo apoio pessoal mais importante e próximo em todos os momentos da realização deste trabalho, um agradecimento especial à minha esposa, Malu Buarque.

Na Academia, meus agradecimentos vão, particularmente:

- Ao Prof. Silvio Meira, pela orientação, apoio e entusiasmo em relação ao trabalho em todas as suas fases;
- Aos membros da banca, que avaliaram e aprovaram esta tese, Professores Geber Ramalho (CIn-UFPE), Aluízio Araújo (CIn-UFPE), André Ponce de Leon (ICMC-USP), Antônio Braga (DELT-UFMG) e Paulo Adeodato (CIn-UFPE), pelas críticas construtivas e sugestões para melhorias na tese;
- Aos meus alunos de Iniciação Científica do Departamento de Sistemas Computacionais da Escola Politécnica da Universidade de Pernambuco (DSC/POLI/UPE), especialmente aos que contribuíram diretamente em parte dos trabalhos desenvolvidos nesta tese: Bruno Melo, Adélia Barros e Gabriel Azevedo;
- Aos professores do Centro de Informática da UFPE (CIn-UFPE), especialmente àqueles com os quais fiz disciplinas da área de Computação Inteligente: Flávia Barros, Francisco Carvalho, Geber Ramalho e Teresa Ludermir;
- Aos professores Fábio Silva e André Santos (CIn-UFPE), por terem participado como orientador e co-orientador no início dos trabalhos.

Agradeço também aos diversos familiares e amigos que apoiaram a realização deste trabalho. Agradecimentos especiais:

- Aos meus pais Paulo Afonso e Maria do Rosário;
- Aos meus irmãos, Poliana, Paulo Henrique e Eugênio Saulo;
- Aos Professores do Departamento de Sistemas Computacionais da Escola Politécnica da Universidade de Pernambuco (DSC/POLI/UPE), especialmente aos pioneiros naquele departamento junto comigo, Professores Carlos Alexandre Mello e Ricardo Massa;

- À direção da Escola Politécnica da Universidade de Pernambuco (POLI/UPE), em particular ao Diretor, Prof. Carlos Calado, pelo apoio às pesquisas realizadas durante este trabalho, especialmente em relação à participação em conferências internacionais.

- Aos amigos do Tribunal de Contas do Estado de Pernambuco, Dora Albuquerque, Eduardo Nevares, Jorge Bandeira, Juliano Cavalcanti, Gustavo Tibério, Moisés Zarzar, Ricardo Beltrão e Waldyr Affonso;

- A todos os colegas e amigos da Coordenadoria Tecnologia da Informação (CTI) do Tribunal de Contas do Estado de Pernambuco. Em particular à atual Coordenadora, Maria Teresa Moura e à gerente da Gerência de Auditoria de Tecnologia da Informação (GATI), Regina Ximenes.

ABSTRACT

Novelty detection can be defined as the identification of new or unknown data that a machine learning system is not aware during training. Novelty detection algorithms are designed to classify input patterns as *normal* or *novelty*. These algorithms are used in several areas such as computer vision, machine fault detection, network security and fraud detection.

The behavior of many systems can be modeled by time series. Recently, the problem of detecting novelties in time series has received great attention, with a number of different techniques being proposed and studied, including techniques based on time series forecasting with neural networks and on classification of time series windows. Forecasting-based time series novelty detection has been criticized because of the not so good performance. Moreover, the small amount of data available in short time series makes forecasting an even harder problem. This is the case of some important auditing problems such as accountancy auditing and payroll auditing. Alternatively, a number of classification-based methods have been recently proposed for novelty detection in time series, including methods based on artificial immune system, wavelets and one-class support vector machines.

This thesis proposes a number of neural networks methods for novelty detection in time series. The methods proposed here were developed aiming to detect more subtle novelties that can be of great concern in fraud detection in financial systems. The first method aims to enhance forecasting-based novelty detection by using *robust confidence intervals*. These intervals are used to overcome the main limitation of forecasting-based novelty detection, namely, the selection of a suitable threshold for detecting a novelty. The proposed method was applied to some real-world time series with good results.

Two methods based on classification are also proposed in this thesis. The first method is based on negative samples whereas the second method is based on RBF-DDA neural networks and does not need negative samples for training. The simulation results on a number of real-world time series show that the RBF-DDA based method outperforms the negative samples one. Moreover, the classification performance of the RBF-DDA method does not depend on the test set size whereas the performance of the negative samples method depends on the size of the test set.

In addition to the novelty detection methods, this thesis proposes four different methods for improving the generalization performance of RBF-DDA neural networks. The proposed methods are evaluated using six benchmark classification datasets and the results show that they considerably improve RBF-DDA performance and also that they outperform MLPs and AdaBoost and achieve results similar to k-NN. These methods were also used in conjunction with the method for novelty detection in time series based on negative samples and the results show that they are also valuable for improving performance of novelty detection in time series, which is the main subject of this thesis.

PUBLICATIONS ARISING FROM THIS WORK

ARTICLES PUBLISHED IN JOURNALS

- Oliveira, A.L.I., Melo, B.J.M., and Meira, S.R.L. (2005) *Integrated Method for Constructive Training of Radial Basis Functions Networks.* IEE Electronics Letters, In Press.

- Oliveira, A.L.I., Melo, B.J.M., and Meira, S.R.L. (2005) *Improving Constructive Training of RBF Networks through Selective Pruning and Model Selection.* Neurocomputing, Volume 64C, pp. 537-541.

FULL PAPERS PUBLISHED IN CONFERENCES PROCEEDINGS

- Oliveira, A.L.I., Melo, B.J.M., Neto, F.B.L., and Meira, S.R.L. (2004) *Combining data reduction and parameter selection for improving RBF-DDA performance.* Lecture Notes in Computer Science, Vol. 3315, pp. 778-787, Springer-Verlag. (Proc. of IX Ibero American Conference on Artificial Intelligence (IBERAMIA'2004))

- Oliveira, A.L.I., Neto, F.B.L., and Meira, S.R.L. (2004). *Novelty detection in time series by neural networks forecasting with robust confidence intervals.* Proc. of Brazilian Symposium on Neural Networks (SBRN'2004), São Luis, Maranhão, IEEE Computer Society Press.

- Oliveira, A.L.I., Neto, F.B.L., and Meira, S.R.L. (2004). *Improving novelty detection in short time series through RBF-DDA parameter adjustment.* International Joint Conference on Neural Networks (IJCNN'2004), Budapest, Hungary, IEEE Press.

- Oliveira, A.L.I., Neto, F.B.L., and Meira, S.R.L. (2004). *Improving RBF-DDA performance on optical character recognition through parameter selection.* 17th International Conference on Pattern Recognition (ICPR'2004), Cambridge, UK, IEEE Computer Society Press, Vol. 4, pp. 625-628.

- Oliveira, A.L.I., Neto, F.B.L., and Meira, S.R.L. (2004). *Combining MLP and RBF neural networks for novelty detection in short time series.* Lecture Notes in Computer Science, Vol. 2972, pp. 844-853, Springer-Verlag. (Mexican International Conference on Artificial Intelligence (MICAI'2004)).

- Oliveira, A.L.I., Neto, F.B.L., and Meira, S.R.L. (2004). *A method based on RBF-DDA neural networks for improving novelty detection in time series.* Proc. of the 17th International FLAIRS Conference, AAAI Press. Miami Beach, Florida, USA.

- Oliveira, A.L.I., Neto, F.B.L., and Meira, S.R.L. (2003). *Novelty detection for short time series with neural networks*. The Third International Conference on Hybrid Intelligent Systems (HIS'03), Melbourne, Australia. In A. Abraham, M. Köppen, and K. Franke, editors, *Design and Application of Hybrid Intelligent Systems*, volume 104 of *Frontiers in Artificial Intelligence and Applications*, pp. 66-76, IOS Press.

- Oliveira, A.L.I., Azevedo, G., Barros, A. and Santos, A.L.M. (2003). *A neural network based system for payroll audit support* (in portuguese). In Proc. of the IV Brazilian National Artificial Intelligence Meeting (ENIA'2003).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This thesis reports on the investigation of methods for the effective detection of novelties in time series. Time series data appear in virtually every area of human knowledge. Examples of time series include electrocardiograms, daily values of stocks and monthly sales of companies. Novelties in time series can be informally defined as unexpected values or sequences of values. Applications of time series novelty detection include the detection of failures in a machine and the detection of frauds in a financial system. The methods proposed in this work were designed especially for the latter application.

In classification problems, novelty detection is the ability of the classifier to reject patterns that do not belong to any of the classes of the training set [MS03a, MS03b]. This is a very important issue in many practical problems such as scene analysis [SM04] and intrusion detection in computer systems [RLM98]. In scene analysis with neural networks, for example, it is important to design the network to recognize images that cannot be classified into none of the classes of the patterns available during training. Popular neural networks architectures such as multilayer perceptrons (MLPs) do not have this inherent ability and therefore additional mechanisms must be designed to make these networks detect novelties properly [MS03a, MS03b, Vas95, VFB95]. A number of approaches for novelty detection in classification problems have recently appeared in the literature. Among these methods, some are based on statistical methods [MS03a], others on artificial neural networks [MS03b, Mar03], and others on support vector machines (SVMs) [SWS$^+$00, STC04].

Time series forecasting, the prediction of future values of a given time series based on its previous values and, maybe, other variables, is the most common type of time series analysis. Nevertheless, the detection of novelties in time series has gained recent interest with a number of methods proposed in the literature [MP03, ONM04e, ONM04b, ONM04d, ONM04a, ONM03, OABS03, GD02, GDK02, STZ00, KLC02]. The problem of novelty detection in time series is referred under various names in the literature including anomaly detection [GD02, GDK02] and surprise detection [KLC02].

The applications of time series novelty detection techniques are very important. One example is the detection of faults in machines. In this case, the dynamic behavior of a machine is modeled by a signal that is, in fact, a time series. This is the case for example for mill machines [DF96]. A novelty detection system would learn the normal behavior of the machine through the time series. Later, the system would be able to detect novelties which, in this case, would mean machine breakage [DF96].

Fraud detection in financial systems is another important application of time series novelty detection. Examples of this application include the detection of frauds in payroll systems [OABS03] and in accounting systems [Kos00, Kos03]. The monthly earnings in a payroll and the accounts in an accounting system are time series. In a fraud detection

application, the normal behavior of these series would be learned by a novelty detection system. Later, the system would be used to detect abnormal behavior which, in this case, could mean fraud.

1.1 MOTIVATION

The work reported on this thesis started with a real-world application in mind: the automatic detection of frauds in payroll systems [OABS03]. This is a very important application because personnel represent a significant portion of the total budget of an organization, be it public or private. In Brazil, the government spends about 50% of its revenue with personnel. Auditing of these expenses is one the responsabilities of Brazilian States and federal Court of Accounts. These organizations give special attention to this kind of auditing due to its financial importance.

Organizations usually employ payroll information systems to manage their personnel payment. These information systems are responsible for payroll calculation, data management and reports generation. Several *earnings* and *deductions* such as base salary, vacation, overtime and health insurance, are taken into account in the calculation of the net income of an employee. In Brazil, normally there are hundreds of different types of earnings and deductions in government payrolls.

Fraud detection in payrolls is the aim of a specialized team of auditors. They audit payrolls by analyzing data stored in payroll information systems. In such audits, auditors take into account a set of laws that regulate inclusion or exclusion of earnings and deductions to employees and the formulas used to calculate the payroll. Payroll auditing becomes complex due to the increasing volume of data to be analyzed and of legal regulations to be considered. In practice, in most works it is not possible to make all auditing tests. Consequently, techniques for selecting the most important cases to test should be applied.

The work is carried out by auditors through the use of rules that they build based on the laws that regulate payrolls. There are some software tools that help auditors in theirs tasks such as ACL [acl] and IDEA [ide] tools. These tools are limited because in fact they only help in the development of database queries.

A possible improvement for audit software tools would be the use of *intelligent systems* that would learn a set of rules based on previous frauds. The problem with this approach is that it is difficult to identify previously all the possible types of frauds that can take place. Moreover, new types of frauds can be created during the use of the system and therefore these auditing software tools would not be able to detect them. Those difficulties also arise in intrusion detection systems, which are used to detect attacks in computer networks [RLM98].

The methods introduced in this thesis can be used to help building automatic fraud detectors that can be applied, for example, in payroll auditing. Instead of being based on rules, a system based on the methods proposed here would learn the normal behavior of each earning and deduction of a payroll. Based on that, it could be used to detect abnormal behavior which, in this case, would be considered fraud. The same idea could be applied to audit systems whose behaviors can be modeled by one or more time series,

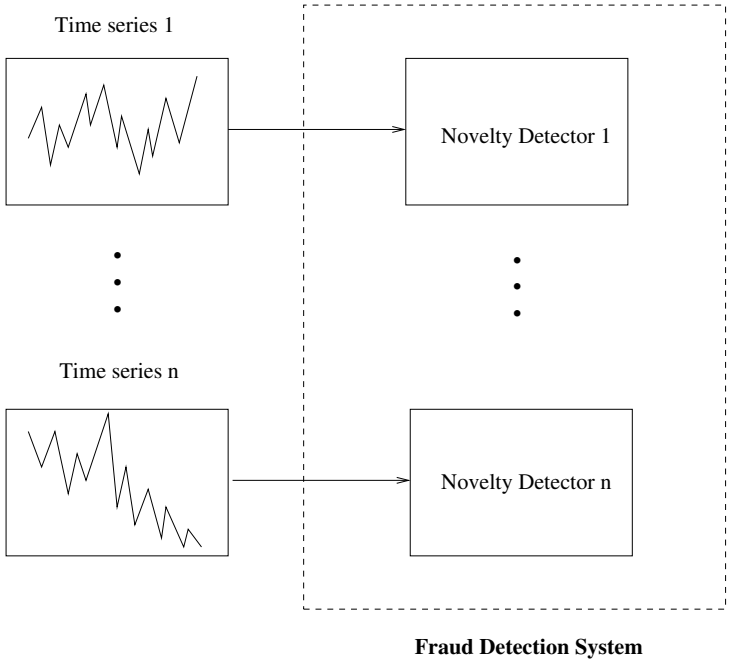


Figure 1.1. Intelligent system for fraud detection in financial systems: training phase.

for example, accounting systems [Kos00]. The advantage of systems based on this idea is that they would be able to detect any kind of fraud. As an alternative to enhance performance of audit tools, the system based on time series novelty detection could be combined with the more traditional rule based systems.

Another example of application for the methods proposed in this thesis would be an intelligent fraud detector system for electrical power consumption. In this case, the system would be fed by time series representing the monthly eletrical power consumption of individual customers. The system should firstly be trained by using historical data free of frauds. One novelty detector should be used for each customer. Subsequently, the system would be used to identify customers whose behavior more likely represented deviations from normality. A specialized auditor would then further investigate those customers in order to verify whether the fraud indeed took place.

An architecture for an intelligent system built using the methods proposed in this thesis is depicted in figures 1.1 and 1.2. Figure 1.1 illustrates the training phase, whereby each time series of the financial system under analysis is fed to a different novelty detector. Each time series is assumed to be free of frauds and therefore each novelty detector learns to recognize the normal behavior of the respective time series in the training phase. In other words, we assume that each time series has been analyzed before and that they represent the normal behavior.

Figure 1.2 illustrates the intelligent system for fraud detection in financial systems in operation. After the training phase the system can be put in operation. Suppose that the system is used to detect frauds in payroll systems. In this case, each earning and deduction is represent by a monthly time series and in many cases there are hundreds of

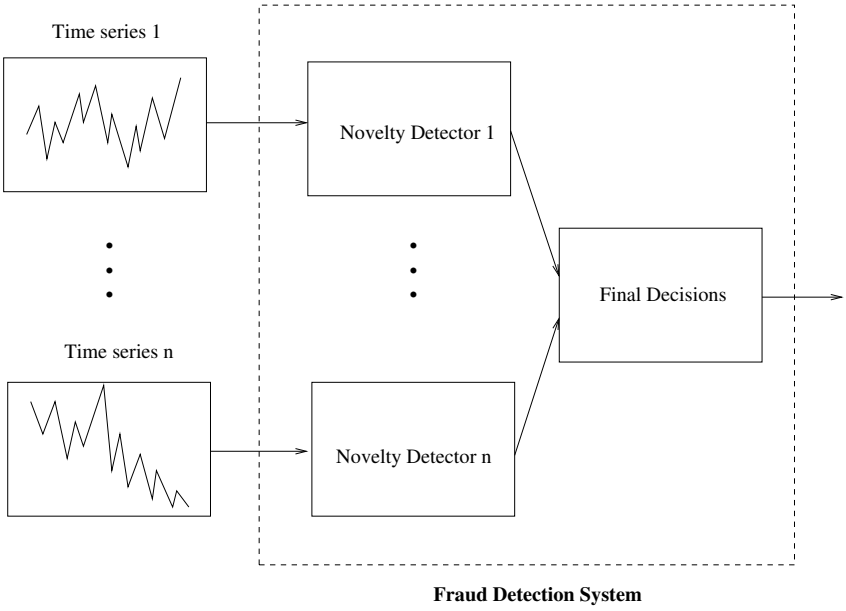


Figure 1.2. Intelligent system for fraud detection in financial systems: system in operation.

them in a company or organization. Furthermore, the company or organization can have hundreds or even thousands of employees.

The system described here could be used to analyze earnings or deductions per employee or the total value of the earnings and deductions considering all employees of the company. In both cases, after the training phase the system would be fed with new values of each time series to be analyzed. In many practical cases, auditing takes place annually. In this case, the final decision provided by the system would consist of a list of months and earnings (or deductions) whose behavior is more likely to represent fraud. Using this information, an auditor would focus his investigations in these cases.

1.2 STATE OF THE ART

Novelty detection can be defined as the process of finding novel patterns in data [SM04, GDK02]. These patterns represent truly new events and cannot be classified into one of the classes defined from previous examples. Novelty detection systems must be able to classify patterns as *normal* or *novelty*. However, in many practical cases there is only normal patterns available to train the classifier, so it is important to design novelty detection systems that overcome this limitation.

The first research efforts on novelty detection were related to classification problems. A number of techniques have been developed for these problems based on statistical pattern recognition or neural networks. Recently, reviews of these techniques have appeared in the literature [MS03a, MS03b, Mar03]. Different neural networks architectures and methods have been used for novelty detection including multilayer perceptrons (MLPs) [SM04, Bis94, Vas95, VFB95, VFB94], radial basis function networks (RBFNs) [FRTT94, ABK$^+$00, Vas95], self-organizing maps (SOM) [EKT00, LV02, MSN02, Mar01],

support vector machines [TD01, DI02], Hopfield networks [BBGC99, CH01], etc.

Early approaches for novelty detection in classification tasks were based on the *negative training approach* [Vas95]. The idea of this technique is to artificially generate a number of patterns, referred as *negative samples*, to represent novelties in the training phase. These patterns are added to the training set to form an augmented training set. Subsequently, a classifier is trained with the augmented training set. This technique has obtained reasonable performance in some cases, especially when the negative samples were carefully selected [Vas95]. However, it has been shown that it may fail because it is unrealistic to expect that randomly selected patterns will accurately represent the input space where novel patterns will fall [Vas95]. This has motivated the development of techniques that do not depend on negative samples.

Radial basis function networks (RBFNs) are a natural candidate for novelty detection without negative training, because the processing units of these networks compute localized functions of the pattern space and therefore patterns falling outside of the localized regions created by an RBFN after training are naturally interpreted as novelties [Vas95, MS03b]. On the other hand, MLPs separate training classes by using hyper-planes forming open boundaries between classes instead of around classes. However, a number of modifications have been proposed to make MLPs construct closed class boundaries and thus be able to detect novelties - also called *spurious patterns* - without the need for negative samples during training [Vas95, MS03b]. It has been shown that MLP networks combined with additional mechanisms have performance similar to RBF networks on an optical character recognition task [Vas95]. RBF networks have the advantage of being faster to train. On the other hand, the modified MLPs have smaller number of processing units and therefore they are less complex from a computational point of view [Vas95].

There has been an increasing interest in time series novelty detection because this kind of data appear in virtually every application domain; novelty detection for this kind of data has been applied in areas such as machine failure detection [GD02] and auditing, that is, fraud detection [Kos00, Kos03, OABS03]. A number of different techniques have been proposed and investigated, including techniques based on time series forecasting with neural networks [ONM04e, Kos00, OABS03], artificial immune system [DF96, GD02], wavelets [STZ00], Markov chains [KLC02] and one-class support vector machines [MP03].

There are some methods in the literature for novelty detection in time series based on time series forecasting [ONM04e, Kos00, Kos03, OABS03]. The basic idea behind these methods consists in training a forecasting model, such as a neural network, with historical data from a time series and use the trained model to predict future values. Subsequently, if a value predicted by the forecasting model deviates significantly from the observed value then the system would detect a novelty which, in some domains such as auditing, would mean fraud. In a simple approach, one must previously define a *threshold* to be used for novelty detection. Differences between predicted and observed values beyond the previously selected threshold indicate a novelty.

This approach has been applied to accounting auditing [Kos00] and payroll auditing [OABS03]. However, it is criticized because of the not so good performance given the complexity of time series forecasting [KLC02, GDK02]. Performance of neural networks in time series forecasting depends on the amount of data used for training them. In many

important auditing problems the available data is limited, for example, in an accounting auditing work there were only 66 samples available for both training and testing [Kos00]; in a payroll auditing problem the number of samples was 84 [OABS03]. This makes forecasting an even harder problem. The definition of a suitable value for the threshold for detecting novelties is also a difficult point for forecasting-based novelty detection techniques [OABS03].

A number of classification-based methods for novelty detection in time series have been proposed to overcome the limitations of forecasting-based methods. These methods are able to classify a given window from a time series into normal or novelty. Methods based on *artificial immune systems* have appeared in the literature for classification-based novelty detection in time series [DF96, GD02, GDK02, DG02]. Artificial immune systems are a recent class of computational intelligence technique inspired on the mammal immune systems [DAO97, Das97]. Methods based on artificial immune systems use the principle of negative-selection, that is, negative samples are used to represent novelties [Vas95]. The artificial immune systems ideas are used to generate *detectors*, which are used to detect novelties.

Another approach for classification-based novelty detection detection in time series is based on a data structure termed TSA-tree [STZ00]. TSA is constructed by recursively applying wavelet transforms [BGG97] on the original time series. A limitation of TSA-tree is that its size is larger that the original time series. Shahabi et al. address this problem by proposing the use of a sub-tree of TSA-tree, termed OTSA-tree, which is equal in size to the original time series and can be used to reconstruct the original time series with no error [STZ00]. Nevertheless, it is argued that this method cannot detect short novel patterns embedded in normal signals [MP03].

The problem of detecting surprise patterns in time series through classification is also tackled by the TARZAN algorithm [KLC02]. This algorithm is based on converting the time series into a symbolic string. Next, a suffix tree is used to efficiently encode the frequency of observed patterns and a Markov model is used to predict the expected frequency of previously unobserved patterns [KLC02]. The problem with this method is that the procedure for discretizing and symbolizing real values in time series can potentially loose meaningful patterns in the original time series.

More recently a method based on one-class support vector machines was proposed for time series novelty detection via classification of a time series window, that is, a sequence of values from the time series [MP03]. This method does not need negative samples for training. Time series windows are used to generate vector patterns and novel events are interpreted as outliers of the "normal" distribution. Support vector machines are powerful classifiers based on the method of structural risk minimization [STC04, Vap95, CV95, Hay98]. One-class support vector machines are trained only with the normal patterns that form the time series, that is, they are all from the same class. Thus, they appear as a natural technique for novelty detection. This method has a potential limitation in that the sensitivity of the algorithm depends on the value of one of its parameters [MP03].

All classification-based techniques mentioned before were designed and tested on time series modeling physical processes, such as a mill machine [GD02], the power demand for

a Dutch research facility [KLC02] and the time series model of a laser used in the well known Santa Fe Institute Competition [MP03, WG94, Wan94]. On the other hand, the forecasting-based methods have been designed and tested on financial time series [ONM04e, Kos00, Kos03, OABS03]. The classification-based methods proposed in this work were also designed for these kinds of series and tested on real-world time series obtained from financial systems such as payroll and accounting systems. As discussed before, in many practical cases these series are much shorter than the time series considered in previous classification-based techniques [GD02, KLC02, MP03]. Moreover, the detection of frauds in financial data requires algorithms designed to detect more subtle novelties.

A limitation shared by all classification-based methods for novelty detection in time series is that these methods are not capable of detecting a novelty in a single point of the time series, because such methods are designed to detect novelties in windows of a time series. An example is in need to make this point clearer. Suppose that a classification-based method is used for novelty detection in a financial system with monthly time series. In addition, suppose that the window length selected was $w = 6$. If such a system was used in practice it could indicate that a novelty took place in the whole window, yet it would not be capable to indicate the particular month (or months) in which there was novelty. Such a capability is very important and particularly desirable for novelty detection applications in financial systems. For other applications, such as machine fault detection, this issue is not important, since in this domain the change in the behavior of the time series modeling the machine can only be detected effectively using a window from the time series [GD02].

1.3 OBJECTIVES

The main objective of the research reported on this thesis was to investigate and present methods for detection of novelties in time series. The methods investigated in this work are all based on neural networks. They can be divided into two classes: forecasting-based and classification-based methods. The methods proposed in this work are designed to deal with the kind of time series that appear in auditing problems such as accounting auditing [Kos00, Kos03] and payroll auditing [OABS03].

Forecasting-based novelty detection systems have the ability to detect novelties on a particular point in time. For example, in the case of monthly time series those methods can detect a novelty in a particular month. Classification-based methods, on the other hand, can only detect a novelty on a time series window. A system of this kind with window size $w = 12$ applied to a monthly time series can only detect novelties in a year but cannot indicate in which particular month (or months) the novelty has happened.

The problem with forecasting-based novelty detection methods is the choice of a suitable value for the threshold for the detection, as commented in subsection 1.2. Nevertheless, due to the important capability of these methods, that is, the capability of detecting single point novelties in time series, it is important to investigate techniques for improving them. This work proposes and investigates a method based on the use of robust confidence intervals [Mas95] as a means for selecting the thresholds to be used to

detect novelties in time series [ONM04e].

Two methods for novelty detection in time series based on classification are also proposed in this work [ONM03, ONM04a, ONM04d, ONM04b]. The first method is based on the use of artificially generated patterns added to the training set. There are two kinds of artificial patterns: random normal patterns and random novelties (negative samples). In our methods, we propose to use an *envelope* defined around each time series windows in order to defined the boundary between normal and novelty regions. Normal patterns lie inside the envelope whereas novelty patterns lie outside of it. After training, a classifier is used to discriminate normal and novelty time series windows [ONM03]. A number of different classifiers including MLPs, RBF-DDA [BD95, BD98], committee machines combining these classifiers, and support vector machines are considered as classifiers and their performance is compared in this task [ONM04a, ONM04b].

The main disadvantage of the method commented above is that its performance depends on the number of artificial patterns introduced in the training set. This has motivated the proposition of a novel method based on RBF-DDA and that does not need artificial patterns for training [ONM04d]. The classification-based methods proposed in this work are compared using some real-world time series.

The classifier is the most important part of the novelty detection methods based on classification proposed in this work. Thus, the performance of the classifier is fundamental to the novelty detection performance. This has motivated an investigation of methods proposed to improve RBF-DDA generalization performance on classification tasks. This classifier was selected for investigation because it has a number of advantage including the effective use of all training data for training, the fast training, and its constructive nature [BD95, BD98]. These advantages have motivated the implementation of this neural network model in hardware [LSP+96, KS02] and has also motivated recent efforts towards improving it with respect to the complexity of the networks constructed [Pae04]. Four methods were proposed in this thesis for improving RBF-DDA generalization performance [ONM04c, OMNM04, OMM05a, OMM05b].

The first method is based on the selection of the optimal value of one of the parameters of the DDA algorithm, namely, θ^- [ONM04c]. The selection of the optimal value is made by evaluating the classifier on a validation set. Experiments reported later on this thesis show that this method considerably improves performance on some datasets, however it produces larger networks than the default RBF-DDA. This has motivated further investigations in order to propose methods that improve DDA generalization without increasing the complexity of the networks. Three different methods were proposed in this direction. The first one builds an RBF network using DDA and then makes a final weights adjustment on RBF-DDA networks. It also improves RBF-DDA performance with the advantage of producing smaller networks than the first method.

The second method aiming at improving RBF-DDA without increasing the networks combines a data reduction algorithm with the adequate selection of one of the parameters of DDA (θ^-) in order to improve generalization performance [OMNM04].

Finally, a third method, which we call RBF-DDA-SP, that is RBF-DDA with selective pruning, is proposed and investigated in this thesis [OMM05a]. RBF-DDA-SP was inspired in the recent work of Paetz, which proposed RBF-DDA-T [Pae04]. RBF-DDA-T

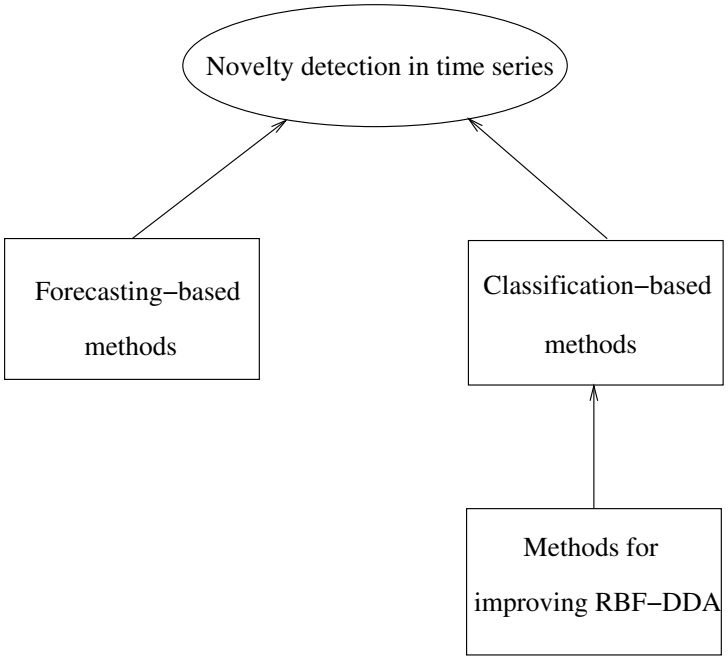


Figure 1.3. This thesis investigates the problem of novelty detection in time series. Methods based on both forecasting and classification are proposed to tackle this problem. In addition, this thesis proposes methods for improving classification performance of RBF-DDA in order to improve classification-based novelty detection methods.

is a modification of RBF-DDA which introduces on-line pruning, that is, pruning during each epoch of training. RBF-DDA-T produces much smaller networks than RBF-DDA, yet it introduces a small degradation in classification performance [Pae04]. On the other hand, RBF-DDA-SP, proposed in this thesis, prunes the network only at the end of training and prunes only a portion of the neurons which cover only one training pattern. The experiments reported this thesis show that RBF-DDA-SP produces networks whose size is between RBF-DDA-T and RBF-DDA. Moreover, the results show that RBF-DDA-SP markedly outperforms both RBF-DDA and RBF-DDA-T regarding classification performance [OMM05a].

It is important to stress that the main objective of this thesis is the investigation of novelty detection methods for time series. This is the main problem addressed by this thesis, as shown in figure 1.3. This thesis has proposed both forecasting-based and classification-based methods to tackle this problem. In addition, this thesis has also proposed four methods for improving RBF-DDA in order to improve one the methods for classification-based novelty detection in time series. The results obtained are compared to those obtained by other classifiers in this thesis and they show that the proposed modifications to RBF-DDA leads to considerable improvements in performance. [ONM04b]. In addition, the methods proposed for improving RBF-DDA were also applied to a number of benchmark classification datasets from the UCI machine learning repository [BM98] in order to show that they improve results on different benchmark classification datasets as well as in the time series novelty detection task.

1.4 OVERVIEW OF THE THESIS

This thesis is organized in seven chapters in total. In this chapter, the motivations for carrying out this research are presented together with a brief overview of works recently carried out that have relation with the subject of the thesis. A more detailed review of related works is reported on chapter 3 of this thesis.

Chapter 2 reviews a number of classification algorithms as well as time series forecasting algorithms that are used in subsequent chapters of this thesis together with the novelty detection algorithms proposed. The classifiers presented in this chapter are the multilayer perceptron (MLP) neural network, the radial basis functions network (RBF) and committee machines. A number of alternative training algorithms used in conjunction with these neural networks are discussed. This chapter also discusses classical time series forecasting techniques such as ARIMA and exponential smoothing as well as neural networks architectures for time series forecasting such as MLPs and Elman neural networks.

In chapter 3, a number of novelty detection techniques are reviewed. This chapter begins discussing novelty detection techniques for classification problems. These techniques have been introduced first and aim at improving the reliability of neural networks with respect to the rejection of *spurious* patterns. Next, a number of recent techniques proposed for novelty detection in time series are reviewed. Both forecasting-based and classification-based time series novelty detection methods are presented in this chapter.

In chapter 4, four alternative methods for improving RBF-DDA performance on classification tasks are presented. Simulation results using six benchmark datasets from the UCI machine learning repository are given. Performance of the improved RBF-DDA classifiers based on these methods are compared to that of the original RBF-DDA. They are also compared to results available in the literature obtained using other classifiers on these datasets, such as MLPs, kNN and support vector machines.

Chapter 5 presents two novel methods proposed in this work for novelty detection in time series based on classification. Both methods are designed to classify an input time series window as normal or novelty. They are based on the notion of envelope introduced in this chapter. The envelope is used to defined normal and novelty regions. One of the methods is based on negative samples whereas the other does not need them for training. The former method can be used with any classifier. In this chapter we consider RBF-DDA with default parameters; the improved RBF-DDA classifier proposed in chapter 4; MLPs; and committee machines of these networks as classifiers. On the other hand the second method is inherently based on RBF-DDA neural networks. This chapter also presents simulations results aiming at comparing the proposed methods. Simulations were carried out using a number of real-world time series, including short time series.

Chapter 6 presents alternative methods proposed in this work for improving forecasting-based novelty detection in time series. The use of absolute values for the threshold instead of relative ones is discussed. Additionally, the use of robust confidence intervals as a natural way to establish thresholds is proposed. This method is applied to a number of real-world time series.

Finally, chapter 7 discusses the main contributions of this thesis. The limitations of

the methods proposed here are also presented in this chapter. A discussion of directions of possible future research to be carried out is also presented.

CHAPTER 2

CLASSIFICATION AND TIME SERIES FORECASTING TECHNIQUES

2.1 INTRODUCTION

This chapter reviews the classification algorithms and time series forecasting techniques used in the thesis. This work uses artificial neural networks for both classification and time series forecasting. Artificial neural networks (ANNs) are a class of computational models inspired on the human brain. ANNs were designed to solve problems for which traditional computing does not perform well, including perception problems such as face and speech recognition [Hay98, BLC00]. ANNs main characteristics include their highly distributed nature and their ability to *learn* from previous examples.

In this work two ANN architectures are considered for classification problems: the multi-layer perceptron (MLP) and radial basis functions networks (RBFNs) [Hay98, BLC00, Nab01]. These neural network models are reviewed together with a number of training methods used in conjunction with them. A discussion about the different properties of these classifiers is also included in this chapter. This chapter also discusses a number of different ways to combine these classifiers by using classifiers referred as *committee machines* [Hay98]. Particular emphasis is placed on the dynamic decay adjustment algorithm (DDA) for RBF networks. This is a fast constructive algorithm for building and training RBF networks. This thesis proposes four different methods for improving DDA classification performance (in chapter 4). This is the reason for a detailed exposition of the DDA algorithm here.

In this work we also use Support Vector Machines (SVMs) for classification and therefore this chapter also presents a brief review of this technique. Support Vector Machine is a more recent technique for classification and regression which has achieved remarkable accuracy in a number of important problems [CV95, Vap95, STC04, CST00, A.03].

This chapter also discusses the use of neural networks for time series forecasting. Time series forecasting is a very important problem with application in various areas. A number of techniques have already been developed for tackling this problem, including exponential smoothing and ARIMA, sometimes referred as *classical techniques* [Cha89, Mas95]. The classical techniques work by building linear combinations of past values in order to forecast future values of time series. On the other hand, neural networks models include non-linearities, which can help to produce more powerful models for some kinds of time series [Cha89, Mas95, ZPH98].

In this chapter two artificial neural network models for time series forecasting are presented: multi-layer perceptron (MLP) and Elman networks [Elm90]. The MLP model, initially presented as a classifier in this chapter, can also be used successfully for time series forecasting with small modifications. The same algorithms used for training MLPs

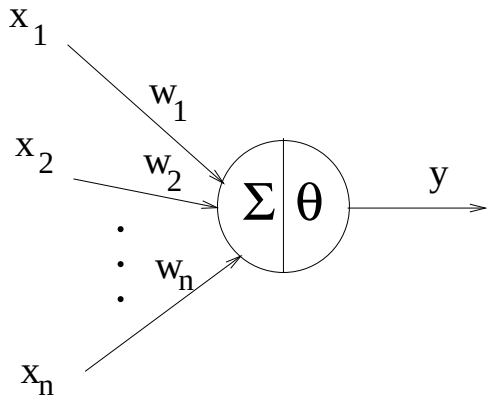


Figure 2.1. McCulloch-Pitts (MCP) neuron model

for classification can also be used to train these networks for time series forecasting. In MLPs the temporal processing is located only in the network input. This limits the application of these networks to the prediction of stationary time series. In order to use this architecture for forecasting of non-stationary time series, it is necessary firstly to pre-process the time series in order to make it stationary. Differentiation is the most common technique used for this purpose [Cha89, Mas95, ZPH98].

Elman networks are a kind of partially recurrent neural network architecture [Elm90, Hay98]. Elman neural networks were designed for temporal processing applications and are a powerful architecture for time series forecasting [Hay98, BLC00, KLSK96]. Elman networks have *context units*, whose nodes make recurrent connections between hidden and input layers. Recurrent connections provide hidden nodes inputs with past values of their own outputs. This adds memory to the network and provides better temporal processing performance. Algorithms originally employed for MLP training were adapted for training these networks.

2.2 NEURAL NETWORKS FOR CLASSIFICATION

2.2.1 Multi-Layer Perceptrons – MLPs

The multi-layer perceptron (MLP) is the most popular neural network architecture. The MLP has its origins on the perceptron, proposed by Frank Rosenblatt in 1958 [Ros58]. The perceptron is historically very important because *learning* in artificial neural networks was introduced for the first time in this model. The artificial neuron model used in the perceptron was developed previously by McCulloch and Pitts [MP43]. This model is depicted in figure 2.1 and is referred as the MCP neuron model. Although the model was developed in 1943, the training algorithm was developed only in 1958 by Rosenblatt [Ros58].

The MCP neuron model works by computing the weighted sum of its inputs. Next, the output y is computed by using an *activation function*. In the original model the activation function is a hard limit, which produces binary output. In this case, the output is 1 only if the weighted sum of the inputs is greater than a threshold θ. Mathematically,

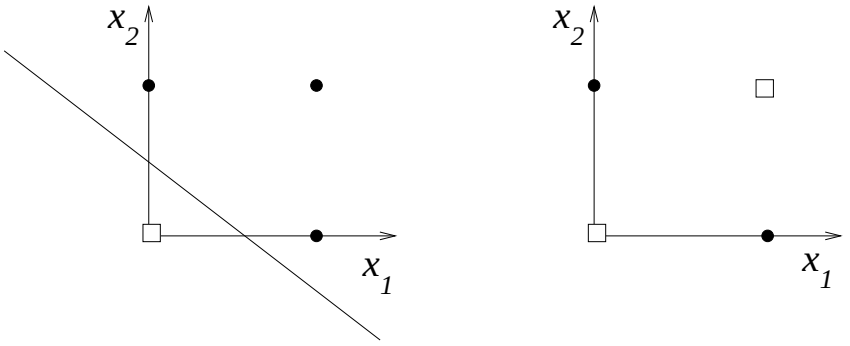


Figure 2.2. Left: AND function (linearly separable). Right: XOR function (non-linearly separable)

$$y = \begin{cases} \sum_{i=1}^{n} x_i w_i \geq \theta \\ \sum_{i=1}^{n} x_i w_i < \theta \end{cases} \tag{2.1}$$

where n is the number of inputs of the neuron, w_i is the *weight* associated to input x_i and θ is the threshold of the neuron.

The perceptron can be used to solve classification problems, however it is a limited model since it can only solve *linearly separable* problems. Linearly separable problems are classification problems that can be solved by using a hyperplane in order to set the boundary between two regions. The logical AND function shown in the left chart of figure 2.2 is a very simple example of linearly separable problem in two dimensions. For two dimensional problems, the equation that defines the MCP operation (eq. 2.1) produces the line $x_1 w_1 + x_2 w_2 = \theta$. The weights w_1 and w_2 and the bias θ define the line. There are other pattern recognition methods which solve classification problems by using hyperplanes. Examples of such methods include Fisher's linear discriminant and linear support vector machines (SVMs) [A.W02, TK03, DHS00, CST00, STC04].

The problem with linear methods is that most practical real-life problems are not linearly separable. This includes the simple XOR logical function depicted in the right chart of figure 2.2. It is clear that no single line can separate the two classes in this case.

The weights of the MCP neuron are adjusted in order to adequately solve a given problem. The weights are *learned* by the training algorithm using the training patterns. Rosenblatt has shown that the MCP neuron can be trained iteratively by adjusting the weights according to

$$\overrightarrow{w}(t+1) = \overrightarrow{w}(t) + \eta \times e(t) \times \overrightarrow{x}(t) \tag{2.2}$$

where $e(t)$ is the output error, that is, the difference between the desired output d and the output produced by the network, $y(t)$; and η is the learning rate, that is, the rate at which the weights are adjusted. Rosenblatt has proved that this training algorithm always converges to the solution if the problem is linearly separable [Ros62, BLC00, Hay98].

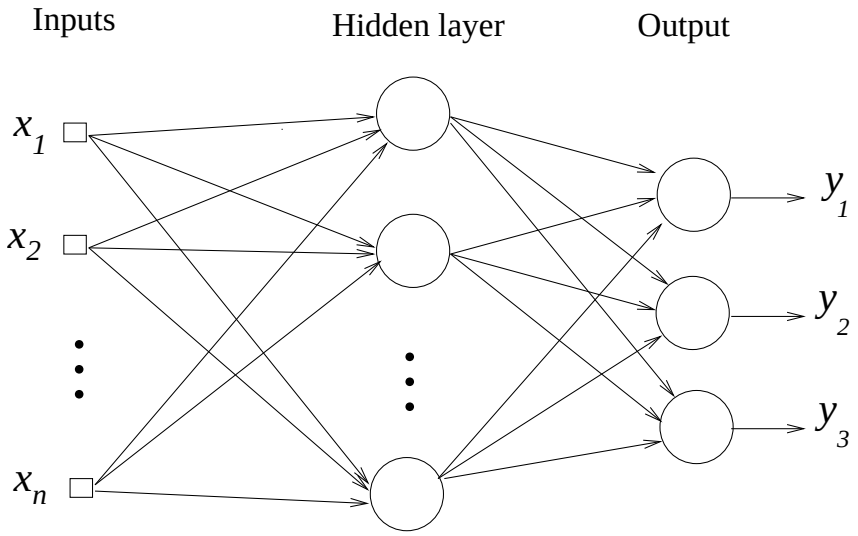


Figure 2.3. An MLP with a single hidden layer

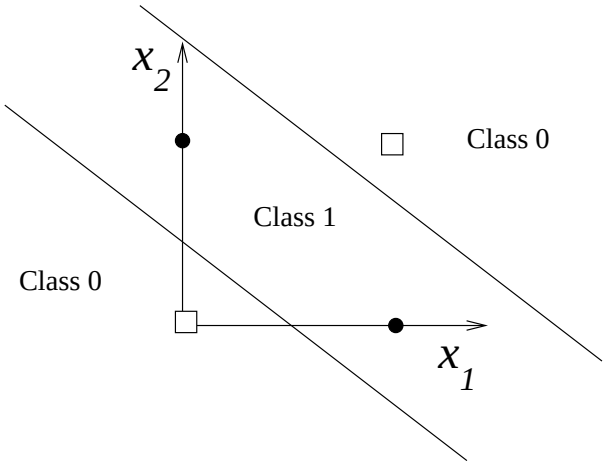


Figure 2.4. XOR function (non-linearly separable)

2.2.1.1 MLP Architecture The multi-layer perceptron (MLP) was proposed in order to solve non-linearly separable problems. The idea consists in adding at least one hidden layer to build a multi-layer network. Figure 2.3 shows a multilayer perceptron with one hidden layer and three output units. The neurons of the hidden layer are responsible for creating hyperplanes whereas the output neurons combine these hyperplanes in order to build more complex boundary regions [Hay98, BLC00, Nab01]. An MLP with a single hidden layer with two neurons and one output neuron is able to solve the XOR problem [Hay98, BLC00]. A possible solution created by such network is depicted in figure 2.4. In this case, each line is created by a hidden neuron. The output neuron combines the outputs from the hidden neurons to generate the solution depicted in figure 2.4.

It was proved that MLPs with a single hidden layer are capable of approximating any continuous functions [Cyb89] and that MLPs with two hidden layers can approximate any

mathematical function [Cyb88]. MLPs with a single hidden layer are the most widely used architecture. However, in some cases the use of two or more hidden layers can easy network training [BLC00]. Most MLP architectures are *fully connected*, that is, all feedforward connections between adjacent layers are present, as in figure 2.3. There has been, however, an increasing number of works aiming at optimizing MLP architecture by using techniques such as evolutionary computing [SM02, BM01, Yao99], tabu search, and simulated annealing [YdSL02, YLdS02b, YLdS02a]. These techniques generate MLP architectures that are not fully connected aiming at improving classification performance. These are global optimization techniques designed to overcome the main limitation of back-propagation, the traditional training technique for MLPs, which often finds solutions that are only local minima of the error function instead of the desired global minimum [WTT+04, Yao99, BLC00].

The input units of an MLP do not perform any transformation in the input signals. They exist only to receive the input signals and distribute them to the neurons of the first hidden layer. Every connection in an MLP has a weight whose value is learned during the training phase. In addition, each neuron has a bias whose value is also learned during training. These values depend on the data used for training the network.

Each unit of the hidden and output layers initially computes a weighted sum of its n inputs $net = \sum_i^n x_i w_i$. Subsequently, an activation function is used to produce the output of the neuron. The activation function uses net as its input. A number of different activation functions have been proposed for use with the MLP [Hay98]. The hard limit activation function cannot be used in this case because in most learning algorithms the gradient of the activation function must be computed. The problem is that the hard limit is not continuous and therefore does not have derivative. The *sigmoid logistic* and the *hyperbolic tangent* are the most frequently employed activation functions. The sigmoid logistic computes the output y as

$$y = \frac{1}{1 + exp(-net)}$$

where net is weighted sum of the neurons inputs. The hyperbolic tangent computes the output as $y = tanh(net)$. In fact the hyperbolic tangent is a kind of sigmoid function. The main difference between the hyperbolic tangent and the sigmoid logistic is that the latter produces continuous values between 0 and +1 whereas the former produces values between -1 and +1.

The number of inputs of an MLP depends on the dimensionality of the problem. For example, in an optical character recognition task with characters represented by 8×8 pixels, the network has 64 inputs. The number of outputs depends on the number of classes. There are a number of alternative ways of coding the outputs [Hay98]. This thesis uses the commonest way, that is, $1 - of - n$ coding. In this case, the number of units in the output layer matches the number of classes. Each unit in the output layer represents a class. After training, the MLP is used to classify a given pattern commonly by using the winner-takes-all approach. In this approach, the unit with the highest activation gives the class.

The number of units of the hidden layer also depends on the problem. This number can greatly influence the classification performance of the network. There are some heuristics for selecting the number of hidden of units based on the characteristics of the training data [BLC00]. However, they do not work for all cases. If few hidden units are used the network may not be capable of learning a good solution for the problem. On the other hand, if the number of units is large for the task, a problem called *overfitting* may occur. In overfitting, the training algorithm adjusts the weights and bias excessively and the network behaves as if it had memorized the training data. The problem is that in practical pattern recognition problems a good *generalization* performance is of great concern. Generalization means that the network will be able to classify correctly patterns that were not part of the training data. A number of techniques exists for improving generalization [BLC00, Hay98, Nab01]. Some of them are discussed in this chapter in conjunction with training algorithms for MLPs.

2.2.1.2 Training MLPs with Back-Propagation The limitations of the perceptron were known in the 1960's. The proposal to use a multi-layer architecture in order to overcome these limitations and solve non-linearly separable classification problems was also from this period. However, no method for training the MLP was available. The first successful training method for MLPs, known as *back-propagation*, was made popular only in 1986 [RHW86, RM86]. In fact this method has been proposed before for different purposes [BH69, Wer74, LeC85].

The back-propagation algorithm is based on error correction as was the case of the perceptron training algorithm. During training, the error of the output units is known because the desired outputs are known in supervised training. However, the errors of the hidden units are not known directly. Thus, a method for estimating these errors would have to be developed. Back-propagation tackles this problem by estimating the error of each hidden unit as the error of the nodes of the adjacent units connected to it weighted by the weights of the respective connections.

The back-propagation algorithm consists of two phases: a *forward* and a *backward* phase. In the forward phase a pattern from the training set is presented at the network inputs. Next, the units of the first hidden layer compute their outputs. These outputs are used by the next hidden layer to compute the respective outputs. This procedure is carried out until the output layer units compute their outputs. Now the outputs produced by the network are compared to the desired outputs and the errors are calculated. In the backward phase, these errors are used to adjust the weights of the connections to the output layer. Subsequently, they are used to estimate the error on the hidden layer connected to the output layer, as described above. If the network has more than one hidden layer, this procedure is carried out until the first hidden layer (the one to which the input layer is connected).

The objective of the back-propagation training algorithm is to minimize the sum of squared errors (SSE) on the training set. The SSE E is given by

$$E = \frac{1}{2} \sum_{j=1}^{p} \sum_{i=1}^{k} (d_i^j - y_i^j)^2 \tag{2.3}$$

where p is the number of training patterns, k is the number of output units, d_i is the desired value of the ith output and y_i is the value produced by the network on the ith output.

For a complete derivation of the back-propagation algorithm see [RHW86, RM86, Hay98, BLC00]. In this algorithm the weights and bias are adjusted, in each training epoch t, according to

$$w_{ji}(t+1) = w_{ji}(t) + \eta \times \delta_j(t) \times x_i(t) \tag{2.4}$$

where η is the learning rate. If j is an output neuron, δ_j is given by $\delta_j = (d_j - y_j)f'(net_j)$, where net_j is the weighted sum of the inputs of the neuron and $f'(.)$ is the partial derivative of the activation function with respect to net_j. On the other hand, if j is a hidden neuron, δ_j is given by

$$\delta_j = f'(net_j) \sum_l \partial_l w_{lj}$$

One of the problems of the back-propagation training algorithm is that the choice of the learning rate η has an important effect on training performance. If this value is too small, too many epochs are needed to reach an acceptable solution. On the other hand, a large learning rate will possibly lead to oscillation, preventing the error to fall below a certain value, that is, the minimum error can possibly not be reached. The best η value depends on the problem.

A number of alternative training algorithms have been proposed in the literature since back-propagation. This includes other algorithms also based on gradient-descent that aim at accelerating training and/or not being dependent on the learning rate. Back-propagation with momentum, QuickProp and Rprop are examples of improved gradient-descent algorithms [BLC00, Hay98, RB93]. There are also a number of training algorithms based on second order optimization, such as *scale conjugate gradient* and the *Levenberg-Marquadt* [Hay98, Zel98]. Second order techniques generally find a better way to a (local) minimum than first order techniques but at a higher computational cost.

The main problem of these traditional training techniques is the local minima problem [WTT+04, BLC00, Hay98]. These algorithms were designed to find a minimum, which is not guaranteed to be the global minimum. They often stop training at a local minima of the error function and are thus unable to find the global minimum. More recently, a number of works have tackled the local minima problem by using global optimization techniques such as genetic algorithms [SM02, Yao99] or by modifying back-propagation [WTT+04].

2.2.1.3 The Rprop Training Algorithm In this work, MLPs are trained by the *resilient back-propagation* algorithm (Rprop) [RB93]. The Rprop is a local adaptive learning algorithm whose main idea consists in eliminating the harmful influence of the size of the partial derivative on the weight update $\Delta w_{ij}(t)$. As a consequence, only the sign of the derivative is considered to indicate the *direction* of the weight update. The *size* of the weight change is exclusively determined by a weight-specific update value Δ_{ij} described below. In practical applications Rprop can train MLPs much faster than standard

back-propagation. Moreover, the former training algorithm has a few parameters that do not much influence results whereas the latter is strongly influenced by the value of the learning rate.

In Rprop each weight of the neural network has its individual update value Δ_{ij}. In each training epoch, every weight is updated using

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

where $\Delta w_{ij}(t)$ is given by

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t), & \text{if } \frac{\partial E(t)}{\partial w_{ij}} > 0 \\ +\Delta_{ij}(t), & \text{if } \frac{\partial E(t)}{\partial w_{ij}} < 0 \\ 0, & \text{else} \end{cases} \tag{2.5}$$

The update values Δ_{ij} are adapted during training based on the local behavior of the error function E, according to equation 2.6. This equation can be interpreted as follows. Every time the error partial derivative changes its sign (from consecutive epochs), the update value is decreased by a factor η^-. If the derivative maintains its sign, the update value Δ_{ij} is slightly increased by the factor η^+ in order to accelerate convergence in shallow regions.

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \times \Delta_{ij}(t-1), & \text{if } \frac{\partial E(t-1)}{\partial w_{ij}} \times \frac{\partial E(t)}{\partial w_{ij}} > 0 \\ \eta^- \times \Delta_{ij}(t-1), & \text{if } \frac{\partial E(t-1)}{\partial w_{ij}} \times \frac{\partial E(t)}{\partial w_{ij}} < 0 \\ \Delta_{ij}(t-1), & \text{else} \end{cases} \tag{2.6}$$

The Rprop algorithm includes a number of parameters, but is has been shown that their choice is rather uncritical, that is, the network performance and training time only slightly depends on the values of the parameters. Thus, the use of the default values of these parameters is recommended [RB93]. At the beginning of training all update values Δ_{ij} are set to an initial value Δ_0. There are also two parameters for restricting the upper and lower limits for the update values. These parameters are Δ_{max} and Δ_{min}. The default values for these parameters are $\Delta_0 = 0.1$, $\Delta_{max} = 50.0$ and $\Delta_{min} = 10^{-6}$. It has been shown that good choices for the increase and decrease factors of equation 2.6 are $\eta^+ = 1.2$ and $\eta^- = 0.5$, respectively [RB93].

2.2.1.4 Criteria for Stopping Training The *generalization* performance of a classifier is very important in practical applications. Generalization is the ability of a classifier to correctly classify patterns that were not used for training. If an MLP is trained for an excessive number of epochs the network can simply "memorize" the training data, which can result in poor generalization capabilities. This problem is called overfitting. It is very difficult to set the number of epochs that should be used for training an MLP for a given problem. This number is also problem dependent like the learning rate of back-propagation. A number of alternative criteria for stopping training were proposed in order to overcome this difficulty.

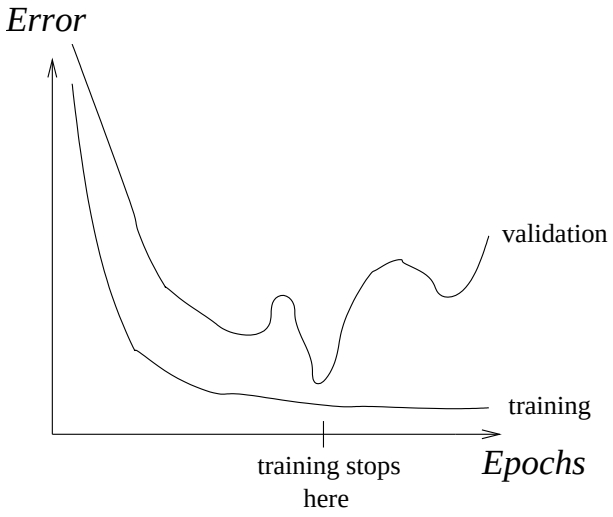


Figure 2.5. Typical behavior of validation set and training set errors during training

Early stopping is one of the most popular techniques used to avoid overfitting in MLP training [Hay98, Pre94]. In this technique, the training data is divided into disjoint training and validation sets. During training, the weights and bias of the neural network are adjusted using only data from the training set. The performance of the network on the validation set is computed periodically during training. This performance is used to estimate the generalization performance of the network, since validation patterns are not used for adjusting weights. The network is trained until the validation error reaches a minimum. Typically, the error on the validation set decreases when training starts, but then increases. This indicates that the network may be overfitting the training data.

A possible stopping criteria would be to stop training at the first increase of the validation error. This, however, can make training stop sooner than desirable because there may be oscillations on the validation error, as shown in figure 2.5. The GL_5 criterion from *Proben1* [Pre94] was proposed to overcome this problem. This criterion is defined as follows. Let $E_{opt}(t)$ be the lowest validation set error (sum of squared errors - SSE - on this set) obtained up to training epoch t. Let $E_{val}(t)$ be the validation error at epoch t. The generalization loss $GL(t)$ at epoch t is defined as the relative increase of the validation error with respect to the minimum $E_{opt}(t)$. The $GL(t)$ in percent is defined as:

$$GL(t) = 100 \times \left(\frac{E_{val}(t)}{E_{opt}(t)} - 1 \right)$$

The GL_5 stop criterion states that training stops when $GL(t) > 5$, that is, when the validation error at epoch t is 5% above the minimum validation error obtained up to epoch t.

A second criterion referred as the *training progress criterion* is also proposed in Proben1 [Pre94]. The training progress $P_k(t)$, measured in parts per thousand, is evaluated after a training strip k. A training strip of length k is a number of consecutive epochs $n+1 \cdots n+k$, where n is divisible by k. The training progress P_k measures how

much the average training error during the strip is larger than the minimum training error during the strip. The stopping criterion from Proben1 states that training must stop when the training progress $P_k < 0.1$ per thousand. The training progress is given by

$$P_k(t) = 1000 \times \left(\frac{\sum_{t' \in t-k+1 \cdots t} E_{tr}(t')}{k \min_{t' \in t-k+1 \cdots t} E_{tr}(t')} - 1 \right)$$

where $E_{tr}(t')$ is the sum of squared errors on the training set at epoch t'. The training progress criterion states that training must stop when the training error is not decreasing significantly anymore. The training progress does not take validation error into account. Proben1 proposes to use this criterion and the GL_5 simultaneously.

For small datasets the use of training data exclusively for validation can mean loosing important information for learning. In order to tackle this problem, a simple generalization of the above method (also referred as *simple validation*), *m-fold cross-validation*, can be used [DHS00]. In this technique, training data is divided into m disjoint sets of (approximately) equal size. Training is carried out m times. In each training instance, one of the m subsets is reserved for testing and for the evaluation of generalization performance (usually the sum of squared errors or the classification error on this set). The estimated performance is the mean of these errors. This procedure can be used in conjunction with any classifier. For MLPs trained with back-propagation, cross-validation can be used to select the optimal learning rate and maximum number of training epochs to be used for a given dataset.

2.2.1.5 The RpropMAP Training Algorithm The RpropMAP training algorithm is also referred as Rprop with adaptive weight-decay. It is an extended version of the Rprop algorithm that uses a weight-decay regularizer whose weighting parameter λ is computed automatically within the Bayesian framework [Mac92a, Mac92b, Nab01, Zel98]. Weight decay is an heuristic method used to simplify a network and avoid overfitting. It is found that in many practical cases this technique can improve generalization performance [DHS00, Nab01, Hay98]. Weight decay is one of the methods based on the idea of complexity regularization. In these techniques, the objective of training is not to minimize the error, but to minimize the total risk, expressed as

$$R(\overrightarrow{w}) = E_s(\overrightarrow{w}) + \lambda E_c(\overrightarrow{w}) \tag{2.7}$$

where $E_s(\overrightarrow{w})$ is the performance measure (training error), which depends on the network and the training data, and $E_c(\overrightarrow{w})$ is a term used to penalize the complexity of the classifier. The latter term depends only on the parameters of the classifier, which are the weights in the case of neural networks. λ is referred as a *regularization parameter*. The total risk presented above is based on Tikhnov's theory of regularization [Hay98], which is used as a base for a number of methods for simplifying neural networks, including weight decay [Hay98].

Training with weight decay is very simple. During training, after each weight update, every weight is decayed or shrunk according to $w^{new} = \lambda \times w^{old}$. Weight decay is used

together with a number of training algorithms, including back-propagation and Rprop [Zel98]. In practical applications the selection of λ may be troublesome. The RpropMAP tackles this program by automatically computing this value using Bayesian methods.

The RpropMAP algorithm uses *adaptive* weight decay. In this case, the value of the weighting parameter is adapted during training. In RpropMAP this is done by using the MAP (maximum a posteriori) approach [DHS00, Mac92a, Mac92b, Nab01]. It is proved that Bayesian methods actually embody *Occam's razor* automatically and quantitatively [Mac92a]. Occam's razor is the principle that states that unnecessarily complex models should not preferred to simpler ones. Bayesian methods are used to automatically compute λ, the regularization parameter of equation 2.7 [Mac92a]. These methods can be used to select the best model to fit a given dataset. It is important to stress that the method is general and therefore the model does not need to be a neural network [Mac92a, Nab01]. The application of the method in conjunction with back-propagation for training MLPs has been detailed in the literature [Mac92b].

RpropMAP assumes that the weights have a Gaussian distribution with zero mean and variance $1/\alpha$ and that the error also has a Gaussian distribution with zero mean and variance $1/\beta$ [Nab01, Mac92b]. These *hyper-parameters* are estimated by maximizing the evidence, which is the a posteriori probability of α and β. They are used to compute $\lambda = \alpha/\beta$. This is done periodically during training, for example, at every 50 epochs. At these epochs α and β are re-estimated by $\alpha_{new} = W/\sum w_i^2$ and $\beta_{new} = N/E_s$, where W is the number of weights and N is the number of patterns.

A validation set is not needed for training MLPs with RpropMAP. This is an important advantage of the method because all training data can be effectively used for training, that is, for adjusting the network weights and bias. In practical applications results are better when the initial guess of λ is good. This reduces the number of necessary iterations as well as the probability of overfitting heavily in the initial epochs of training [Zel98]. A good initial guess for λ can be obtained by using cross-validation techniques.

2.2.2 Radial Basis Functions Networks – RBFNs

Radial basis function networks (RBFNs) are among the most widely used neural networks models, along with MLP networks [Hay98, BLC00, Nab01]. RBF networks are widely used in classification, regression and time series forecasting. These networks are normally much faster to train than MLPs, which can be an advantage in practical applications. This neural network model behaves quite differently regarding classification because its approach to separate class regions is very different from that of MLPs, as explained below. Hence, the selection of one or the other of these classifiers for a given dataset can sometimes be of great concern.

RBF networks are feedforward neural networks that usually have a single hidden layer. There are no weights associated with the connections between the input and hidden layers. The units of the hidden layer use *radial basis functions* (RBFs) instead of the sigmoidal functions used in MLPs. RBFs are a special class of functions whose value decreases or increases with respect to the distance to a central point. The commonest RBF employed in these networks is the well known Gaussian function. When this function is used, each

hidden unit computes its output R_i by

$$R_i(\overrightarrow{x}) = \exp\left(-\frac{||\overrightarrow{x} - \overrightarrow{r_i}||^2}{\sigma_i^2}\right) \tag{2.8}$$

where $\overrightarrow{x}$ is the input vector, $||\overrightarrow{x} - \overrightarrow{r_i}||$ is the Euclidian distance between the input vector $\overrightarrow{x}$ and the Gaussian center $\overrightarrow{r_i}$, and σ_i is the Gaussian width.

Notice that instead of computing the internal product between the input vector and the respective weights, as in MLPs, RBF hidden units compute the Euclidian distance between the input vector and the Gaussian center and, next, compute their output by using a Gaussian function. This is carried out in order to transform the non-linearly separable problem into a linearly separable problem. In this way, the output layer of the RBF network has to address a much simpler problem and therefore linear neurons are usually employed in this layer. Hidden neurons are connected to output neurons by weighted connections. Each linear output neuron computes its output $f(\overrightarrow{x})$ as

$$f(\overrightarrow{x}) = \sum_{i=1}^{m} A_i \times R_i(\overrightarrow{x}) \tag{2.9}$$

where m is the number of hidden RBFs and A_i is the weight of connection i.

The justification for the operation of RBF networks can be found in Cover's separability theorem [Hay98], which states that a non-linear pattern classification problem in a high dimensionality space has greater probability of being linearly separable in this space than in a low dimensionality one. RBF neural networks usually have great numbers of hidden units (when compared to MLPs) in order to increase the space dimensionality. This is carried out in order to make the problem linearly separable in this novel space. This idea is also employed in support vector machines, a recent class of powerful classifiers [STC04, Hay98, A.03, MLH03].

RBF and MLP partition the training space in very different ways. This is due to the different activation functions used in the hidden layer of these architectures. The radial basis functions used by RBF networks group similar data in clusters and partition training space by using closed regions. The left graphics of figure 2.6 illustrates such partitioning for an RBF with four hidden units and a training set with three classes. On the other hand, MLPs partition the training space using hyperplanes, as discussed before. Thus, the same training space would be partitioned by an MLP with one hidden layer as shown in the right graphics of figure 2.6.

A number of different techniques have been proposed for training RBF networks. Most training methods have two phases: in the first phase the number of hidden units and their parameters are determined. This is usually carried out by using unsupervised methods, including *k-means-clustering* and the *self-organizing map* neural network (SOM) [BLC00]. In the second phase, the weights of the connections between hidden and output units are adjusted. This phase can be done using a supervised error correction rule similar to that used for training the perceptron. Other techniques for training RBF networks include methods based on genetic algorithms [LCL03] and based on Kalman filtering [Sim02].

In this work, RBF networks are trained using the dynamic decay adjustment algorithm (DDA), a constructive training algorithm for RBF [BD95] and probabilistic neural

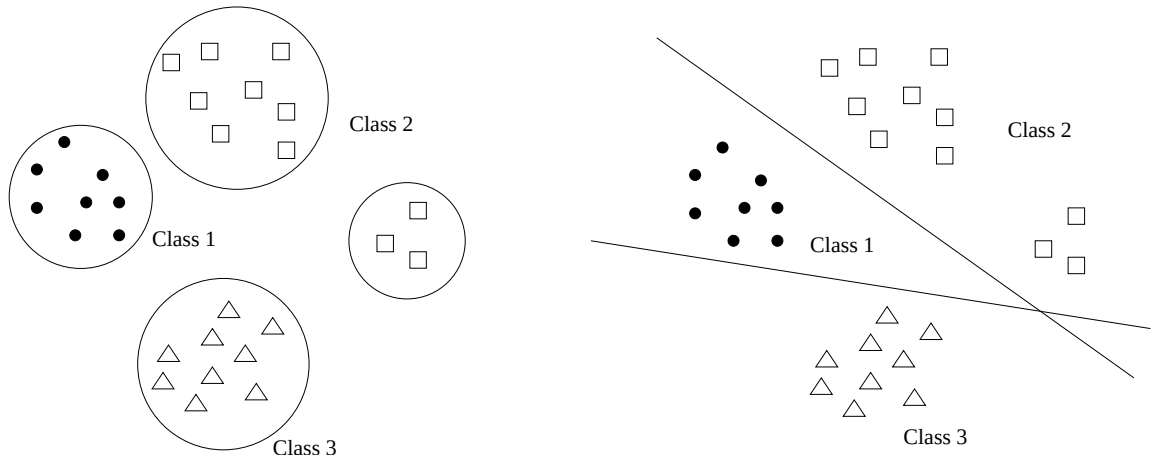


Figure 2.6. Comparing RBF to MLP partitioning of training data. Left: RBF, right: MLP.

networks (PNNs) [BD98]. This algorithm is shown to perform better than previous ones on a number of datasets [BD95]. We also employ a number of extensions to RBF-DDA introduced in chapter 4 of this thesis for training RBF networks for novelty detection in time series.

2.2.2.1 The Dynamic Decay Adjustment Algorithm (DDA) The DDA algorithm is a very fast constructive training algorithm for RBF and probabilistic neural networks (PNNs) [BD95, BD98, HM98, LSP$^+$96]. In most problems training is finished in only four to five epochs. This algorithm was inspired by an older constructive training algorithm called RCE (*Restricted Coulomb Energy*) [Hud92]. Experimental results have shown that DDA outperforms RCE and other training algorithms for RBF networks in a number classification tasks [BD95]. The DDA algorithm was also adapted to be used for training TDRBF (*Time Delay RBF networks*). TDRBF adds temporal processing to RBFs and are used in speech recognition and other applications [Ber94, VHB02, HB02, HB98]. The advantages of RBF-DDA have motivated its implementation in hardware [LSP$^+$96, HM98, KS02]. In addition, extensions to the method have been recently proposed in the literature [Pae04, ONM04c, ONM04b, OMNM04]. A recent extension to DDA, referred under the name of RBF-DDA-T, was proposed in the literature aiming to reduce the complexity of the networks built by DDA [Pae04]. This thesis also introduces four different methods for improving RBF-DDA performance [ONM04c, ONM04b, OMNM04] (see chapter 4).

An RBF trained by DDA is referred as RBF-DDA. Figure 2.7 depicts a typical RBF-DDA architecture. The number of units in the input layer represent the dimensionality of the input space. The input layer is fully connected to the hidden layer. RBF-DDAs have a single hidden layer. The number of hidden units is automatically determined during training. Hidden units use Gaussian activation functions. RBF-DDA uses 1-of-n coding in the output layer, with each unit of this layer representing a class. Classification uses a winner-takes-all approach, whereby the unit with the highest activation gives the class. Each hidden unit is connected to exactly one output unit. Each of these connections has

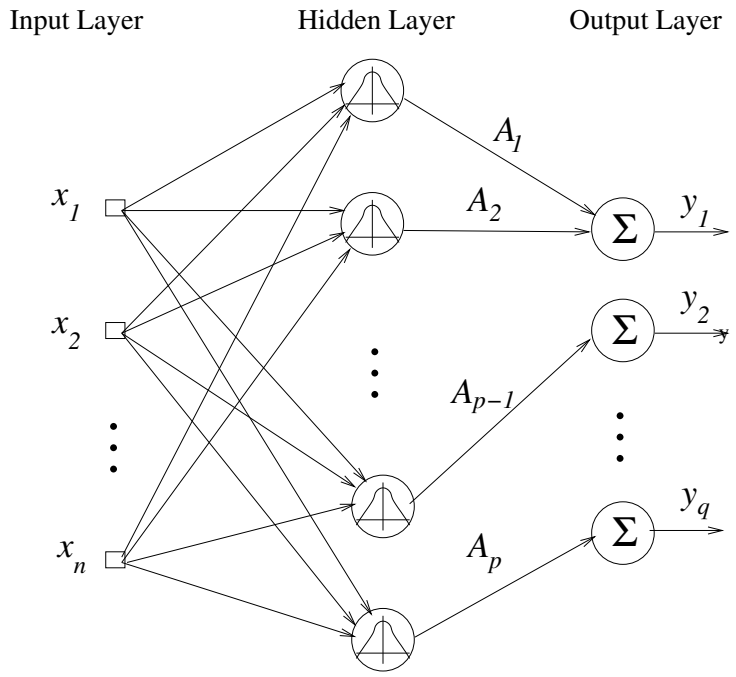


Figure 2.7. RBF-DDA neural network architecture

a weight A_i. Output units uses linear activation functions with values computed by

$$f(\overrightarrow{x}) = \sum_{i=1}^{m} A_i \times R_i(\overrightarrow{x}) \tag{2.10}$$

where m is the number of RBFs connected to that output.

The DDA training algorithm is constructive, starting with an empty hidden layer, with units being added to it as needed. The centers of RBFs, $\overrightarrow{r_i}$, and their widths, σ_i are determined by DDA during training. The values of the weights of connections between hidden and output layers are also given by the DDA algorithm.

The DDA algorithm relies on two parameters in order to decide about the introduction of new prototypes (RBF units) in the networks. One of these parameters is a *positive threshold* (θ^+), which must be overtaken by an activation of a prototype of the same class so that no new prototype is added. The other is a *negative threshold* (θ^-), which is the upper limit for the activation of conflicting classes [BD95, BD98]. The use of two thresholds results in improved classification in areas where the algorithm did not introduce new prototypes. An example is depicted in figure 2.8. In this example, a new input pattern of class B correctly results in activations above the positive threshold for the correct class B and below the negative threshold for conflicting class A.

A trained RBF-DDA network holds the following two equations for every training pattern $\overrightarrow{x}$ of class c [BD95, BD98]:

$$\exists i : R_i^c(\overrightarrow{x}) \geq \theta^+ \tag{2.11}$$

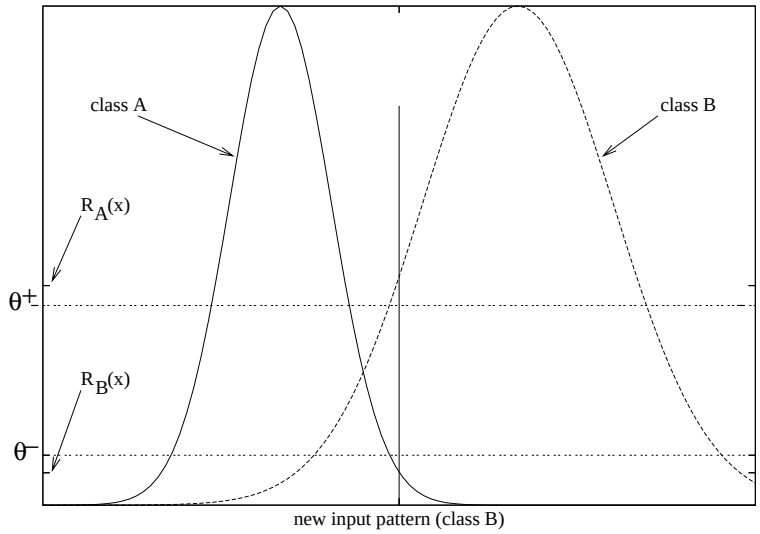


Figure 2.8. Classification of a new pattern of class B by an RBF-DDA network

$$\forall k \neq c, 1 \leq j \leq m_k : R_j^k < \theta^- \tag{2.12}$$

Notice that the above conditions do not guarantee the correct classification of all training patterns, because they hold for hidden units, not for output units.

The DDA algorithm for one training epoch is shown in algorithm 1 [BD95]. This algorithm is executed until no changes in the parameters values (number of hidden units and their respective parameters and weights values) are detected. This usually takes place in only four to five epochs [BD95]. This natural stopping criterion leads to networks that naturally avoid overfitting training data [BD95, BD98]. Notice that, in each epoch, the algorithm starts by setting all weights to zero because otherwise they would accumulate duplicate information about training patterns.

During training, the DDA algorithm creates a new prototype for a given training pattern $\overrightarrow{x}$ only if there is no prototype of *the same class* in the network whose output $R_i(\overrightarrow{x}) \geq \theta^+$. Otherwise, the algorithm only increments the weight A_i of the connection associated with one of the RBFs (of the same class of the training pattern) which gives $R_i(\overrightarrow{x}) \geq \theta^+$ (step 3 of algorithm 1). When a new prototype is introduced in the network, its center will have the value of the training vector $\overrightarrow{x}$ and the weight of its connection to the output layer is set to 1 (step 4 of algorithm 1). The width of the Gaussian will be chosen in such a way that the outputs produced by the new prototype for existing prototypes of conflicting classes is smaller than θ^- (step 5 of algorithm 1). Finally, there is a *shrink phase*, in which the widths of conflicting prototypes are adjusted to produce output values smaller than θ^- for the training pattern $\overrightarrow{x}$ (step 6 of algorithm 1).

Figure 2.9 presents a small example of the DDA algorithm in action. The first pattern from the training set is from class A. The DDA algorithm creates a prototype (RBF) centered on this pattern, as shown in (1). The next pattern is from class B, as shown in (2). This leads to the introduction of a new prototype for class B and also to shrinking

1: *// reset weights:*
FORALL prototypes p_i^k **DO**
  $A_i^k = 0.0$
ENDFOR
2: *// train one complete epoch*
FORALL training pattern $(\overrightarrow{x}, c)$ **DO**
  IF $\exists p_i^c : R_i^c(\overrightarrow{x}) \geq \theta^+$ **THEN**
3: *// sample covered by existing neuron of the same class*
    $A_i^c += 1.0$
  ELSE
4: *// "commit": introduce new prototype*
    add new prototype $p_{m_c+1}^c$ with:
    $\overrightarrow{r}^c_{m_c+1} = \overrightarrow{x}$
    $A^c_{m_c+1} = 1.0$
    $m_c += 1$
5: *// adapt radii*
    $\sigma^c_{m_c+1} = \max_{k \neq c \wedge 1 \leq j \leq m_k} \{\sigma : R^c_{m_c+1}(\overrightarrow{r}^k_j) < \theta^-\}$
  ENDIF
6: *// "shrink": adjust conflicting prototypes*
  FORALL $k \neq c, 1 \leq j \leq m_k$ **DO**
    $\sigma_j^k = \max\{\sigma : R_j^k(\overrightarrow{x}) < \theta^- \}$
  ENDFOR
ENDFOR

Algorithm 1. DDA algorithm for RBF training

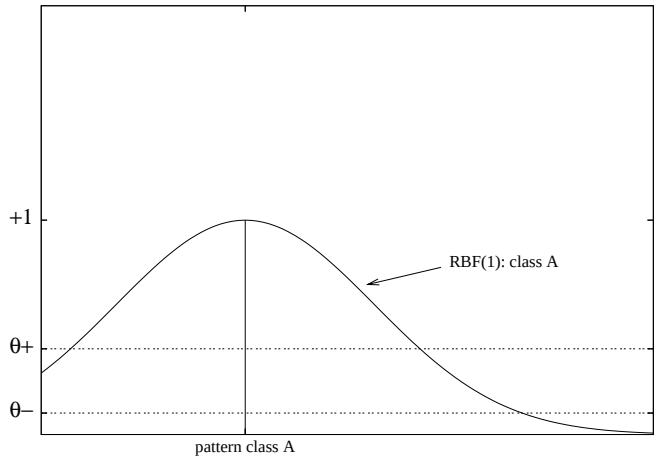
the previous class A prototype. In (3) the third training pattern, also from class B, is presented. The DDA does not introduce a new prototype since there is already a prototype whose output for this new pattern is greater than θ^+. Nevertheless, shrinking the prototype of class A is needed because otherwise it would produce an output value greater than θ^- for a pattern of class B. Finally, in (4) a new prototype for class A is introduced because the output of the existing prototype of this class is smaller than θ^+ for the fourth training pattern.

Experimental results using some datasets have shown that the generalization performance of RBF-DDA depends only slightly on the values of the parameters θ^+ and θ^-. This has lead Berthold and Diamond, who proposed DDA for RBF and PNN, to believe that these parameters were not critical for classification performance [BD95, BD98]. Therefore, they recommend using the default values of these parameters, $\theta^+ = 0.4$ and $\theta^- = 0.1$ (for RBF-DDA) [BD95] or $\theta^- = 0.2$ (for PNN-DDA) [BD98] for any dataset. We have observed, however, that for some important datasets, such as image recognition ones, smaller values of θ^- lead to a considerable improvement in performance. This has motivated the proposal of methods for improving RBF-DDA performance. These methods are introduced in chapter 4 of this thesis.

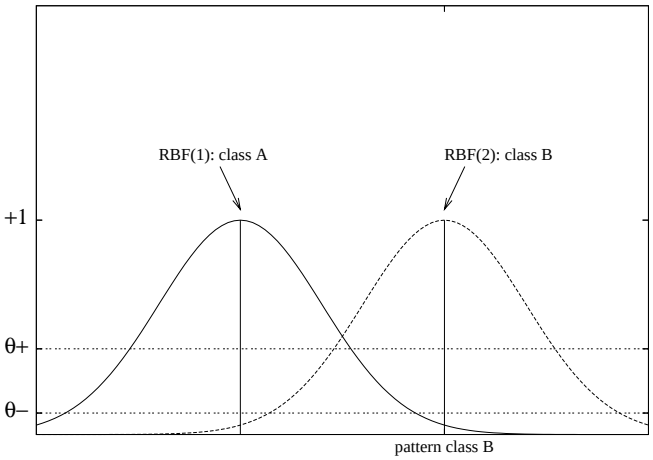
2.2.2.2 RBF-DDA-T: A Method for Reducing the Number of Hidden Units of RBF-DDA RBF-DDA-T is a constructive training algorithm for RBF networks recently proposed by Paetz [Pae04]. The algorithm is an extension of DDA and aims at reducing the number of hidden neurons produced by DDA. Paetz argues that DDA introduces too many neurons because many neurons are inserted for noisy, overlapping data or for outliers. The idea of RBF-DDA-T consists in reducing the number of hidden neurons through on-line pruning. RBF-DDA-T modifies the original DDA algorithm by pruning neurons which are considered superfluous *after each training epoch.* [Pae04]. This is done during training by marking neurons as *temporary* neurons as long as they do not cover a sufficient number training patterns. After each training epoch temporary neurons are pruned and their centers are not used anymore as training data in the following learning epochs. The new algorithm with on-line pruning is called RBF-DDA with temporary neurons (RBF-DDA-T) [Pae04].

The RBF-DDA-T algorithm for one training epoch is shown in algorithm 2. RBF-DDA-T introduces a new parameter, ν_c, which is a threshold (per class) for marking a neuron as temporary. However, this parameter was fixed for all classes in all experiments, $\nu_c = 2$. This means that to avoid pruning during training, a neuron must cover at least two training patterns. RBF-DDA-T modifies the original RBF-DDA algorithm in steps 3 and 4 and introduces an additional step (step 7). Steps 1, 2, 5 and 6 are identical to the original DDA algorithm. In RBF-DDA-T, a new prototype (neuron) introduced in the network is marked as temporary, as shown in step 4 of the algorithm. Step 3 is modified such that now if a neuron covers more than one training sample it is marked as permanent. At the end of each training epoch, every neuron marked as temporary is removed from the network (step 7 of the RBF-DDA-T algorithm).

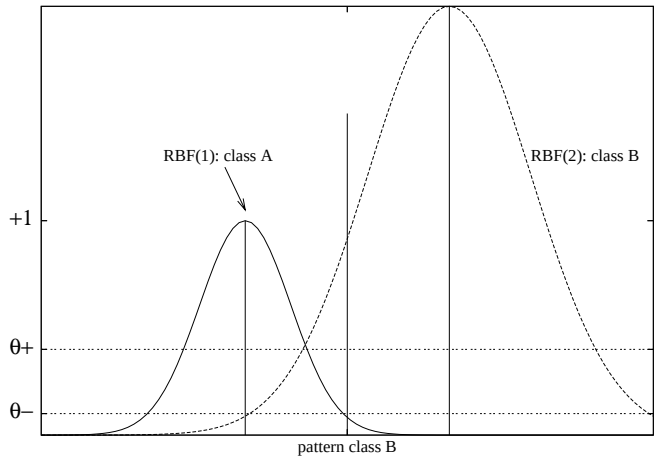
Paetz reports a number of experiments using benchmark datasets from the UCI machine learning repository [BM98] as well as a medical dataset. He concluded that the



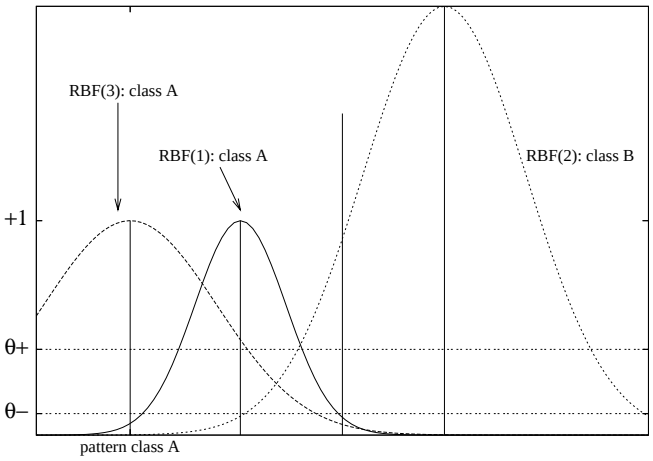
(1) First pattern: class A



(2) Second pattern: class B



(3) Third pattern: class B



(4) Fourth pattern: class A

Figure 2.9. Training with the DDA algorithm.

1: *// reset weights:*
FORALL prototypes p_i^k **DO**
  $A_i^k = 0.0$
ENDFOR
2: *// train one complete epoch*
FORALL training pattern $(\overrightarrow{x}, c)$ **DO**
  IF $\exists p_i^c : R_i^c(\overrightarrow{x}) \geq \theta^+$ **THEN**
3: *// sample covered by existing neuron of the same class*
    $A_i^c += 1.0$
    IF $A_i^c \geq \nu_c$ **THEN**
      $\text{temp}(p_i^c) = \text{FALSE}$ *// permanent neurons*
    ENDIF
  ELSE
4: *// "commit": introduce new prototype*
    IF $\overrightarrow{x} \notin OutlierList$ **THEN**
      add new prototype $p_{m_c+1}^c$ with:
      $\overrightarrow{r}^c_{m_c+1} = \overrightarrow{x}$
      $A_{m_c+1}^c = 1.0$
      $m_c += 1$
5: *// adapt radii*
      $\sigma_{m_c+1}^c = \max_{k \neq c \wedge 1 \leq j \leq m_k} \{\sigma : R_{m_c+1}^c(\overrightarrow{r}^k_j) < \theta^-\}$
    ENDIF
  ENDIF
6: *// "shrink": adjust conflicting prototypes*
  FORALL $k \neq c, 1 \leq j \leq m_k$ **DO**
    $\sigma_j^k = \max\{\sigma : R_j^k(\overrightarrow{x}) < \theta^- \}$
  ENDFOR
ENDFOR
7: *// delete neurons that are not recognized as permanent*
  FORALL neurons p_i^s **DO**
    IF $\text{temp}(p_i^s)$=TRUE **THEN**
      insert z_i^s in $OutlierList$
      delete neuron p_i^s
      $m_s = m_s - 1$
    ENDIF
  ENDFOR

Algorithm 2. RBF-DDA-T training algorithm for one epoch

RBF-DDA-T algorithm clearly reduces the complexity of the networks. The number of neurons in the datasets considered are reduced are reduced by 57.7% on average (up to 93.9% in the DNA dataset) [Pae04]. The reduction in complexity results in a slight degradation in performance for some datasets and a slight improvement for others [Pae04]. It is important to stress that RBF-DDA-T does not improve generalization performance with respect to the original DDA algorithm.

Paetz has not investigated the influence of parameter θ^- on the performance of his algorithm. In all experiments reported, he used $\theta^+ = 0.4$, and $\theta^- = 0.2$ [Pae04], which are the default values for PNN-DDA proposed by Berthold and Diamond [BD98]. This is an important issue since we have observed that θ^- considerably influences RBF-DDA performance for some datasets (see chapter 4 of this thesis and [ONM04c, ONM04b, OMNM04]). In chapter 4 of this thesis we investigate the influence of θ^- on the performance of RBF-DDA-T and conclude that smaller values unfortunately strongly degrades generalization performance because of the strong pruning strategy adopted by RBF-DDA-T.

2.2.3 Committee Machines

Committee machines are a class of classifiers built by combining a number of individual classifiers [Hay98, DHS00, A.W02]. There are a number of different ways to combine classifiers. These ways fall into two categories: *static structures* and *dynamic structures* [Hay98]. In the former case the outputs of the classifiers are combined by a mechanism that does not involve the input signal. This category includes committee machines based on *ensemble mean* and *boosting*. In dynamic structures, the input signal is involved in the mechanism used to integrate the individual outputs of the classifiers. This category includes *mixture of experts* and *hierarchical mixture of experts*. This work uses only committee machines based on *ensemble means* and therefore this section presents only this kind of committee machines.

The motivation for combining classifiers is to try to improve the overall classification performance. For example, the weights of an MLP obtained by training depend on the initial values used. Therefore, performance of an MLP depends on the initial values of its weights. Hence, in experimental works the mean and standard deviation of the performance measures, such as the classification error, should be reported. It is common to perform such experiments from 10 to 30 times to obtain the means and standard deviations.

An alternative consists in building a committee machine by combining these MLPs trained with the same training set but with different initial weights. Ensemble means can be used for this purpose. Suppose that an ensemble mean classifier is built by combining n trained MLPs. It is important to stress that each MLP is trained individually and independently. For each MLP j, suppose that the value of output i is y_{ij}. Given a pattern p in the test set, the output i of the committee classifier built by ensemble means is computed by summing the outputs of the individual MLPs, giving $\sum_{j=1}^{n} y_{ij}$. Next, pattern p is classified by the committee machine according to the winner-takes-all criterion [Hay98]. This procedure is carried out for each pattern in the test set, in order to compute the classification error in this set. Experiments have shown that ensemble

mean committee machines of MLPs can enhance classification performance in many cases [Hay98].

Classifiers of different natures can also be combined using ensemble means [DHS00]. In this thesis, for example, we build ensemble classifiers by combining MLPs and RBF-DDA (see chapter 5) [ONM04a]. This can improve performance with respect to individual MLP or RBF classifiers in some problems because these classifiers partition the training space in very different ways, as discussed before (and illustrated in figure 2.6).

In order to integrate the information given by MLP and RBF-DDA classifiers, it is necessary initially to transform their results. This is needed because RBF-DDA outputs can be greater than 1, because of the linear activation functions employed by output neurons. On the other hand, MLPs with sigmoid logistic activation functions produce outputs from 0 to 1. The *softmax* transformation is used for this purpose [DHS00]. Suppose that the neural networks have o outputs. Firstly, the results of MLPs are combined to form the MLP committee machine using ensemble means, as described previously. Next, the softmax transformation

$$g_i = \frac{\exp(y_{ij})}{\sum_{i=1}^{o} \exp(y_{ij})} \tag{2.13}$$

is applied to each committee output i [DHS00]. The softmax transformation is also applied independently to the outputs of the trained RBF-DDA. Finally, the softmax transformed outputs of the MLP committee and those of the RBF-DDA are combined to obtain an ensemble mean. This is done, for each output, simply by summing the softmax transformed value produced by the MLP committee to the corresponding softmax transformed value of RBF-DDA. Finally, the winner-takes-all criterion is applied for classification of a given pattern.

2.3 SUPPORT VECTOR MACHINES

Support vector machines (SVMs) are a more recent class of feedforward networks used for pattern classification, function approximation and time series forecasting [CV95, Vap95, STC04, CST00, A.03, Hay98, A.W02, MSR+97]. SVMs consists in an implementation of the method of structural risk minimization [Vap95, Hay98]. The learning algorithm of SVMs can be used to build a number of different learning machines, including polynomial learning machines, RBFs and MLPs. The number of hidden units in all of these cases is automatically determined by the SVM learning algorithm, which can be an advantage in practical applications.

Figure 2.10 illustrates the hyperplane built by a support vector machine for the case of a classification problem in two dimensions with two classes. This problem can be solved by the perceptron, as discussed in section 2.2.1. The solution provided by the perceptron depends on the initial values of the weights and bias. On the other hand, an SVM builds an *optimal hyperplane* to separate training patterns for classes 1 and 2, as shown in figure 2.10. The optimal hyperplane is the one that maximizes the *margin of separation* between hyperplanes defined by the *support vectors* of each class. These vectors correspond to the patterns that are nearest to the decision boundary and therefore are the most hard to

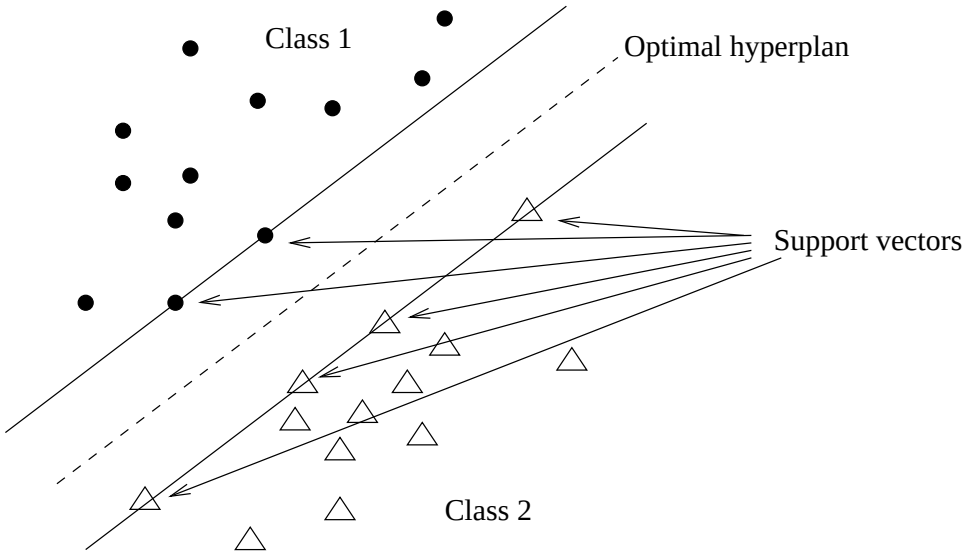


Figure 2.10. The optimal hyperplane built by an SVM for linearly separable patterns

classify. Hence, they have a direct influence on the optimal localization of the hyperplane, as illustrated in figure 2.10.

In order to use an SVM for non-linearly separable classification problems it is necessary firstly to carry out a non-linear mapping to a feature space of high dimensionality. This is done because Cover's separability theorem, already commented in section 2.2.2, states that a non-linear pattern classification problem in a high dimensionality space has greater probability of being linearly separable in this space than in a low dimensionality space [Hay98]. The optimal hyperplane is then built in the feature space of higher dimensionality. Notice that this procedure does not guarantee that the error on the training set will be zero. The objective of SVMs is to minimize the error on the training set and, at the same time, control the complexity of the model via a regularization term similar to that of the RpropMAP (see section 2.2.1). The optimal hyperplane is found by using the method of Lagrange multipliers [STC04, CST00, Hay98].

Despite its popularity in the machine learning and pattern recognition communities, a recent study has shown that simpler methods, such as k-NN and neural networks, can achieve performance comparable to or even better than SVMs in some classification and regression problems [MLH03].

The main idea of support vector machines is to built optimal hyperplanes - that is, hyperplanes that maximize the margin of separation of classes - in order to separate training patterns of different classes. This was illustrated for the two classes case in figure 2.10. An SVM minimizes the first equation below subject to the condition specified in the second equation

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i (w^T \phi(x_i) + b) \geq 1 - \xi_i, \end{aligned} \tag{2.14}$$

$$\xi_i \geq 0.$$

The training vectors x_i are mapped into a higher (maybe infinite) dimensional space by the function ϕ. Then SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. A kernel $K(\overrightarrow{x}, \overrightarrow{y})$ is an inner product in some feature space, $K(\overrightarrow{x}, \overrightarrow{y}) = \phi^T(\overrightarrow{x})\phi(\overrightarrow{y})$. A number of kernels have been proposed in the literature [STC04, CST00, A.03, A.W02]. In this work we use the radial basis function (RBF) kernel, which is the kernel used more frequently. The kernel function $K(x_i, x_j)$ in an RBF kernel is given by $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2), \gamma > 0$.

SVMs with RBF kernels have two parameters, namely, C, the penalty parameter of the error term ($C > 0$) and γ, the width of the RBF kernels. These parameters have great influence on performance and therefore their values must be carefully selected for a given problem. In this work, model selection is carried out by 5-fold cross-validation on training data. A grid search procedure on C and γ is performed, whereby pairs of (C, γ) are tried and the one with the best cross-validation accuracy is selected [HCL04]. A practical method for identifying good parameters consists in trying exponentially growing sequences of C and γ. In the experiments carried out in this thesis with SVMs, the sequence used was $C = 2^{-5}, 2^{-3}, \cdots, 2^{15}$, and $\gamma = 2^{-15}, 2^{-13}, \cdots, 2^{3}$ [HCL04].

2.4 TIME SERIES FORECASTING

This section reviews some techniques used for forecasting time series. Initially two of the so-called classical techniques are briefly presented, namely, exponential smoothing and ARIMA. Next, two neural networks architectures that can be used for time series forecasting are presented: multi-layer perceptrons (MLPs) and Elman networks.

2.4.1 Classical Techniques

2.4.1.1 Exponential Smoothing Exponential smoothing is a simple forecasting technique originating for moving averages. Despite its simplicity, this technique can achieve good results for some time series [Cha89]. In moving averages, the value of the time series at time $t + 1$ is predicted as the average of the previous w values of the series. For example, for $w = 3$, given a time series $x_1 \cdots x_l$, x_{l+1} is predicted by $\hat{x}_{l+1} = (x_{l-2} + x_{l-1} + x_l)/3$. In practice the data points closer to $l + 1$ can have greater influence on x_{l+1}. Therefore, the weighted moving averages is more reliable. In this case, x_{l+1} is predicted by $\hat{x}_{l+1} = (w_1 x_{l-2} + w_2 x_{l-1} + w_3 x_l)/3$

Exponential smoothing is a kind of weighted moving average. The original exponential smoothing – referred as simple exponential smoothing or single exponential smoothing – assumes that the means of the time series varies slowly in time. In single exponential smoothing, the original time series $x_1 \cdots x_l$ is used to build a smoothed version of it, $S_2 \cdots S_l$. The smoothed time series is obtained by

$$S_t = \alpha \times x_t + (1 - \alpha) \times S_{t-1} \tag{2.15}$$

where $0 < \alpha \leq 1$, and $t \geq 3$. α is referred as the *smoothing constant* [Cha89]. Notice that the smoothed series does not has the first value, S_1. The smoothed time series is used to

forecast the value of the original series in time $l+1$, that is, $\hat{x}_{l+1} = S_{l+1}$.

The choice of the initial value of the smoothed series S_2 can affect prediction performance. There are a number of alternatives for choosing S_2, including setting $S_2 = x_1$. Another common alternative consists in using the average of the first four to five observations of the original series as the value of S_2.

The value of α can also greatly influence prediction performance. The optimal value of α depends on the time series. Given a time series, one should select the value of α that minimizes the *mean squared error* (MSE) on the known time series data points. One of the techniques commonly used for this purpose consists in a trial-and-error method. This is an iterative procedure beginning with a range of α between 0.1 and 0.9. The best initial choice for α is determined – minimizing the MSE – and next a more refined search can be carried out between $\alpha - \delta$ and $\alpha + \delta$. This procedure can be repeated perhaps one more time in order to find the best α to 3 decimal places. Alternatively, a more powerful optimization technique such as the Marquardt algorithm can be used to find the optimal α. This is a nonlinear optimizer that minimizes the sum of squares of residuals.

There are also variations of exponential smoothing for time series with a trend (referred as double exponential smoothing) as well as for time series with both trend and seasonality (referred as triple exponential smoothing). In the former variation, there is a second equation with a second parameter γ that deals with trends. The triple exponential smoothing introduces also a third equation with a third parameter β to take care of seasonality.

2.4.1.2 ARIMA Auto-Regressive Integrated Moving Average (ARIMA) time series models consist in a general class of linear models that are widely used in modeling and forecasting time series [BJ76, Cha89, Mas95]. ARIMA(p, d, q) models combine an autoregressive part AR(p), an integrating part I(d), and a moving average part MA(q). An autoregressive model of a time series x_t (AR(p)) assumes that the current value of the series is equal to a weighted average of a finite number of previous values, plus a constant offset ξ, plus a random noise term, a_t. The AR(p) model is expressed in equation 2.16.

$$x_t = \xi + \phi_1 \times x_{t-1} + \phi_2 \times x_{t-2} + \phi_3 \times x_{t-3} + \cdots + a_t \tag{2.16}$$

The MA(q) part of the model states that the the current value of a time series is equal to a weighted sum of a finite number of previous random noise terms $(a_{t-1}, a_{t-2}, \cdots, a_{t-n})$, plus a constant offset, plus the current value of the random noise, a_t. This is expressed in equation 2.17. By definition, the noise terms are identically distributed and are assumed to have zero mean and finite variance.

$$x_t = \xi + \theta_1 \times a_{t-1} + \theta_2 \times a_{t-2} + \theta_3 \times a_{t-3} + \cdots + a_t \tag{2.17}$$

The autoregressive part (AR) together with the moving average part (MA) of ARIMA form a class of models known as ARMA(p, q). The number of AR terms (not counting the constant offset) is traditionally called p, and the number of MA terms is traditionally called q. Thus, for example, an ARMA(2,1) model would make use of the prior two values

of the series and one prior value of the random noise. These models are adequate only for stationary time series.

A time series is said to be *strictly stationary* if its statistical distribution does not change across time. Given m samples of a time series made at times t_1 through t_m, not necessarily contiguous times. The time series is strictly stationary if the joint probability density function of those m samples is identical to the joint distribution of another m samples taken at times $t_1 + k$ to $t_m + k$. This must be true for all choices of m and k, as well as choices of the m relative sample times.

If the time series is not stationary, it must be made stationary before using an ARMA model. After forecasting, the reverse operation must be done in order to obtain forecasts of the original time series. *Differencing* is the most common way of making a non-stationary series stationary. It is a very simple operation with the advantage that the reverse operation, called *integrating* is also very simple. Given a time series $\{x_1, \ldots, x_N\}$, a differenced time series $\{y_2, \ldots, y_N\}$ is formed by $y_t = x_t - x_{t-1}$. Note that the differenced time series does not have the first data point. For some time series, differencing must be carried out two times in order to obtain a stationary series. A third differentiation is rarely needed to achieve stationarity [Mas95].

The process of differencing a given time series d times, fitting an ARMA model, then integrating d times to return to the original domain, is often referred as ARIMA(p, q, d) [BJ76]. There are a number of methods for computing the optimal values of p, q, and d as well as for adjusting the weights of the AR and MA terms (ϕ_i's and θ_i's). For details, see [BJ76, Mas95, Cha89]. ARIMA techniques give accurate predictions for a number of time series. However, recent studies suggest that these techniques have reached their limitations in applications with nonlinearities in the data set such as stock indices [RZF94, YT00, ZPH98].

2.4.2 Neural Networks Techniques

2.4.2.1 Multi-Layer Perceptrons – MLPs Multi-layer perceptron neural networks have also been applied successfully to time series forecasting [Hay98, YT00, Mas95]. This architecture outperforms classical techniques in some kinds of time series because this is a non-linear model since it uses non-linear activation functions. This feature can help provide more accurate predictions for some kinds of time series [ZPH98, RZF94, Hay98, YT00, Mas95, KLSK96, Wan94]. This is particularly true if the time series under analysis is generated by a non-linear process. In fact, real world system are often non-linear and thus it is unreasonable to assume a priori that a given time series was generated from a linear process [ZPH98].

MLPs can be used both for univariate or multi-variate time series forecasting. In the former case, the network receives past values of the time series as inputs and provides forecasts of this time series. In the latter case, the MLP receives more than one time series as inputs and can therefore produce one prediction for each input time series, at the same time. This may be useful when there are a number of different time series and they influence each other. This may happen, for example, with accountancy time series [Kos00].

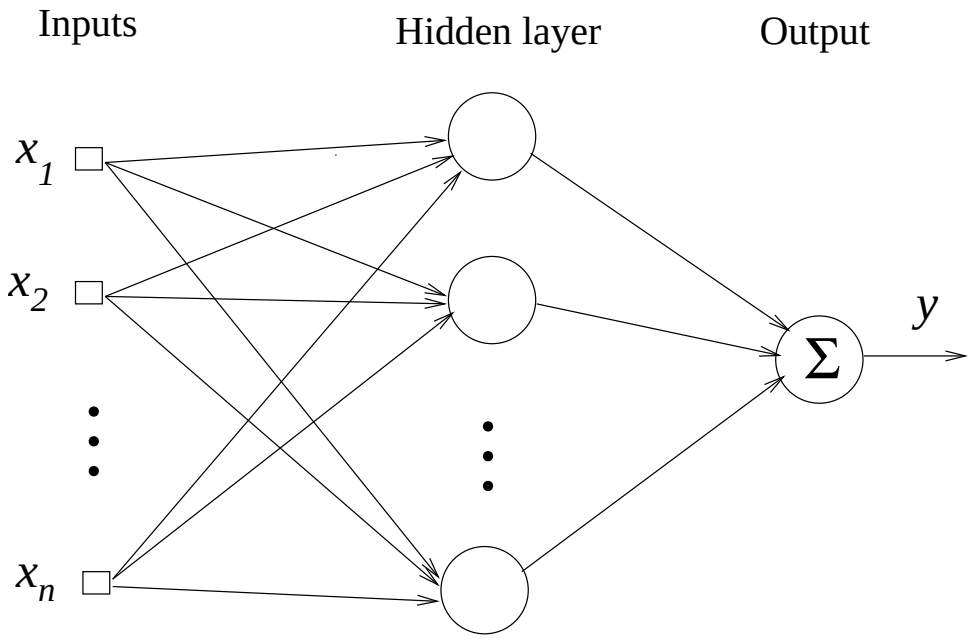


Figure 2.11. MLP network architecture for time series forecasting

This thesis addresses only the problem of univariate time series forecasting. An MLP used for this purpose is depicted in figure 2.11. The number of inputs of the network n correspond to the window size w. Usually one or two hidden layers are used. The units in these layers can use sigmoid logistic or hyperbolic tangent activations functions, in order to introduce non-linearities. There is only one output unit in this case. This unit provides the forecast value. A linear activation function is used in the output unit because this activation function does not limit the values of outputs. Recall that sigmoid logistic activation function produces outputs from 0 to 1. This is not a problem for classification tasks but can degrade performance in time series prediction applications.

A fixed window size w must be chosen in order to use MLPs for time series forecasting. If the time series has l data points then the pattern set with window size w will have $l - w$ patterns. Each pattern will have w attributes and one output. For example, if the network has three inputs (window size $w = 3$) and the time series has 10 data points, 7 patterns will be generated in order to train and test the network. The first pattern is formed by the four first data points. The second pattern will by formed by sliding the window by one and taking the next four data points. This is done to obtain all patterns. Table 2.1 shows the pattern set generated from the time series y_t in this case.

The pattern set should be divided into training and test sets. The test set is used to evaluated the generalization performance of the network. Early stopping can also be used together with the GL_5 criterion to train MLPs for time series forecasting. In this case, the training data must be divided in order to build training and validation sets.

The window size and the number of hidden units can have great impact on the prediction accuracy of the network [ZPH98]. These parameters can be determined by simple validation or cross-validation. Networks with small number of hidden units may not have

Inputs			Output
y_1	y_2	y_3	y_4
y_2	y_3	y_4	y_5
y_3	y_4	y_5	y_6
y_4	y_5	y_6	y_7
y_5	y_6	y_7	y_8
y_6	y_7	y_8	y_9
y_7	y_8	y_9	y_{10}

Table 2.1. Pattern set generated from time series y_t for training a neural network for forecasting.

the necessary number of parameters for successful prediction of a given series. On the other hand an excessive number of hidden units can lead to overfitting as in classification problems. Techniques for the optimization of the network architecture such as evolutionary computing can also be applied to improve time series forecasting [YL97].

In the example above with a time series with $l = 10$ data points and $w = 3$, suppose the network is trained with the first 5 patterns and tested with the last 2 patterns. This network can be used directly to forecast the value of the 11th data point of the time series. This is called short term forecasting. In some applications, however, long term forecasting may be needed. In this case, one needs to forecast values beyond the 11th. For example, a company may need to forecast the monthly values of its sales 12 months ahead. Long term forecasting is much more difficult than short term forecasting and the results are usually worse, as expected. In order to carry out long term forecasting with MLPs, estimated values of some or all input data points are needed. For example, in order to forecast data point y_{12} with window size $w = 3$, one needs the values y_9, y_{10} and y_{11} but y_{11} is not known. A solution to this problem is to use the value predicted by the network $\hat{y}_{11}$. This introduces an additional imprecision in the forecasting process and can lead to more inaccurate forecasting [ZPH98].

The MLP is a kind of focused TLFN (*Time Lagged Feedforward Network*), because the temporal processing is located only in the network input. This limits the use of this architecture to stationary time series [Hay98, Mas95]. Stationary time series are those whose statistics do not vary with time. In order to forecast non-stationary time series with MLPs, the time series must be transformed into a stationary time series. Differencing is the most widely transformation used for this purpose [Mas95, Cha89] as discussed before.

There are more advanced neural networks architectures. One of them is the FIR neural neural network, a distributed TLFN [Wan94]. This network won the Santa Fe time series forecasting contest [WG94]. FIR neural networks are feedforward networks with all connections substituted by FIR (finite impulse response) filters of order p [Wan94, Hay98]. This network can be used to predicted directly non-stationary time series. Recurrent neural networks are also very powerful for time series forecasting. Examples include the Jordan and Elman partially recurrent neural networks [Hay98, BLC00, Elm90]. The latter is discussed below.

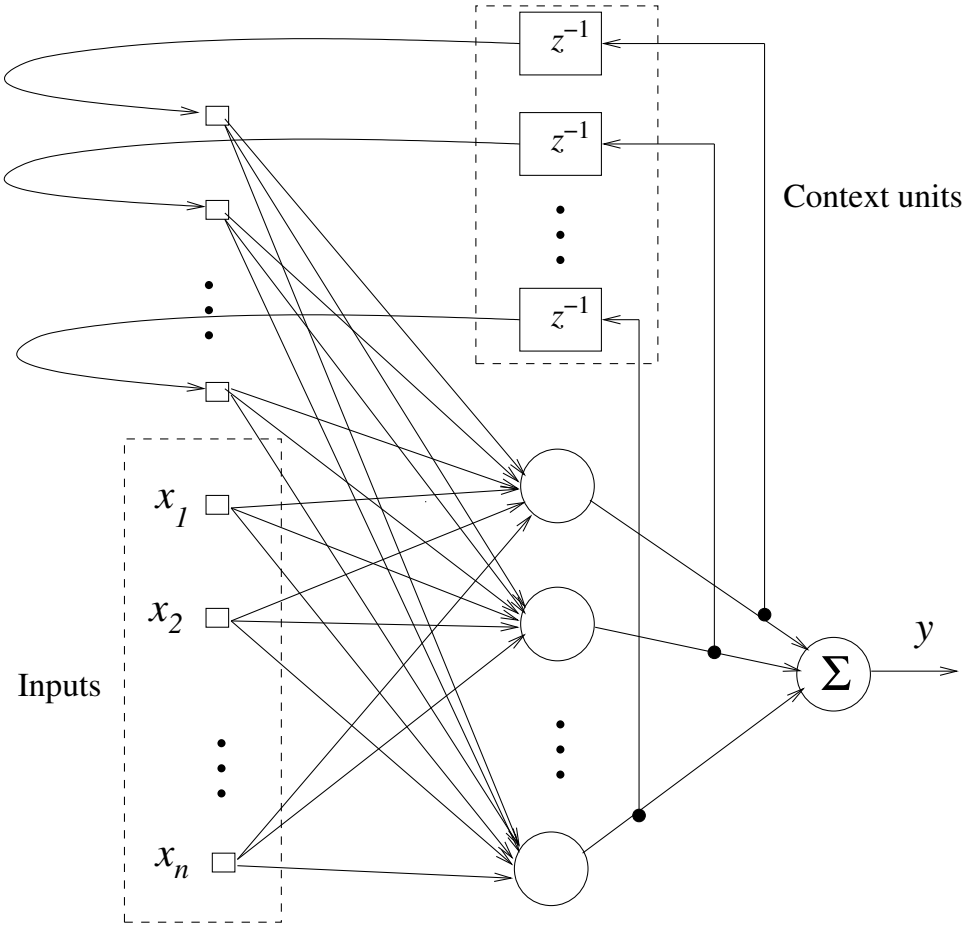


Figure 2.12. Elman neural network architecture for time series forecasting

2.4.2.2 Elman Networks Elman neural networks are partially neural networks designed for temporal processing [Elm90, Hay98, BLC00, KLSK96]. This architecture can improve forecasting performance because it introduces a number of *context units* in a context layer. Context units consists of unitary delay. They save the output values of hidden units for a time step an then feedback these values for input units. In this way, the hidden units have their previous outputs registered. This enables the network to learn tasks that extends in time.

The architecture of an Elman network for time series forecasting is depicted in figure 2.12. The network consists of an input layer whose number of units correspond to the window size w. There is usually a single hidden layer whose number of units can have great influence on performance. The activation functions of hidden units are sigmoidal functions. The output layer has a single unit that gives the prediction. The activation function of this unit is linear for the same reason of MLPs.

The pattern sets for training and test Elman networks are generated as for MLP neural networks. The training algorithms for MLPs were adapted for training Elman networks as well as other recurrent networks. There are versions of back-propagation and Rprop for training Elman networks [Zel98]. There are also training algorithms based on Kalman

filters [Hay98]. The early stopping technique can also be used to avoid overfitting during Elman network training. The number of hidden units and the window size w can also be optimized by simple validation or cross-validation.

2.5 CONCLUSIONS

In this chapter a number of classification and time series forecasting techniques were reviewed. This chapter has provided some background information on these techniques because they are used to build the novelty detection methods proposed in this work.

Two neural networks architectures commonly used for classification have been presented: multi-layer perceptrons (MLPs) and radial basis functions networks (RBFNs). A number of different algorithms for training these networks were also presented. Particular emphasis was given to the dynamic decay adjustment algorithm (DDA) for constructive training of RBF networks. The RBF-DDA-T, a recent extension to DDA proposed to reduce its number of hidden units, was also reviewed. A more detailed discussion of DDA was provided because this thesis also contributes with four methods for improving the performance of this algorithm in classification tasks. These methods are introduced in chapter 4.

This chapter has also briefly reviewed Support Vector Machines (SVMs) for classification tasks, a recent technique which has achieved outstanding performance in a number of classification tasks.

This chapter has also presented a brief overview of two classical time series forecasting techniques, namely, exponential smoothing and ARIMA. A more detailed overview of two neural networks architecture for time series forecasting – MLP and Elman networks – was provided, since these architecture are used later in this thesis in conjunction with time series novelty detection methods.

The classification techniques presented here can be used to build novelty detection algorithms for both classification tasks and time series. The time series forecasting techniques are used by time series novelty detection algorithms, such as those introduced in chapter 6. A number of such novelty detection algorithms reported in the literature are discussed in the next chapter.

CHAPTER 3

NOVELTY DETECTION METHODS

3.1 INTRODUCTION

This chapter reviews a number of novelty detection methods for both classification problems and time series. The detection of novelties, also referred as anomalies or spurious patterns, is a very important issue in many classification problems. For example, in an optical character recognition application the detection of spurious patterns, that is, input signals that do not correspond to any character in the training set, can be as important as the classification accuracy.

There are a number of methods for novelty detection in classification problems based on statistical techniques and neural networks. Different neural network architectures are used for this purpose, including feedforward networks, such as MLP and RBF, and unsupervised trained architectures such as SOM and ART networks [MS03b]. This chapter reviews only novelty detection methods based on feedforward neural networks, since these are the models used in the novelty detection methods proposed in chapter 5 of this thesis. Recent surveys are available in the literature concerning statistical methods [MS03a] and neural networks [MS03b] for novelty detection in classification tasks.

This chapter also discusses methods for the detection of novelties in time series. They are very important because many systems can be modeled by time series. Therefore, techniques can be designed to learn the normal behavior of the systems based on their time series. Later, the novelty detection systems could be used to alarm whenever an unexpected behavior takes place. This is carried out by continuously monitoring the time series. A number of techniques based on both forecasting and classification have been proposed for detecting novelties in time series. Applications of these techniques include machine failure detection, noise detection in signal processing, patient's condition monitoring, and fraud detection in financial systems.

Forecasting based time series novelty detection is one of the most straightforward methods. The past behavior of the system is supposed to represent its normal behavior and is used to train a forecasting system. This system is used subsequently to predict future values of the series. An alarm is issued whenever the difference between predicted and observed values exceeds a pre-defined threshold. This technique has been used on fraud detection in accountancy [Kos00, Kos03] and payroll [OABS03] auditing. The definition of the value for the threshold is the main difficulty of the methods based on forecasting [ONM04e].

As an alternative to forecasting-based novelty detection, a number of different methods based on classification have been proposed in the literature. This chapter reviews recent techniques based on artificial immune systems [DF96, GD02, GDK02], wavelets [STZ00], Markov models [KLC02], and one-class support vector machines [MP03]. These techniques classify a given *time series window* into normal or abnormal. They are not

able to detect novelty in a single data point from a time series. This can be done only by forecasting-based methods.

3.2 NOVELTY DETECTION IN CLASSIFICATION TASKS

This section reviews some methods for novelty detection in classification tasks. This task is also often referred as rejection of spurious patterns. Ideally, a trained neural network should classify only patterns belonging to classes available during the training phases. Random patterns (also called spurious patterns) and patterns from classes not available during training should be rejected by the network. The rejection of spurious patterns is a very important issue to be taken into account in the design of classifiers.

The standard MLP network is a very popular and successful classifier. It achieves good generalization performance on a number of tasks with respect to the classification of patterns of valid classes, that is, of classes available in the training sets. On the other hand, MLPs are unreliable with respect to the rejection of spurious patterns. These networks can accept as valid patterns with completely random appearance.

The reason for the acceptance of spurious patterns by standard MLPs is that these networks solve a classification problem by creating open decision regions in input space, as discussed in chapter 2 and illustrated in figure 2.6. On the other hand, RBF networks create closed partitions of the input space as discussed in the same chapter and illustrated also in figure 2.6. RBFs have the additional advantage of being much faster to train then MLPs. On the other hand, MLPs use much less hidden units and offer a better generalization performance than RBFs for a number of problems. The open decision regions created by MLPs explain its superior performance in some classification problems. This has motivated the investigation of a number of methods for improving the spurious pattern rejection ability of MLPs without much degradation of its classification performance.

An MLP network with sigmoid logistic functions produces outputs in the range $[0, 1]$. One of the most popular approaches for classification with this network is the winner-takes-all, whereby the class is given by the output neuron that produced the highest output (see chapter 2). The problem with this method is that all input patterns are classified, that is, there is no rejection. This is so even when an RBF network is used. A more refined classification decision method classifies a given input pattern with a *confidence level* imposed at the network's output. A pattern is classified as belonging to a class if the associated output (in the range [0,1]) for that class exceeds that of the others by a chosen confidence level; otherwise, the pattern is rejected.

The results of a handwritten recognition experiment illustrate the differences between MLP and RBF networks with respect to their spurious rejection abilities. In these experiments the networks were trained using a dataset of 1000 handwritten *digits* [Vas95]. The trained networks were tested on an independent set of 2000 digits. In addition, a second dataset with 7800 handwritten *alphabetic letters* was used to test the ability of the networks in rejecting patterns not belonging to the training classes. The results showed that the spurious pattern rejection performance of standard MLPs is indeed very poor. For example, for a 0.45 confidence level, the network was able to reject only 46% of the

entire letters dataset. On the other hand, an RBF network was able to reject 82% of these spurious patterns using the same confidence level for the classification decision. Nonetheless, the MLP generalization performance measured on the second digits dataset was much better than that of the RBF. The former network classified correctly 80.8% of the patterns whereas the latter classified correctly only 66.7% of the patterns in the test set. In both cases a 0.45 confidence level for the classification decision was used. For other values of the confidence level the same qualitative results were also observed, namely, RBF outperforms MLP regarding the rejection of spurious patterns but obtains worse classification performance.

Due to its advantages over RBFs in some aspects, mechanisms were investigated for improving MLPs with respect to the rejection of spurious patterns. Some of these mechanisms are reviewed below. The three first could also be considered for use in conjunction with RBF networks.

3.2.1 Negative Training

The negative training approach consists in training the network with an augmented training set which has the original training patterns plus a number of *negative samples* [LK89, SM92, Vas95]. This was one of the first methods proposed for improving MLPs with respect to the rejection of spurious patterns. There are basically two ways to represent negative samples in the MLP network. The first consists in adding an output unit associated with negative samples. The second maintains the network architecture and trains the network by mapping negative samples to the null output vector, that is, null output in all output units. The idea of the method is to create an attractor in the input space so that when another random pattern (novelty) is presented to the network after training it is expected that this random pattern be classified as random (novelty) instead of as a valid member of a training class.

If the negative samples are carefully chosen, it is shown that the effect of the method is to make MLPs generate close regions in the training space instead of the standard open regions that it generates without them [Vas95]. This happens only if the negative samples are selected surrounding the training classes. In this case, the network will have the information necessary to generate closed regions around the valid training classes. The problem is that in practical applications it is very difficult to select those desirable negative samples. Instead, it is common to use a large number of random negative samples. Some experiments have indicated that this can sometimes lead to improved rejection performance but in other cases this improvement does not takes place.

In the handwritten recognition experiments mentioned before, the negative training approach lead to variable performance improvements over MLP without negative samples for different confidence levels [Vas95]. For example, for a confidence level of 0.15 the letters rejection rate was improved by 11%, for a confidence level of 0.55 the improvement was by 5%, whereas for a confidence level of 0.65 there was a small decrease in the rejection rate. In addition, a small degradation in the performance of digit recognition was introduced by the negative training approach. The degradation in digit recognition performance happened for low confidence levels and was 1.8% on average. On the other hand, for the

0.65 confidence level there was an increase in digit recognition performance by around 6%.

A more recent work in the area of scene analysis has proposed a novel method for generating negative samples for novelty detection [SM04]. This method tries to generate better distributed negative samples, aiming at improving performance of the negative training approach. It is applied to a scene analysis problem, yet Singh and Markou argue that the method is generic and can improve negative training performance in other problems as well [SM04].

The proposed method trains the MLP network by assigning negative samples to the null output vector. After training, a pattern is rejected if the values produced by all output units are smaller than 0.5. Negative samples are generated as follows [SM04]:

- Find the range of each attribute per class. Define a new range to create a space to be used for generating the negative samples. The new range is defined as $(\mu - 2.5 * \sigma, \mu + 2.5 * \sigma)$, where μ and σ are the mean and standard deviation of the attribute.
- Generate a large number of negative samples within the newly created ranges on each attribute and remove those samples that lie within the distributions of known classes.
- A density thinning procedure is carried out in order to achieve a more homogeneous spread of negative samples. In this procedure, each negative sample is considered in turn and all other negative samples within a hypersphere of predefined size are removed.

3.2.2 Combined MLP and Parzen Window Approach

Another possible approach for improving MLP spurious pattern rejection performance is based on the estimation of the probability density of training data [Bis94]. This can be done by using both parametric and non-parametric techniques. The goal of an MLP training is to find a good approximation to the regression by minimization of a sum of squared errors defined over a finite training set, as discussed in chapter 2. This approximation will be more accurate in regions of the input space where the density is high, since only then does the error function penalize the network mapping if it differs from the regression. After training, if a given input pattern falls in a region of high density then the network is effectively interpolating between training data points and the network performance is expected to be good. On the other hand, if the input pattern falls in a low density region, then the input data should be rejected because the network could easily generate erroneous outputs.

The proposed procedure for novelty detection is then to compute an estimate $\hat{p}(\overrightarrow{x})$ of the unknown density $p(\overrightarrow{x})$ by using the training patterns [Bis94]. Then, when the network is in use, each new input pattern $\overrightarrow{x}$ is presented to the trained network and is also used to compute $\hat{p}(\overrightarrow{x})$. $\hat{p}(\overrightarrow{x})$ gives a quantitative measure of the degree of novelty of the input pattern. Smaller values of $\hat{p}(\overrightarrow{x})$ indicate novelties. The Parzen window non-parametric estimation technique [DHS00] was used for this purpose in some of the

experiments [Bis94]. The author also discusses other alternatives for the estimation of $p(\overrightarrow{x})$. The author states that it is important that density estimation be carried out on the original input data, before any pre-processing [Bis94].

3.2.3 Guard Units Approaches

The guard units approaches were proposed to enhance the rejection performance of MLP networks without the need of negative samples in the training phase [VFB93, Vas95]. There are two variations of this method, known as single guard units and multiple guard units. In both cases, the guard units form an integrated architecture with the MLP. They are used to provide a final "accept" or "reject" decision with respect to an input pattern after the MLP classification decision is issued. If the guard units accept the input pattern then it is classified according to the output of the MLP. Otherwise, the input pattern is rejected.

In the single guard units approach there is one guard unit for each class of the training set. Each guard unit is fully interconnected with the input layer and its output is used by the final decision mechanism in order to accept or reject the input pattern. Each connection between the input layer and the guard units has a weight. Training of the guard units is independent from MLP training. It is a fast training that requires only a single processing step over the entire training set. The weight of the guard unit associated with class i is given by the mean of the training patterns from that class. After training, the guard units operate by providing an output proportional to the similarity between their weight vectors and the input pattern. Two variations were investigated in this respect: the use of linear guard units and of Euclidean guard units [Vas95]. The former variation operates by computing the inner product between the input and weight vectors whereas the latter computes the Euclidian distance between these vectors, as in RBF units.

Single guard units do not take into consideration situations where the training data is not confined to a convex region of the pattern space. This has motivated the introduction of the multiple guard units approach [Vas95]. In this case, there are more than one guard unit per class. This allows a more refined representation of the areas in the input space occupied be the training classes. Training in the multiple guard units approach starts with the adoption of a single guard unit for each class and then, by performing a single step over the entire training set, new guard units are added whenever the criterion of similarity exceeds a pre-defined threshold. The value of the threshold controls the number of clusters formed and consequently the number of guard units defined for each training class.

Experiments carried out by using the already mentioned handwritten recognition problem have shown that the multiple guard units approach with Euclidean distance improved letter rejection performance with only small degradation in digit recognition performance [Vas95]. These effects were observed for every confidence level considered.

3.2.4 Alternative MLP Configurations

Modifications in the original MLP architecture were also investigated in order to improve the spurious rejection performance of these networks without the need for negative samples in the training phase [Vas95]. All of the modified MLP architectures try to improve the rejection performance without degrading classification performance of valid patterns. These architectures try to improve rejection performance by designing networks that create more closed regions in the input pattern space instead of the open regions generated by the standard MLP architecture. Four modified MLP architectures were considered in that study: the paraboloidal MLP network, the normalized MLP networks, MLPs with direct connections, and the Gaussian MLP network [Vas95]. The two last MLP variations have achieved the best results.

The standard MLP architecture has connections only between adjacent layers, as discussed in chapter 2. The MLP with direct connection (DMLP) has, in addition, direct connections from the input to the output layer of the network. The motivation for these additional connections is that they tend to correspond more closely to template-like representations of the training classes, instead of the normal feature based representations obtained in hidden layer(s) of the network. This makes the DMLP produce more closed decision regions in the pattern space because the response of the network's output units to the input patterns is more directly dependent on the similarity of the patterns to a template representation of the training examples.

The Gaussian MLP (GMLP) network is another alternative for the generation of closed decision regions by MLP networks. The open decision regions generated by standard MLP can be partially attributed to the sigmoid activation function used by this network. The GMLP tackles this problem by using Gaussians as activation functions. Notice that the GMLP network is different from RBF networks in that in the former the inner product is used as the propagation rule whereas the latter computes the Euclidean distance between the input and weight vectors. The GMLP is capable of approximating the same sort of functions as MLP and RBF networks and it is claimed to provide faster training times and to reduce the number of hidden units necessary for a given problem [DS92, Fla93].

Experiments with the handwritten dataset have shown that both DMLP and GMLP improve the letter rejection performance when compared to standard MLP. It is shown that these architectures and the MLP with multiple guard units achieve similar letter rejection performance on this dataset, with a slight advantage for the GMLP [Vas95]. The results were also compared to those obtained by RBF networks for a number of different confidence levels. They show that these networks achieve slightly superior performance than that of MLP with multiple guard units. RBF networks are faster to train but require a much larger number of hidden units. The GMLP and DMLP presented the best compromise in terms of memory requirements and classification performance when compared to both MLP with multiple guard units and the RBF [Vas95].

3.3 METHODS FOR NOVELTY DETECTION IN TIME SERIES

This section reviews a number of methods available in the literature for the detection of novelties in time series. Both forecasting and classification-based methods are considered here.

3.3.1 Methods Based on Forecasting

The methods based on forecasting work by building a model of the time series behavior based on its past values. This model is then used to predict future values. If the difference between predicted and observed values is above a certain threshold a novelty is assumed to have occurred. A number of different forecasting methods can be used, including those reviewed in chapter 2. As stressed in that chapter, neural network models offer the advantage of better modeling time series generated by non-linear phenomena. This novelty detection technique has been applied in auditing applications for detecting frauds in accountancy and payroll systems [Kos00, Kos03, OABS03]. In both cases neural networks were used for forecasting.

The problem with forecasting-based novelty detection is the definition of the value of the threshold to be used to detect novelties. In a work in the accountancy domain, a relative threshold was pre-defined arbitrarily [Kos00]. For example, if the relative difference between predicted and observed values is above 10% the system would detect a novelty. The problem with that approach is that it can lead to large number of false positives if the threshold is small. On the other hand, large thresholds can lead to false negatives. This problem is further discussed in chapter 6. In order to tackle this problem we have proposed to use absolute thresholds that are learned from the previous forecasting performance of the neural network [OABS03]. This idea is further explored in chapter 6 together with a new approach proposed there based on *robust confidence intervals* [ONM04e]. This new method is also based on the idea of learning the thresholds through the previous forecasting performance of the neural network.

The forecasting-based methods have an advantage over classification-based ones. The latter methods can detect novelties in a time series windows. They are not capable of telling the specific point of the time series in which the novelty has taken place. This can be achieved by forecasting-based methods and can be very important for some applications such as fraud detection applications.

3.3.2 Methods Based on Classification

This subsection reviews methods based on classification for novelty detection in time series. These methods are able to classify a given time series window into normal or novelty. Five recent methods are reviewed here, based on different techniques, namely, artificial immune systems, wavelets, suffix trees and Markov chains, and one-class support vector machines.

3.3.2.1 Methods Based on Artificial Immune Systems Artificial immune systems consists of a relatively new class of algorithms originally developed for detecting

computer viruses [FPAC94, DFH96, DAO97, Das97, HF00]. These systems were inspired on the natural immune systems of mammals. They were later applied to novelty detection in time series aiming at signal processing and machine fault detection applications [DF96, GD02, GDK02].

Artificial immune systems are inspired by the information-processing properties of natural immune systems [DAO97, Das97, HF00], which are very complex systems with several mechanisms of defense. There are various cell types that can attack invaders directly or secrete molecules with a variety of functions including attacking foreign cells and signaling other immune cells to proliferate. One of the main tasks to be solved by the natural immune system is *self-nonself discrimination*. Robust self-nonself discrimination is achieved in these systems through the *negative selection* mechanism. This mechanism eliminates self-reacting immunity cells and therefore only cells that fail to bind to self-proteins are allowed to leave the thymus (where they are created) and become part of the body's immune system.

The negative selection mechanism found in natural immune systems has inspired algorithms for change detection [FPAC94]. In this case, the *self* is defined as the normal pattern of activity of a system that one wishes to monitor. A diverse set of *detectors* is generated in the complement space of the self, the so-called *non-self*. Then the detectors are used to monitor *self* for changes by continuously matching the detectors against the representative of *self*. If any detector ever matches, a change (also referred as novelty or anomaly) must have occurred in the *self*. The negative selection algorithm can be summarized as follows:

- Define *self* as a collection S of strings of length l over a finite alphabet, a collection that needs to be monitored. For example, S may be a normal pattern of activity, which is segmented into equal sized sub-strings.
- Generate a set R of *detectors*, each of which fails to match any string in S.
- Monitor S for changes by continually matching the detectors in R against S. If any detector ever matches, then a change is known to have occurred, as the detectors are designed to not match any of the original strings in S.

There are a number of ways to generate detectors. The simplest and more expensive computationally consists in randomly generating candidate detectors and then eliminating if they match *self*. A more efficient algorithm for generating detectors has been proposed, which runs in linear time with respect to the size of self [DFH96].

In order to apply the ideas of artificial immune systems to novelty detection in time series it is necessary to define a way to represent the series. Originally, a binary codification was employed [DF96]. In this case, the time series is firstly normalized between 0 and 1. Next, each data point of the time series is converted to binary by using m bits. m may be chosen according to the desired precision. The system works by detecting novelties in a time series windows. A suitable window size w should be selected which captures regularities of interest. Each window from the time series is used to generate a string that is part of the *self*. In the experiments reported, non-overlapping windows

were used, that is, the window was slided along the series by w to obtain the next string of the *self*. This process was repeated until the end of the series. Then a set of *detectors* was generated that did not match any of the self strings. Assuming that the series represents the normal behavior, the detectors can be used subsequently to detect any changes (novelties) in patterns of unseen time series data. A time series window is firstly encoded as in the training phase in order to be fed into the novelty detection system.

This novelty detection method based on the artificial immune system has been tested on two time series, one corresponding to the cutting force signal of a machine tool used in milling industries and the other, a periodic signal common in signal processing applications. The former series is typical of machine failure detection applications. Experiments with windows sizes $w = 5$ and $w = 7$ were reported and the system was able to detect simulated novelties in both cases. The proposed method was also able to detect noise in the periodic signal from signal processing applications.

One of the problems of the original approach for novelty detection in time series based on artificial immune system is the use of the binary codification. The problem is that the procedure for discretizing and symbolizing real values in time series can potentially lose meaningful patterns in the original time series. This has motivated the proposal of a novel method based on artificial immune systems [GDK02]. This method introduces a real valued algorithm for generating the detectors. Furthermore, it combines the negative selection approach with a classifier for improving novelty detection performance. The method is in fact a kind of negative training approach (discussed in subsection 3.2.1). It introduces only a different way for generating the negative samples by using the proposed real valued negative selection algorithm. These negative samples together with the window patterns extracted from the time series are used to train a classifier. Later, the trained classifier can be used to classify a given time series window into normal or novelty. Two classifiers were considered for use in conjunction with the method:

- An MLP trained with the back-propagation algorithm.
- An evolutionary algorithm that generates fuzzy classifier rules [DG01].

This second approach based on artificial immune system was tested by using only one time series. It was also tested on a classification dataset. The first time series was the well known Mackey-Glass series, with 1000 data points. This series is generated by a non-linear delay-differential equation and exhibits chaotic behavior. The novelties used for testing were generated artificially by modifying the parameters used in the equation for generating the original series. The classification dataset used was the Iris dataset, available from the UCI repository of machine learning problems [BM98]. In this set there are three different classes and each element is described by four attributes. Three different experiments were performed. In each case one class was used as the normal class and the other two as the abnormal ones [GDK02].

Notice that the second technique for novelty detection in time series based on artificial immune systems solves one of the problems of the first one, that is, the use of binary codification. However, a potential problem is common to both techniques: they are based on negative training. As discussed in subsection 3.2.1 this means that the number of

negative samples introduced in the training phase can greatly influence novelty detection performance. Furthermore, it is argued in the literature that the method can also fail because *the negative set will go to null with the increasing diversity of the normal set* [MP03, KLC02]. This last problem can be an issue for large time series.

3.3.2.2 TSA-tree: A Method Based on Wavelets The TSA-tree is a wavelet-based tree structure proposed to improve the efficiency of multi-level trend and surprise (novelty) queries on time series [STZ00]. Each node of TSA-tree contains pre-computed trends and surprises at different levels. Each TSA node is recursively built by wavelet transforms [BGG97]. The algorithm was developed for queries at multiple levels. An example of such queries may appear, for example, related to a database containing temperature time series of various cities. The following queries could be of interest in such database:

- Find the cities where the temperature has sudden changes (daily) during the month.
- Find the cities where the temperature has sudden changes (monthly) during the last decade.

Of course in the second of these queries sudden changes within a day or two can be ignored. The authors argue that by storing pre-computed trends and surprises at different levels of abstractions by using their TSA-tree structure, multiple-level queries can be supported much more efficiently [STZ00].

Experiments were carried out using both SP500 stock prices from 1998 [ts-] and synthetic time series generated by the random walking algorithm, which is reported as a good model for mimicking stock prices [Cha89]. According to [MP03] the problem of the TSA-tree method is that it is not able to detect short novel patterns embedded in normal signals. Another work argued that the method gives a very strict definition of novelty and that this makes the method unable to detect important novelties in some time series [KLC02]. These experiments are discussed in subsection 3.3.2.3.

3.3.2.3 TARZAN: A Method Based on Suffix Trees and Markov Chain The TARZAN method is based on suffix trees and Markov chain [KLC02]. In this work a pattern is considered a surprising pattern (another name for novelty) if *the frequency of its occurrence differs substantially from that expected by chance, given some previously seen data*. The time series is firstly discretized. In this process, features are extracted from the time series windows. The best feature extraction technique may be domain dependent and possible features include the mean of the windows, the slope of the best fitting line and the second wavelet coefficient. In the experiments, the authors have used the slope of the best fitting line for feature extraction. Next, the features vector generated is converted from real to integer values. After the discretization phase, a *suffix tree* is used to organize a dictionary of words and then a Markov chain is used to help count the number of occurrences of each substring in a sequence, that is, to compute the surprising ratings of each of them. Finally, those substrings which have surprising ratings exceeding a certain user defined threshold can be returned to be examined by the user.

The TARZAN method was tested by using two time series. The results were compared to those given by the first artificial immune system method [DF96] and to those of the TSA-tree method [STZ00].

The first time series used in the experiments was a simple noisy sine wave. The methods were trained using this sine wave an tested on another noisy sine wave generated by using the same parameters, with the only exception that an artificial novelty was introduced between the 400^{th} and 432^{th} data points [KLC02]. In this interval, the period of the sine wave was halved. The simulation results have shown that the TARZAN method was able to detect this novelty without generating false positives. The artificial immune system method is probabilistic since it depends on the detectors generated. In some runs of the experiments it was able to detect the novelty whereas in others it was not. In all runs, the method has generated a number of false positives. The TSA-tree method was not able to detect this novelty.

The second series used in the experiments contained the power demand for a Dutch research facility for the entire year of 1997. The power demand was sampled over 15 minutes averages producing a time series with 35,040 points. This series presents a repeating pattern of a strong peak for each of the five weekdays, followed by relatively quiet weekends. The exceptions appeared on holidays, whose behavior was similar to weekends. The novelty detection methods were trained used only data with normal behavior, that is, weeks without holidays. Next, they were tested by using a period that included both normal and novelty weeks (that is, weeks with one or more holidays). The results have shown that TARZAN method was able to detect these novelties whereas the artificial immune system and TSA-tree methods were not [KLC02].

The main problem of the TARZAN method is also the procedure for converting the time series with real values into a symbolic string by discretization and symbolization. This procedure can lose meaningful information from the original time series [MP03]. Another possible difficulty consists in the definition of the value of the threshold above which a substring with a given surprising rating is to be considered surprising. In the TARZAN method this threshold is defined by the user [KLC02].

3.3.2.4 Method Based on One-Class Support Vector Machines One-class support vector machines were proposed for novelty detection, that is, for problems that have only one class, namely, the class of normal patterns [SWS$^+$00, CLC03]. Later, this method was applied to the problem of novelty detection in time series [MP03]. The idea of one-class SVMs consists in building a hyperplane $f(\overrightarrow{x})$ in the higher dimensional feature space in order to separate as many as possible of the mapped vectors from the origin in the feature space. Then, a pattern $\overrightarrow{x}$ is assumed to be a novelty if $f(\overrightarrow{x}) < 0$. A regularization parameter ν directly determines the sensitivity of this novelty detection algorithm [SWS$^+$00, MP03].

In order to detect novelties in time series using the one-class SVM method it is necessary firstly to convert the time series into a set of vectors. One of the ways for doing this is the same used for time series forecasting with MLP networks presented in chapter 2. This method is referred also as unfolding the time series into a *phase space* [MP03]. The size of the vectors correspond to the time series windows chosen. In addition, a different

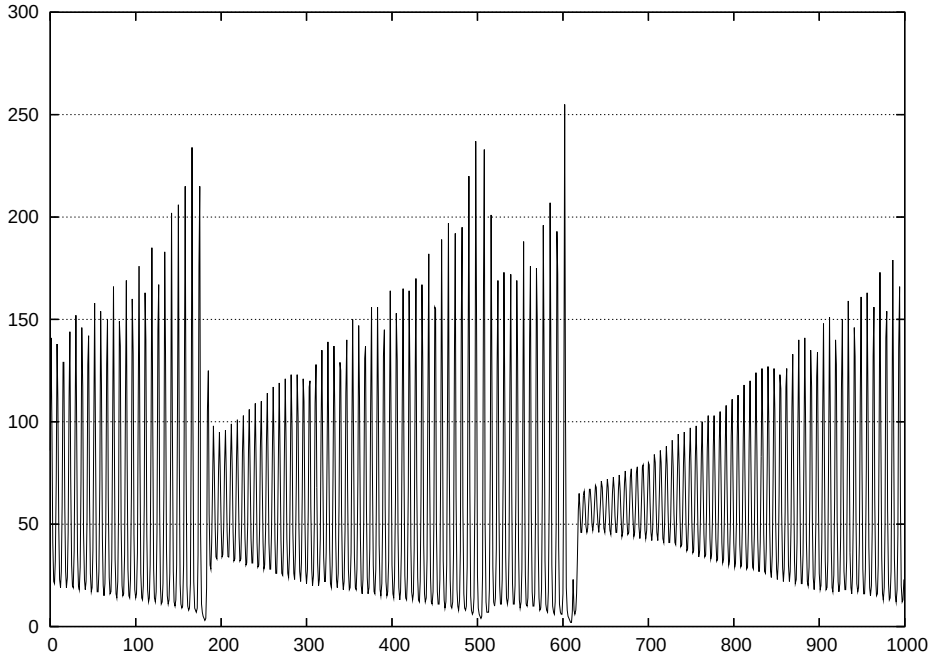


Figure 3.1. The *laser* time series used in the Santa Fe Institute Competition

method was proposed called *projected phase space* [MP03]. This method was proposed to better cope with time series mostly composed of low frequency components.

The one-class SVM method was evaluated using two time series. The first one was a synthetic time series generated from a sinusoid signal with small additive noise. A similar series was used for testing. This one had the same size of that used for training, that is, 1200, the only difference being that the testing series had a small segment with larger additive noise. Experiments were carried out using both phase spaces and projected phase spaces. In the latter case, the novelty was detected without false alarms. On the other hand, phase spaces was also able to detected the novelty but has produced two false alarms.

The second series used in the experiments was a chaotic time series with 1000 points, used in *The Santa Fe Institute Time Series Prediction and Analysis Competition* [WG94]. This series was built from the chaotic intensity pulsation of an NH_3 laser. It is depicted in figure 3.1. The experiments reported used only the phase spaces for representing the time series. The results have shown that novelties were indicated in points near 200, near 500 and near 600. Ma and Perkins argue that this agrees with the human visual novelty detection results [MP03].

3.4 CONCLUSIONS

This chapter has reviewed a number of novelty detection methods available in the literature. Methods for both classification problems and time series were reviewed. There are a number of novelty detection methods for the former class of problems, both based

on statistical pattern recognition and on neural networks. This chapter has focused on methods based on feedforward neural networks, since they are the ones used along with the time series novelty detection methods based on classification introduced in chapter 5 of this thesis.

The first methods for novelty detection in time series were based on time series forecasting. Such methods learn the normal behavior of a system through the past values of one or more time series that represent the system. Later, the system is used to predict future values of the time series. These systems issue an alarm if the difference between predicted and observed values exceed a pre-defined threshold. The problem of these methods is the difficulty to define the values of the threshold. This problem is further investigated in chapter 6 of this thesis. In that chapter, a more robust method is introduced in order to automatically compute the threshold for novelty detection.

The difficulties of forecasting-based novelty detection for time series have motivated the introduction of a number of classification-based methods in the literature. The classification-based methods reviewed in this chapter were applied to time series from machine failure detection and signal processing domains. In addition, some of the proposed methods have been evaluated on artificial time series. In chapter 5 of this thesis, two classification-based method for novelty detection in time series are proposed for fraud detection in financial systems, for example, payroll and accountancy systems. The time series available in these systems are often quite short and in many cases one is interested in detecting more subtle deviations from normality. On the other hand, the classification-based methods reviewed in this chapter were designed and tested in signal processing and artificial time series. In contrast, the forecasting and classification-based novelty detection methods for time series introduced in this thesis are evaluated on real-world time series from financial systems.

CHAPTER 4

METHODS FOR IMPROVING RBF-DDA PERFORMANCE

4.1 INTRODUCTION

The Dynamic Decay Adjustment (DDA) algorithm, reviewed in chapter 2 of this thesis, has been originally proposed for constructive training of radial basis functions neural networks (RBFs) [BD95]. Later, the algorithm was adapted for constructive training of probabilistic neural networks [BD98]. The algorithm has a number of interesting characteristics. It does not depend on weights initialization, training is very fast and it has achieved classification performance comparable to multi layer perceptrons (MLPs) in a number of tasks [BD95].

The articles that introduced DDA have argued that the classification performance of the networks generated by the algorithm on test sets has weak dependence on training parameters θ^+ and θ^- [BD95, BD98]. In particular, the paper that introduced RBF-DDA explicitly states that *"the choice of the two new parameters, θ^+ and θ^- are not as critical as it would initially appear. For all of the experiments reported, the settings $\theta^+ = 0.4$ and $\theta^- = 0.1$ were used, and no major correlations of the results to these values were noted"* [BD95]. The article that introduced DDA for probabilistic neural networks stated that *"For all experiments conducted to date, the choice of $\theta^+ = 0.4$ and $\theta^- = 0.2$ led to satisfactory results. ... in practice, especially if the feature space is only sparsely occupied, the values of the two thresholds are not critical"* [BD98]. Therefore, the authors advocate the use of the default parameters values, $\theta^+ = 0.4$ and $\theta^- = 0.1$ ($\theta^- = 0.2$ in the case of PNNs) for training using DDA [BD95, BD98].

In contrast with what has been argued previously regarding DDA parameters, we have observed experimentally that for some datasets the value of θ^- considerably influences performance. The important influence of θ^- on performance has been observed on both benchmark classification datasets from the UCI repository [ONM04c, OMNM04, OMM05b, OMM05a] and novelty detection in time series via classification (see chapter 5) [ONM04b]. DDA training with θ^- smaller than the default produces networks with larger number of RBF units in the hidden layer. This can improve coverage of the input space and thus enhance generalization performance; too small θ^- values can lead to overfitting [ONM04c, ONM04b, OMNM04].

The observation that a considerable improvement of DDA performance could be achieved has motived an investigation of methods for this purpose in this thesis. This chapter proposes four different methods for improving RBF-DDA performance.

The first method is based on the selection of an appropriate value for θ^- for boosting performance. In this method, RBF-DDA training is carried out in two phases. In the first phase, the training data is divided into *training* and *validation* sets. Training is carried out

using the training set. After training, the network generalization performance is tested using the validation set. The optimal θ^- value will be the value that minimizes error on the validation set. In the second phase, the RBF-DDA is trained with the complete training set, using the optimal θ^-.

Although the first method proposed here considerably improves RBF-DDA performance on the datasets considered it also leads to much larger classifiers [ONM04c]. This has motivated us to propose a second method intended to improve RBF-DDA performance without increasing the resulting networks. The idea is to adjust the weights of the RBF network built by DDA by a gradient-descent algorithm. Therefore, training is also carried out in two phases in this method. In the first phase, an RBF-DDA network is trained using the complete training set. In the second phase the weights of the trained network are adjusted using the reduced training set. The performance on the validation training set is evaluated during training. Training stops when the sum of squared errors on the validation set increases.

The third method proposed here for improving RBF-DDA performance combines a recent method for data reduction, proposed for expediting model selection for support vector machines [OCHO03], with the selection of θ^- [OMNM04]. This method also aims to improve RBF-DDA generalization performance without producing larger networks. The idea of the method consists in firstly applying the data reduction algorithm in order to select the most relevant patterns for training, that is, those training patterns that lie near the boundary that separate different classes. Subsequently, an RBF network is trained with DDA using only the patterns in the reduced training set. The problem with this method is that the data reduction algorithm has a parameter k which considerably influences the data reduction rate and, consequently, the classification performance. Therefore, in our method, both k and θ^- are carefully selected via cross-validation. The experiments reported in this chapter show that this method achieve classification performance similar to the two previous methods with the advantage of producing smaller networks.

One disadvantage of the former method for improving RBF-DDA performance is that the data reduction algorithm employed may consume considerable computational time. The fourth method proposed in this chapter was intended to tackle this problem. This method, which we call RBF-DDA-SP [OMM05a], was inspired on RBF-DDA-T, which was described in chapter 2 [Pae04]. In this chapter, we will show that RBF-DDA-T performance degrades for smaller values of θ^- instead of improving. To overcome this problem we propose RBF-DDA-SP, a method in which pruning takes place only at the end of DDA training [OMM05a]. Additionally, RBF-DDA-SP prunes only a portion of the neurons which cover only a training sample, in contrast to RBF-DDA-T [Pae04]. We show that this method, combined with θ^- selection, leads to small networks with good generalization capabilities.

The proposed methods have been tested on six benchmark datasets obtained from the UCI machine learning repository [BM98]. Five of these datasets correspond to image recognition tasks, including three optical character recognition (OCR) datasets. The sixth dataset is *soybean*, which is a smaller dataset also available from the UCI repository [BM98]. The results obtained by the training methods proposed in this chapter are

compared with a number of classifiers proposed in the literature, including RBF-DDA itself (with default parameters) and MLP, AdaBoost, k-NN and SVM.

The methods proposed in this chapter can also be used in conjunction with one of the classification-based methods for novelty detection in time series proposed in chapter 5 of this thesis. In fact, this was the main motivation behind the investigations leading to the methods of this chapter. They were tested on benchmark classification datasets in order to show that they achieve good results on other datasets as well. The experiments described in chapter 5 demonstrate that these methods are also valuable for improving RBF-DDA performance on time series novelty detection.

4.2 THE PROPOSED METHODS

4.2.1 Improving RBF-DDA through θ^- Selection

Berthold and Diamond argue that the DDA training parameters θ^+ and θ^- have weak influence on the network performance [BD95, BD98]. Hence, they propose that the default values of these parameters should be always used ($\theta^+ = 0.4$ and $\theta^- = 0.1$). This is true for some problems. However, we have observed experimentally that for some classification problems the value of θ^- has considerable influence on generalization performance. Therefore, we propose a method for boosting RBF-DDA performance by adequate selection of the value of θ^-.

The value of θ^- influences the number of RBF units added by DDA during training, as discussed in chapter 2. Training with smaller θ^- can produce larger number of RBF units. This can lead to a classifier that correctly classifies all training samples, but that, on the other hand, does not generalizes well, that is, overfit training data. To avoid overfitting, we propose to use part of the training data to form a *validation set*. We will use the validation set to select the θ^- that achieves the best performance on this set.

Initially, we divide the available data into training and test sets. We also refer to the former as the *complete training set*. Next, we divide the complete training set into a *reduced training set* and a *validation set*. Training is carried out in two phases. The first phase uses the reduced training set to train RBF-DDA and the validation set to evaluate generalization performance. This phase is carried out with decreasing values of θ^- in order to select θ^-_{opt}, the value of θ^- that (nearly) optimizes performance on the validation set, that is, generalization performance. In the second phase, training is carried out using the complete training set with $\theta^- = \theta^-_{opt}$. The default θ^+ value, 0.4, is used in both training phases. Finally, the generalization performance of the resulting RBF-DDA network is assessed using a disjoint test set. The training method just described is presented in algorithm 3.

Our method tests a number of θ^- values using the validation set, starting with the default value, $\theta^- = 0.1$. Next, θ^- is decreased by $\theta^- = \theta^- \times 10^{-1}$ (line 5 of algorithm 3). This is done because we have observed that performance does not change significantly for intermediate values of θ^- [ONM04c]. θ^- is decrease until the validation error starts to increase, since smaller values lead to overfitting [ONM04c]. The near optimal θ^- found by this procedure is used to train using the complete training set [ONM04c].

It is important to stress that the method introduced here maintains two important RBF-DDA characteristics: (a) the constructive nature of the algorithm and (b) the effective use of all training data to adjust the parameters of the model (RBF network).

1: $\theta^+ = 0.4$
2: $\theta^-_{opt} = \theta^- = 10^{-1}$
3: Train one RBF-DDA with θ^- using the reduced training set and test on the validation set to obtain $ValError = MinValError$
4: **REPEAT**
5: $\theta^- = \theta^- \times 10^{-1}$
6: Train one RBF-DDA with θ^- using the reduced training set and test on the validation set to obtain $ValError$
8: **IF** $ValError < MinValError$
9: $MinValError = ValError$
10: $\theta^-_{opt} = \theta^-$
11: **ENDIF**
12: **UNTIL** $ValError > MinValError$ **OR** $\theta^- = 10^{-10}$
13: Train one RBF-DDA with θ^-_{opt} using the complete training set
14: Test the RBF-DDA trained with θ^-_{opt} on the test set

Algorithm 3. Improving RBF-DDA through θ^- selection

4.2.2 Improving RBF-DDA through Weights Adjustment

The method of subsection 4.2.1 considerably improves RBF-DDA generalization performance on some datasets, as the experiments reported below will show. However, the resulting networks are much bigger than those generated by RBF-DDA trained using default parameters ($\theta^+ = 0.4$ and $\theta^- = 0.1$). For one dataset considered in the experiments the RBF created by this method had almost one RBF (prototype) for each pattern of the complete training set.

In practical applications the complexity of the classifier can be as important as its performance. This has motivated us to propose a second method for improving RBF-DDA performance. This method firstly builds an RBF network using DDA. Next, a final adjustment of the weights of the network created by DDA is performed. Thus, the RBF network generated by this method maintains the architecture of the RBF built by DDA.

The DDA algorithm, reviewed in chapter 2, was designed for building and training RBF networks for classification tasks only. A trained RBF-DDA network classifies a given pattern using the winner-takes-all approach, where the unit with the highest activation gives the class. The sum of squared errors (SSE) has no meaning for these networks. The weights of a trained RBF-DDA network have integer values (see the algorithm in chapter 2). The method proposed in this subsection produces networks with much smoother weights and, consequently, outputs. This can improve the generalization capability of the networks, as shown in the experiments below.

The training method proposed in this subsection has also two phases. The data available for training is also divided into a reduced training set and a validation set, as in the previous method (subsection 4.2.1). In the first phase, an RBF network is built and trained by the DDA algorithm using the complete training set, as usual [BD95]. In the second phase, the weights of the resulting RBF-DDA network are adjusted using a gradient-descent algorithm in supervised mode [Hay98].

The second phase is carried out using only the patterns from the reduced training set. The validation set is used to determine when to stop the adjustment of the weights in order to avoid overfitting. After each training epoch of the second training phase, the sum of squared errors (SSE) on the validation set is computed. Training stops when the SSE on the validation et starts to increase. This corresponds to the well known early stopping procedure used in MLP training to avoid overfitting [Hay98, Pre94]. In each training epoch every weight A_i of the RBF-DDA network is adjusted according to:

$$A_i(t+1) = A_i(t) + \eta \times e_j(\overrightarrow{x}, t) \times R_i(\overrightarrow{x}) \tag{4.1}$$

where $R_i(\overrightarrow{x})$ is the output of the ith RBF unit for training pattern $\overrightarrow{x}$, η is the learning rate and $e_j(\overrightarrow{x}, t)$ is the error of the output unit j to which the RBF unit i is connected for the training pattern $\overrightarrow{x}$. The individual error $e_j(\overrightarrow{x}, t)$ is used above because the output units are linear and independent from each other [BD95]. In fact, each output unit can be regarded as an independent linear unit. In each epoch, the weights of the network are adjusted for every pattern of the reduced training set.

Notice that the number of hidden units and the center and the width of each RBF are not altered during the second phase of training. This phase only adjusts the weights in order to minimize the SSE on the reduced training set. The use of a validation set in the second phase of training helps to avoid overfitting training data.

The complete training method just described is presented in algorithm 4. Notice that any values can be used for DDA parameters θ^+ and θ^-. The experiments reported in subsection 4.3.3 show that $\theta^- = 0.1$ and $\theta^- = 10^{-2}$ considerably improve performance without increasing the networks. Smaller values of θ^- do not lead to further improvements, as the experiments show. Moreover, smaller values of θ^- builds larger networks.

4.2.3 Integrating Data Reduction and Parameter Selection for Improving RBF-DDA Performance

This section proposes an alternative method which also aims to improve RBF-DDA performance without increasing resulting networks [OMNM04]. The method proposed here employs a data reduction algorithm recently put forward for expediting support vector machines model selection [OCHO03]. In the original application, the data reduction algorithm was used to reduce the number of patterns in the training set in order to speed the selection of the training parameters of SVM, namely, the cost parameter C and the kernel parameter γ, for optimal SVM performance. These parameters are selected via cross-validation using the reduced training set (obtained by the data reduction algorithm) instead of the full training set. This procedure results in faster model selection. Subsequently, the SVM is trained with the full training set using the optimal values of

1: Initialize θ^+ and θ^-
2: Train one RBF-DDA with these parameters using the complete training set
3: Compute $SSEVal = SSEValMin$
4: Initialize η and $maxEpochs$
5: $Epochs = 0$
6: **REPEAT**
7: **FORALL** training pattern from the reduced training set $(\overrightarrow{x}, c)$ **DO**
8: **FORALL** RBF unit i **DO**
9: Compute $R_i(\overrightarrow{x})$
10: Compute $e_j(\overrightarrow{x})$
11: Adjust A_i by $A_i = A_i + \eta \times e_j(\overrightarrow{x}) \times R_i(\overrightarrow{x})$
12: **ENDFOR**
13: **ENDFOR**
14: compute $SSEVal$
15: **IF** $SSEVal \leq SSEValMin$
16: $SSEValMin = SSEVal$
17: **ENDIF**
18: $Epochs = Epochs + 1$
19: **UNTIL** $SSEVal > SSEValMin$ **OR** $Epochs >= MaxEpochs$
20: Test the improved RBF-DDA on the test set

Algorithm 4. Improving RBF-DDA through weights adjustment

the parameters C and γ. The data reduction algorithm has a parameter k which has direct influence on the reduction rate [OCHO03].

In the method proposed here, we combine the data reduction algorithm mentioned above with the method based solely on the selection of the parameter θ^- (proposed section 4.2.1) in order to improve RBF-DDA performance. Firstly, the training set is reduced by using the data reduction algorithm with a suitable value for the parameter k. Next, an RBF network is built by using the DDA algorithm. Notice that, in contrast to its previous use, the data reduction algorithm is used here to produce a reduced training set that is used to train an RBF network, instead of just selecting its optimal parameters. The motivation for using the data reduction algorithm is to eliminate redundant training patterns in order to help DDA produce much smaller networks. Moreover, the experimental results presented in subsection 4.3.4 show that such networks generalize considerably better. The integration of DDA training with the data reduction algorithm requires selection of three parameters: θ^+, and θ^-, and k. The first one has weak influence on performance. The second and third ones are selected via cross-validation using training data. Alternatively, for large datasets, θ^- can be selected as in section 4.2.1 [ONM04c].

The data reduction algorithm employed in our method is presented below (see algorithm 5). The idea of the algorithm is to reduce the training set by removing those training patterns that are far away from the boundary that separate different classes of training data. The algorithm works by first sorting the training dataset in descending

order according to the distance of each pattern's *nearest enemy* [OCHO03]. The nearest enemy of a pattern is defined as the nearest neighbor of the pattern that belongs to a different class. Next, the patterns are examined for removal beginning at the pattern furthest from its nearest enemy. Finally, the data reduction algorithm examines patterns in the ordered training set one by one, removing a pattern if the pattern and all of its k nearest neighbors in the remaining training set belong to the same class. In general, larger values of k result in lower reduction rates (fewer training patterns are removed from the training set). In the context of SVM model selection, $k = 3$ was used [OCHO03].

Input: Training dataset T
Output: Reduced dataset R
// $NED(x_i)$: the distance of x_i's nearest enemy
// $N_k(x_i)$: the set of x_i's nearest neighbors
// $C(x_i)$: the class label of x_i

Data Reduction(T)
1: $R \leftarrow T$
2: **FOREACH** instance x_i in R
3: Find $NED(x_i)$
4: **SORT** R in descending order by $NED(x)$
5: **FOREACH** instance x_i in R
6: Find $N_k(x_i)$ in R
7: **IF** $C(x_i) == C(p)$, $\forall p \in N_k(x_i)$
8: Remove x_i from R
9: **RETURN** R

Algorithm 5. The data reduction algorithm

The experiments presented in next section show that the parameter k of the data reduction algorithm has great influence on the performance of the training method proposed here. Thus we must carefully select k in order to achieve good generalization. Notice that in the context of SVM model selection k was fixed: $k = 3$. In our experiments we have observed that this k produces a large reduction rate. RBF-DDA trained with such small training sets has not a good generalization performance. In general, smaller value of k results in large reduction rates. This can result in the removal of important information for DDA training. On the other hand, larger values of k produce larger training sets, that is, smaller reduction rates. We have observed empirically that the optimal value for k in our method is near the k which reduces the training set by half. Thus, the search starts by using this value of k (which is different for each dataset) and then other values in its vicinity are tried. For a given k, θ^- is searched as explained in subsection 4.2.1.

4.2.4 RBF-DDA-SP: DDA with Selective Pruning and Parameter Selection

The experiments of subsection 4.3.4 have shown that the integrated method proposed in the last subsection indeed produces networks with much improved generalization performance and compact size in comparison with the original RBF-DDA. Nonetheless, that method has one drawback: it can be slow to train because it employs a data reduction algorithm whose reduction rate has to be carefully selected. This subsection introduces another method aimed to tackle this difficulty.

A recent extension to DDA has appeared in the literature with a different aim, namely, reducing the number of neurons generated by DDA [Pae04]. The method, referred as RBF-DDA with temporary neurons (RBF-DDA-T), introduces on-line pruning of neurons on each DDA training epoch (see subsection 2.2.2.2 and [Pae04]). We have attempted to integrate RBF-DDA-T with θ^- selection, however, we have observed that the method severely prunes the networks for smaller values of θ^-, thereby generating much smaller networks with heavily degraded performance. Conversely, RBF-DDA generates larger networks for smaller θ^-, which, for some datasets, considerably improves performance [ONM04c, ONM04b].

This subsection proposes an extension to RBF-DDA which combines *selective pruning* and *parameter selection* (θ^- selection). We call this extension *RBF-DDA-SP* [OMM05a]. In contrast to RBF-DDA-T, the method proposed here prunes only a portion of the neurons which cover only one training sample. Moreover, in our method pruning is carried out only after the last epoch of DDA training. The neurons to be pruned are randomly selected from those which cover only one training sample. Recall that in a trained RBF-DDA, A_i gives the number of training samples covered by RBF neuron i with $R_i(\overrightarrow{x}) \geq \theta^+$ [BD95].

RBF-DDA-SP, combines selective pruning with θ^- selection [OMM05a]. In our method, pruning is carried out only after the network is built by DDA, which takes place generally after four to five epochs of training [BD95, BD98]. In addition, we perform *selective pruning*, which means that we remove only a percentage p of those hidden neurons which cover only one training pattern. Conversely, RBF-DDA-T implements on-line pruning, that is, pruning after each DDA training epoch and removes all neurons which cover only one training pattern [Pae04], that is, whose weight $A_i = 1.0$. These are the neurons whose outputs are greater than θ^+ for only one training pattern.

We argue that only a portion of those neurons should be removed in order to guarantee a good coverage of the input space. If a neuron of this type is removed from the network, a neighbor neuron of the same class remaining in the network will cover the respective training pattern with an output smaller than θ^+. Even so, most of these training patterns - corresponding to pruned neurons - will be correctly classified, because RBF-DDA uses the winner-takes-all criterion for classification [BD95]. This will not happen if most of the neurons are pruned from the network. Thus, the percentage of pruning should be carefully selected in order to guarantee good coverage of the input space. The neurons to be pruned are randomly selected from those which cover only one training sample. Thus, our method has two critical parameters, namely, p and θ^-. These parameters can be selected via cross-validation for improved performance.

Dataset	No of Inputs	No of Classes	No of Patterns Red. Tr.	Validation	Complete Tr.	Test
optdigits	64	10	2,868	955	3,823	1,797
pendigits	16	10	5,621	1,873	7,494	3,498
letter	16	26	11,250	3,750	15,000	5,000
satimage	36	6	3,327	1,108	4,435	2,000
segment	19	7	158	52	210	2,100
soybean	82	19	385	128	513	170

Table 4.1. Characteristics of the datasets

The idea of RBF-DDA-SP is to reduce the complexity of the network by removing neurons considered redundant. In addition, θ^- is selected as explained in subsection 4.2.1 in order to improve generalization performance [ONM04c]. The percentage of pruning to be carried out by RBF-DDA-SP can also be selected via cross-validation together with θ^-. The neurons to be removed are randomly selected from those which cover only one training sample (that is, those that have weight $A_i = 1.0$).

4.3 EXPERIMENTS

4.3.1 Datasets

The training methods proposed in this chapter were tested on five image recognition datasets available from the UCI machine learning repository [BM98]. Three of these datasets are optical character recognition (OCR) datasets. The *soybean* dataset, also available from UCI was also used in the experiments. This dataset was selected because it is much smaller than the image recognition datasets and it was our intention to evaluate the performance of the methods on such datasets as well.

Two of the optical character recognition datasets (*optdigits* and *pendigits*) are databases of handwritten digits (one is optical and the other is pen-based) [KA00]. The test sets in these datasets are formed by data obtained from independent writers, that is, writers whose data are not present on the respective training sets. The third dataset is a database of letters that contains 20,000 patterns from 26 classes, corresponding to the 26 characters (letters) [FS91, KA00, KGC00]. The fourth and fifth datasets are image recognition datasets. Table 4.1 summarizes the characteristics of these datasets. The reduced training sets, used in the methods of subsections 4.2.1 and 4.2.2, have 75% of the training patterns. These are the first patterns of the original training sets made available by UCI. The validation sets are formed by the remaining 25% of the training patterns. The complete training sets contain all training patterns.

The first dataset, henceforth referred as *optdigits*, is a database of digits handwritten by Turkish writers [KA00]. This database contains digits written by 44 writers. The training and validation sets were generated from the first 30 writers. The test set was generated by the remaining independent 14 writers. The database was generated by scanning and processing forms to obtain 32×32 matrices of handwritten digits which were then reduced to 8×8 [KA00].

The second dataset, henceforth referred as *pendigits*, is a database of pen-based handwritten digits [KA00]. It contains samples written down on a touch-sensitive tablet, which were then resampled and normalized to a temporal sequence of eight pairs of (x, y) coordinates. The resulting patterns were divided into training, validation and test set according to table 4.1. It is important to stress that the test set is formed entirely from written digits produced by independent writers.

The third dataset, henceforth referred as *letters* contains 20,000 labeled samples of, on an average, 770 samples per letter [FS91, KGC00, KA00]. The character images were based on 20 different fonts and each letter within these fonts was randomly distorted to produce a dataset of 20,000 unique stimuli.

The fourth dataset is referred as *satimage*. This dataset was generated from Landsat Multi-Spectral Scanner image data. Each pattern from this dataset contains 36 pixel values and a number indicating one of the six class categories of the central pixel [MST94, WNC03].

The fifth dataset is referred as *segment* [BM98]. It is composed of image segmentation data. The instances were drawn randomly from a database of seven outdoor images. The images were handsegmented to create a classification for every pixel. The classes are *brickface*, *sky*, *foliage*, *cement*, *window*, *path*, and *grass*. There are 30 instances per class for training data and 300 per class for test data. In the experiments, we have used the division into training and test sets as in the UCI repository [BM98] (see table 4.1). Some of the experiments also considered this dataset in *inverted order*, that is, using the test data for training and the training data for testing. This was done in order to consider also a larger pattern set for training. We refer to the former version of the dataset (i.e., the original) as *segment1*. The latter is referred as *segment2*.

Finally, the sixth dataset is the *soybean large* dataset from the UCI repository [BM98, MC80]. This is a classification problem aiming at recognizing 19 different diseases of soybeans. The discrimination is carried out based on a description of the bean and the plant plus information about the history of the plant's life [Pre94] . This dataset is also part of the *Proben1* collection of benchmarking problems for neural networks [Pre94]. We have used the versions of the datasets from this last source because the patterns are encoded in a suitable form for neural networks processing. All datasets from *Proben1* are available in three partitioning versions. We have used the three soybean versions from Proben1, that is, *soybean1*, *soybean2* and *soybean3*. Notice that the soybean dataset has much less patterns than the previous five datasets considered in this chapter. We considered the different partitionings from *Proben1* because performance can vary significantly for different partitionings for small datasets.

4.3.2 Results and Discussion for RBF-DDA Improvement by θ^- Selection

This section reports results obtained using the first method proposed in this chapter for improving RBF-DDA generalization performance, namely, the method based on θ^- selection, described in section 4.2.1. The experiments reported here were carried out using the *holdout method*, whereby the available data is divided into training and test sets [TK03]. The division of data into training and test sets is shown in table 4.1.

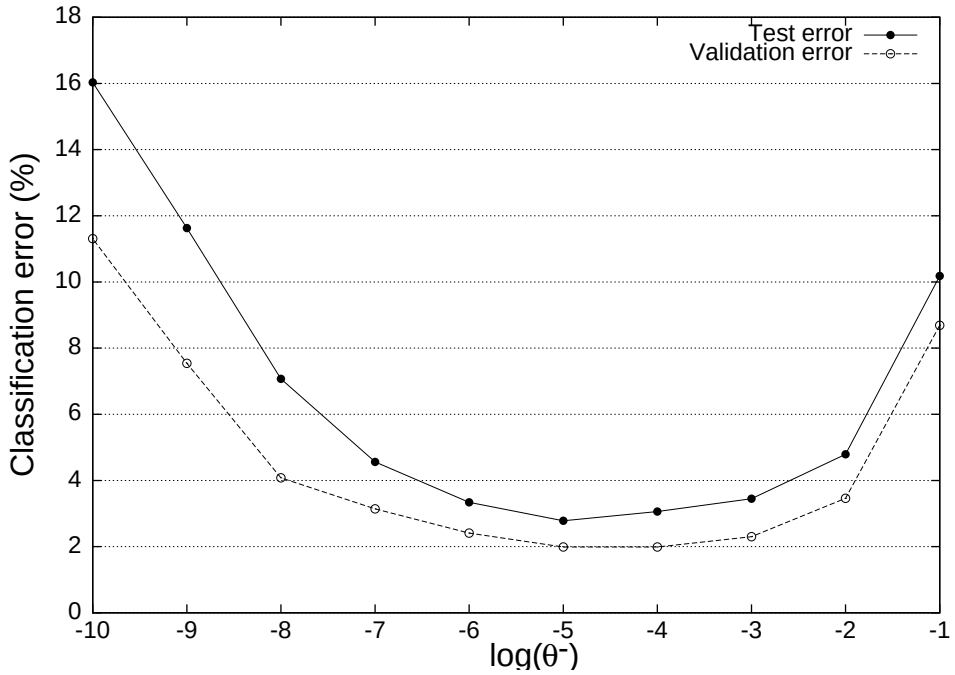


Figure 4.1. Classification error on test and validation sets for *optdigits* dataset

Figures 4.1, 4.2, 4.3, 4.4 and 4.5 show the classification errors on test and validation sets as function of $\log(\theta^-)$ for *optdigits*, *pendigits*, *letter*, *segment1* and *segment2* datasets, respectively. It can be seen that for each dataset the θ^- that minimizes classification error on validation set is the same that minimizes this error on the respective test set. A quite similar behavior happens for the *satimage* dataset. Thus, it is clear that performance on validation sets can be used to select θ^- that optimizes performance on test sets. Moreover, it can also be seen that when the validation error reaches a minimum, overfitting starts with decreasing θ^- values. Therefore, training should stop after the first increase in validation error, as proposed in algorithm 3.

Figures 4.6, 4.7 and 4.8 show the classification errors on test and validation sets as function of $\log(\theta^-)$ for each partitioning of the *soybean* dataset. For the *soybean1* and *soybean2* partitioning the same observation made for the optical character recognition datasets is valid, that is, algorithm 2 selects the value of θ^- that minimizes error on the test sets. For *soybean 3*, on the other hand, this does not happen. In figure 4.8 the smallest classification error for the validation set is achieved with $\theta^- = 10^{-3}$, whereas the test set error is minimized with $\theta^- = 10^{-4}$. Even in this case, the method proposed here achieves classification error on test set very near the optimal. The optimal classification error (for $\theta^- = 10^{-3}$) is 12.94%. Our method achieves 13.53% classification error, with $\theta^- = 10^{-4}$, while the error with the default $\theta^- = 10^{-1}$ is 34.12%. We believe that this has happened because the training set has a small number of patterns. An alternative option to overcome this limitation would be to use *n-fold* cross-validation instead of the simple validation adopted in our method. This could improve performance for smaller datasets at the expense of larger training periods [DHS00, TK03].

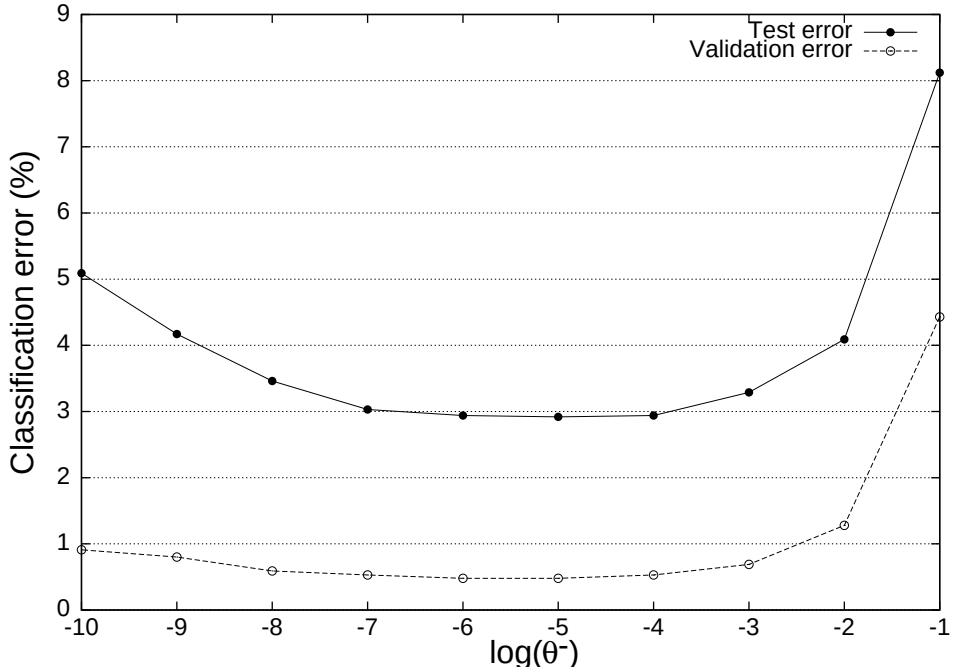


Figure 4.2. Classification error on test and validation sets for *pendigits* dataset

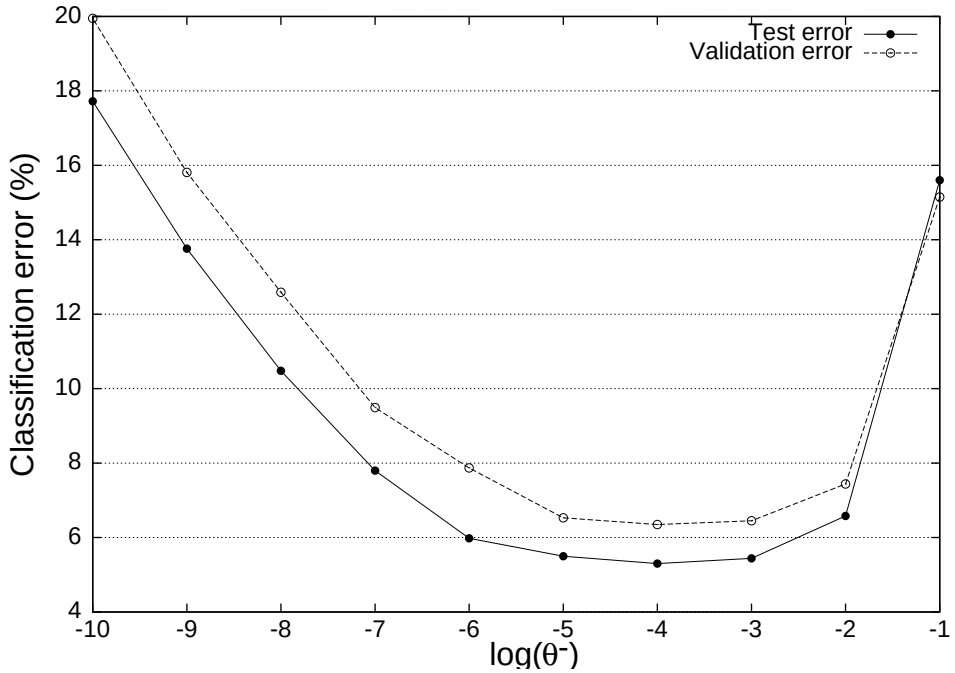


Figure 4.3. Classification error on test and validation sets for *letter* dataset

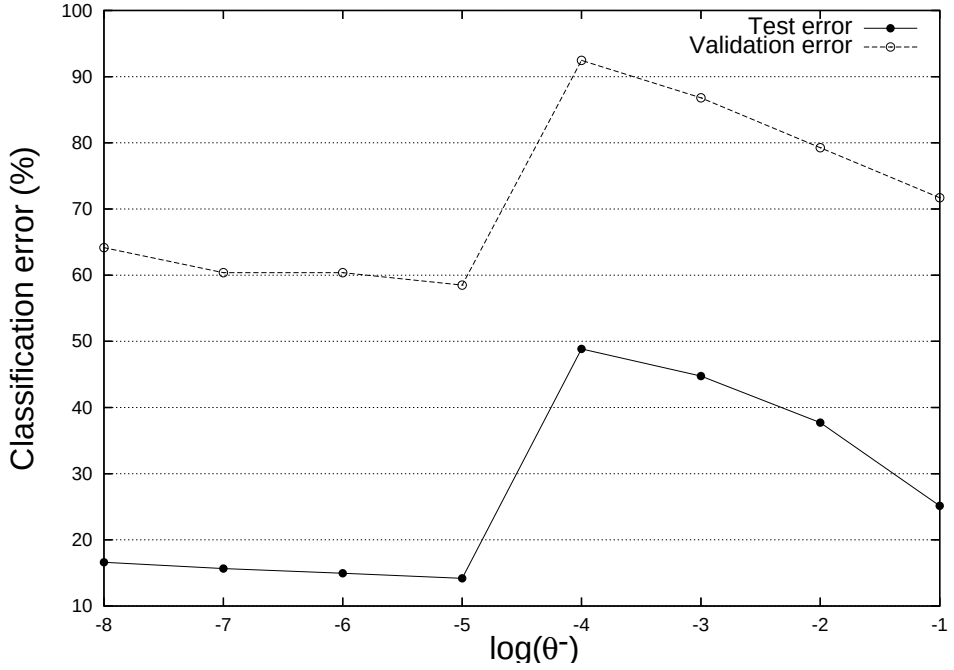


Figure 4.4. Classification errors on test and validation sets for *segment* dataset

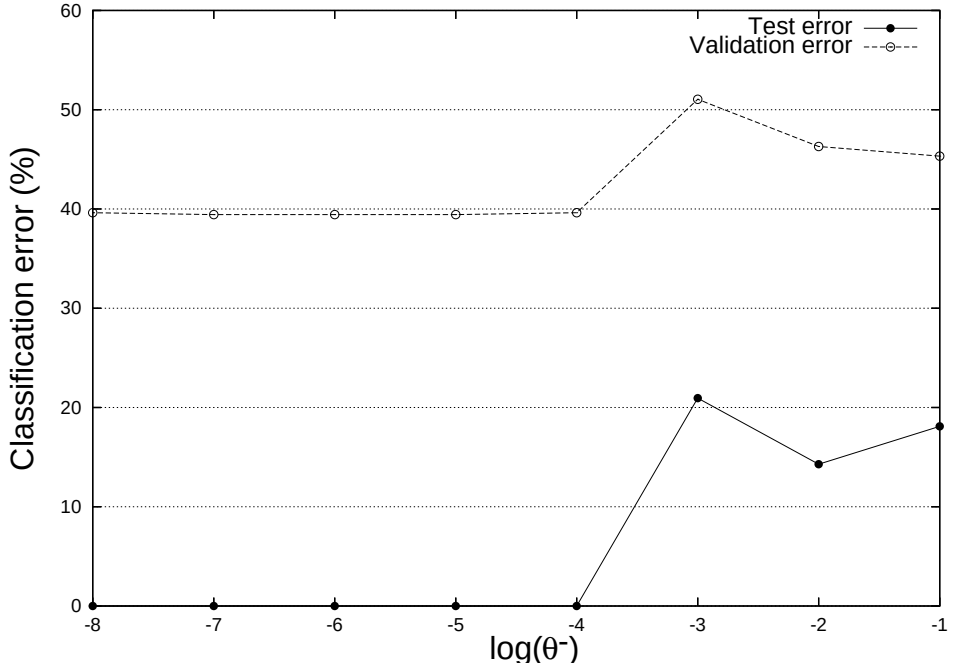


Figure 4.5. Classification errors on test and validation sets for inverted *segment* dataset

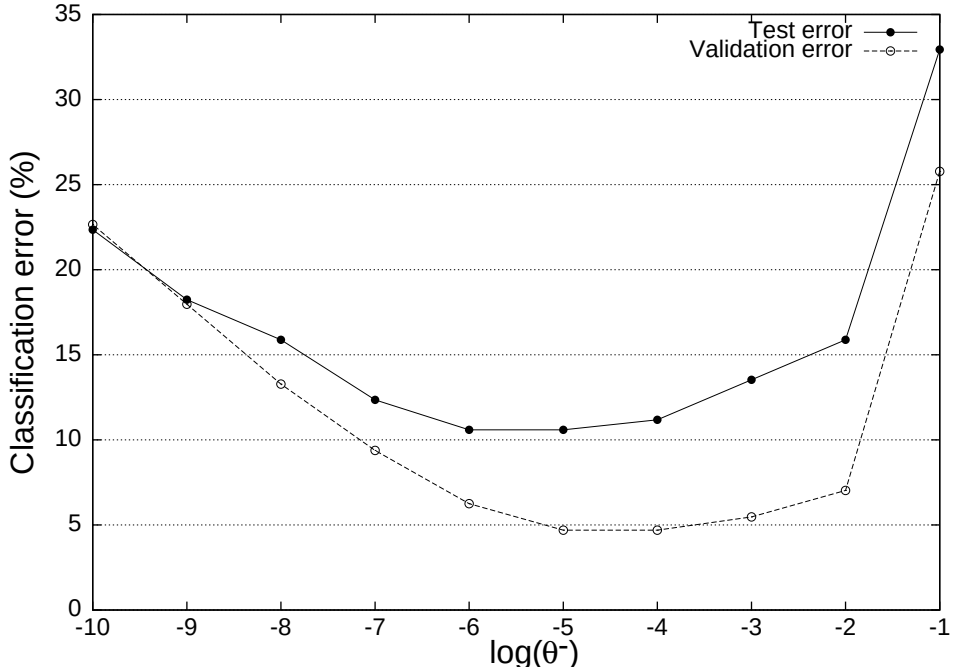


Figure 4.6. Classification error on test and validation sets for *soybean1* dataset

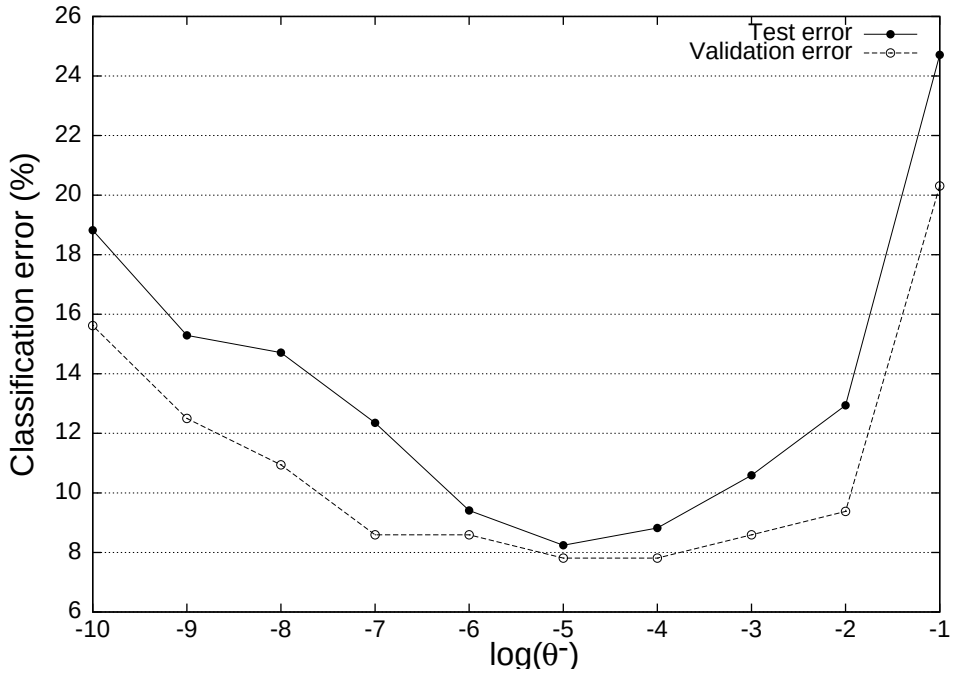


Figure 4.7. Classification error on test and validation sets for *soybean2* dataset

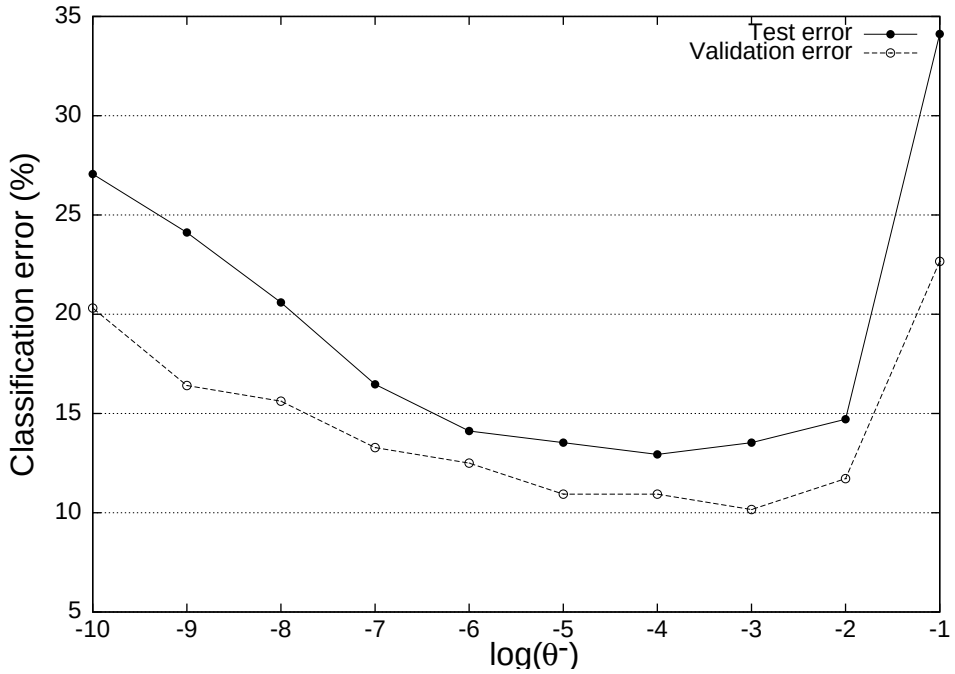


Figure 4.8. Classification error on test and validation sets for *soybean3* dataset

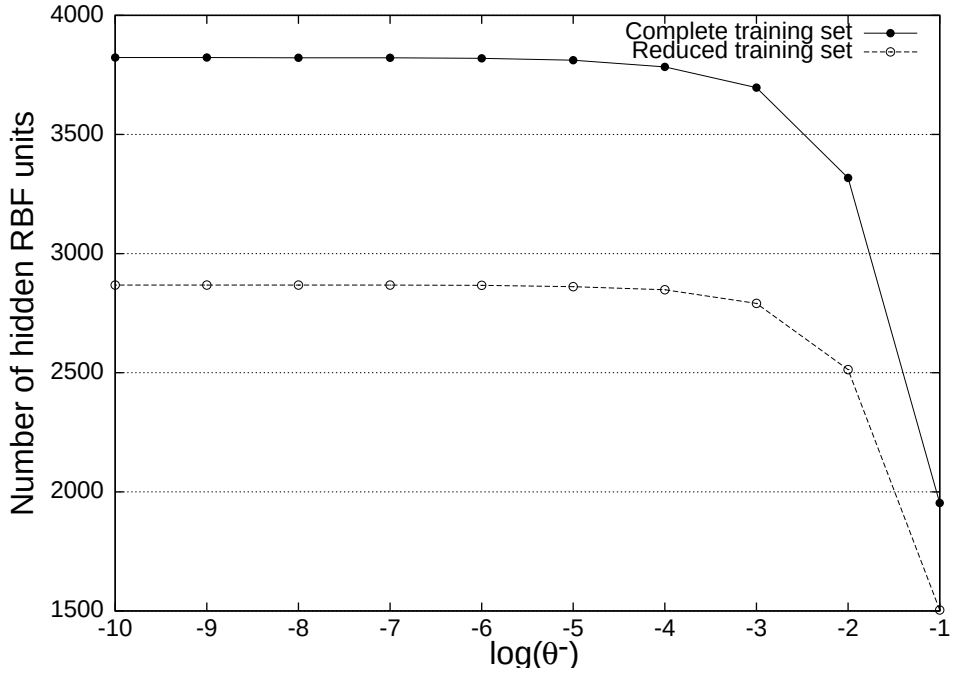


Figure 4.9. Number of hidden RBFs for complete and reduced training sets for *optdigits* dataset

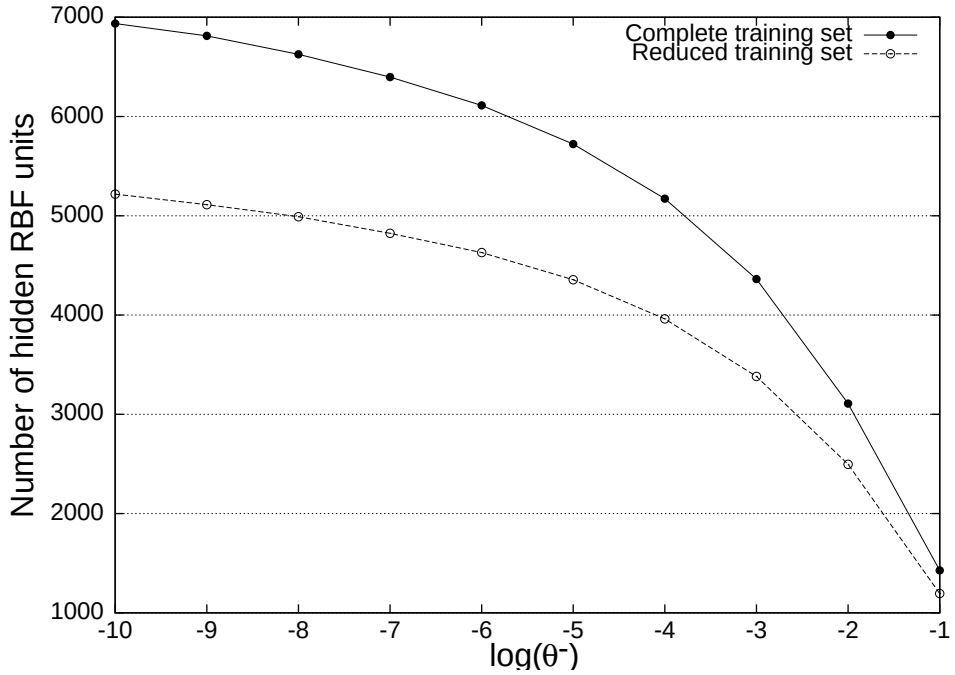


Figure 4.10. Number of hidden RBFs for complete and reduced training sets for *pendigits* dataset

Figures 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, 4.15 and 4.16 depict the number of RBF hidden units as a function of $\log(\theta^-)$ for the *optdigits*, *pendigits*, *letter*, *segment1*, *segment2*, *soybean1*, *soybean2* and *soybean3*, respectively. Each figure shows the number of RBF units of the network generated in phase 1 of the method of subsection 4.2.1, which utilizes the reduced training set, together with the number of RBF units generated in phase 2, where training is carried out with the complete training set. For each dataset, the graphics show that the number of hidden RBFs increases with decreasing θ^- values. This behavior has also been observed for the *satimage* dataset.

When θ^- approaches zero, the number of RBF units added by the DDA algorithm approaches the number of patterns of the training set as shown in the referred graphics. We have observed previously that this can lead to overfitting. The method proposed in subsection 4.2.1 selects θ^- so that the number of RBF units included in the network is the one that achieves near optimal coverage of the input space without overfitting.

Table 4.2 shows classification results on test sets for the image recognition datasets (*optdigits*, *pendigits*, *letter*, and *satimage*). Table 4.3 shows classification results on test sets for both versions of the *segment* dataset (normal and inverted). Both tables compare the proposed RBF-DDA with selected θ^- with RBF-DDA trained with default parameters ($\theta^+ = 0.4$ and $\theta^- = 0.1$). The results of these tables show that the proposed method considerably outperforms the default RBF-DDA for all image recognition datasets considered.

Table 4.2 also compares the generalization performance of the proposed method (RBF-DDA with selected θ^-) with other classifiers. The classification algorithms compared are:

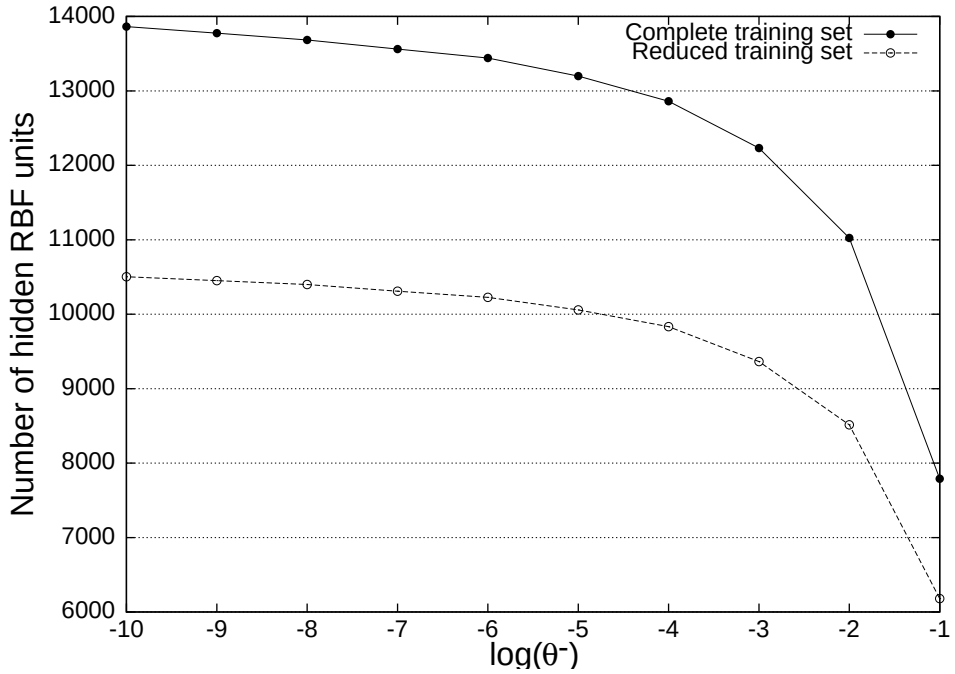


Figure 4.11. Number of hidden RBFs for complete and reduced training sets for *letter* dataset

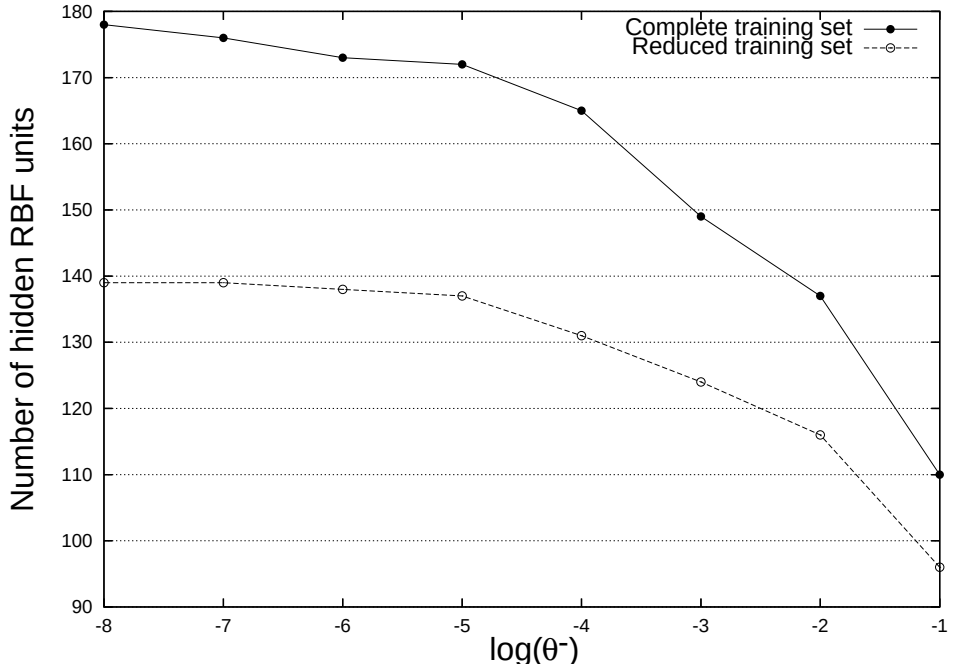


Figure 4.12. Number of hidden RBFs for complete and reduced training sets for *segment* dataset

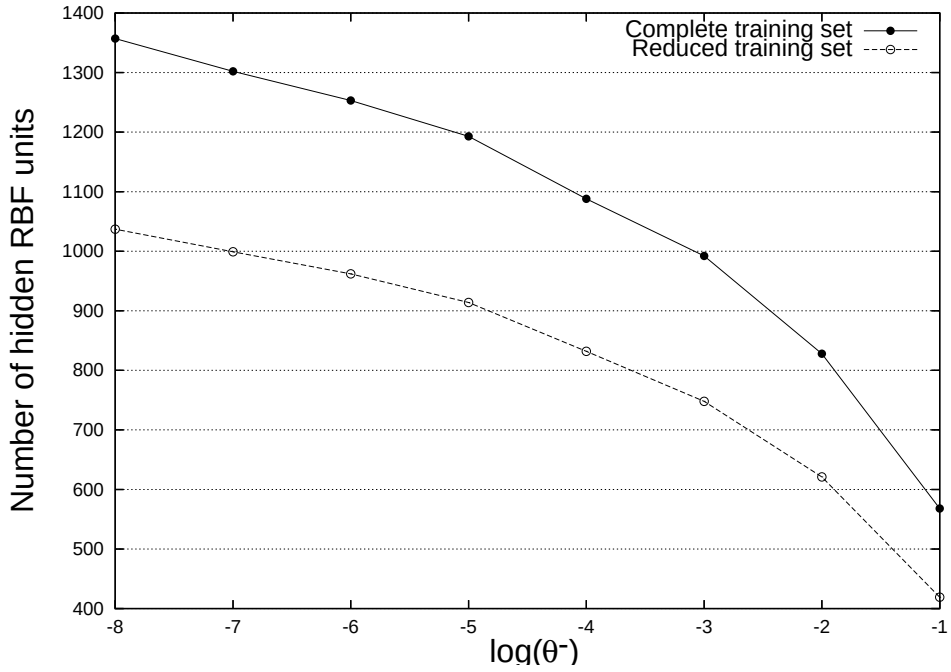


Figure 4.13. Number of hidden RBFs for complete and reduced training sets for the inverted *segment* dataset

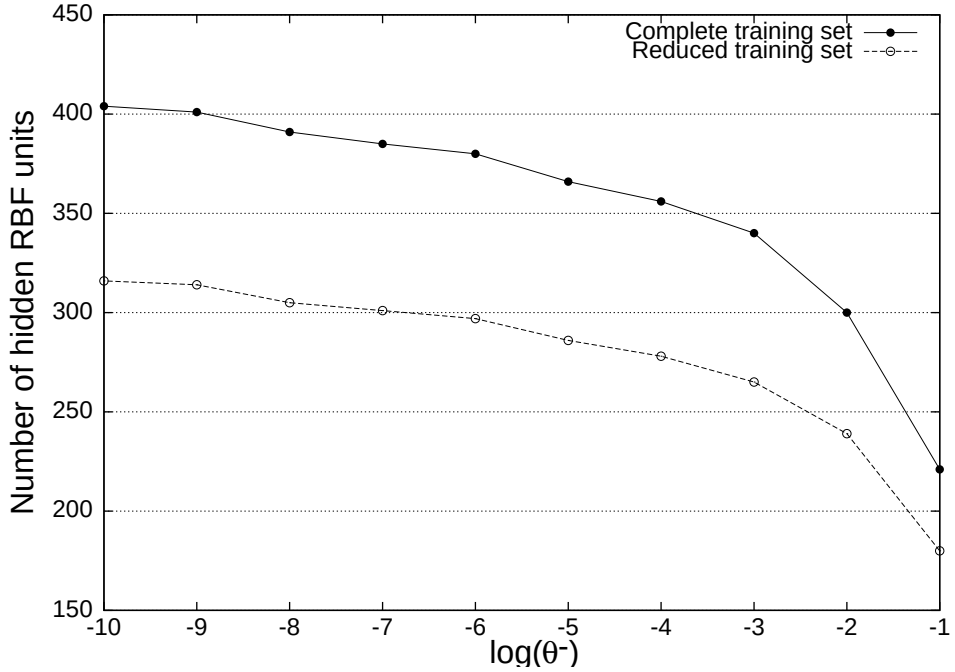


Figure 4.14. Number of hidden RBFs for complete and reduced training sets for *soybean1* dataset

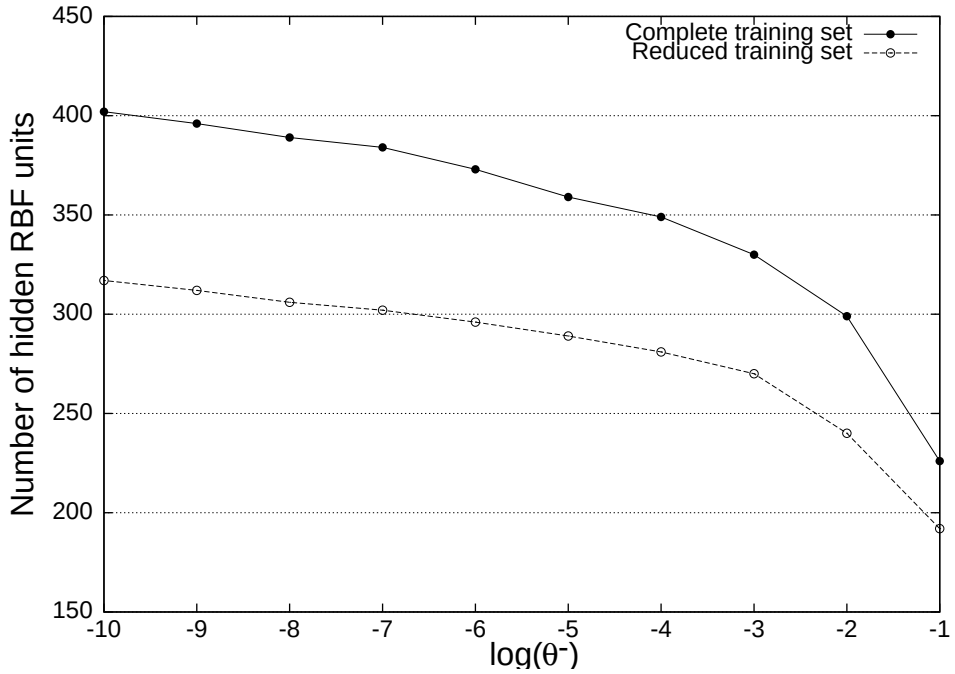


Figure 4.15. Number of hidden RBFs for complete and reduced training sets for *soybean2* dataset

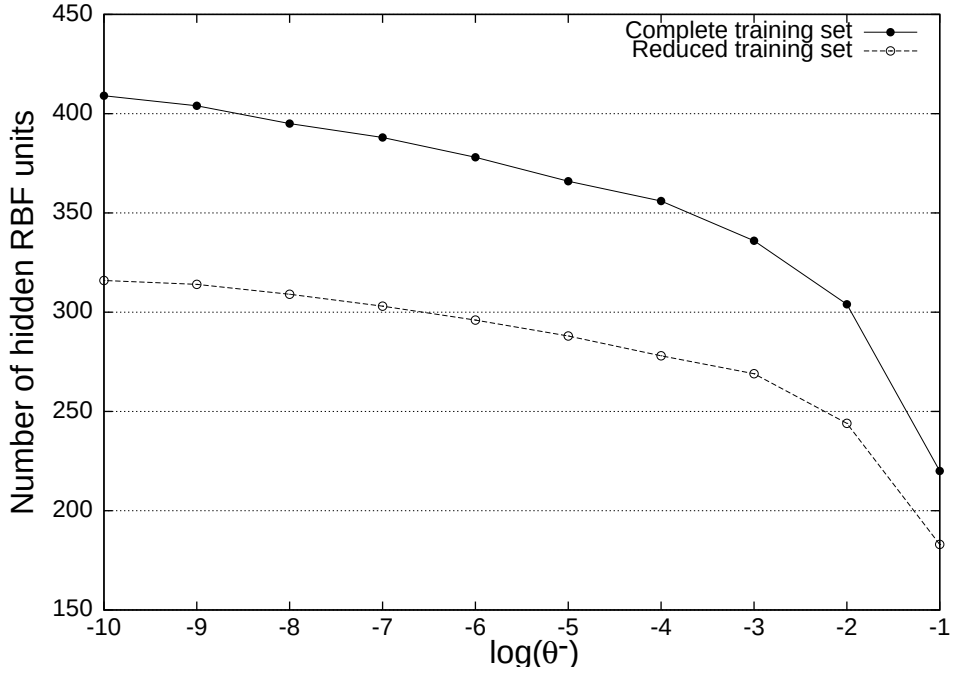


Figure 4.16. Number of hidden RBFs for complete and reduced training sets for *soybean3* dataset

Dataset	RBF-DDA (selected θ^-)	RBF-DDA Default	MLP	k-NN	RBF
optdigits	**2.78%**	10.18%	10.95% (1.90%)	3.51%	-
pendigits	2.92%	8.12%	4.83% (0.50%)	**2.29%**	-
letter	**5.30%**	15.60%	23.59% (0.90%)	6.62%	23.30%
satimage	**8.55%**	14.95%	13.90%	9.40%	12.10%
mean across datasets	**4.89%**	12.21%	13.32%	5.45%	-

Table 4.2. Classification errors on test sets for image recognition datasets

Dataset	RBF-DDA (selected θ^-)	RBF-DDA Default
segment1	**14.19%**	25.14%
segment2	**0.00%**	18.10%

Table 4.3. Classification errors on test sets for the *segment* datasets

RBF-DDA with θ^- selected according to the method proposed here; RBF-DDA with default parameters ($\theta^+ = 0.4$ and $\theta^- = 0.1$); MLP (mean classification error); k-NN; and RBF trained as described in the StatLog book [MST94]. In that book, RBFs were trained with centers selected randomly from training data, except that centers were allocated to each class in proportion to the number of representatives of that class in the dataset, with at least one center provided for each class. Each Gaussian radius was set to the distance to the nearest neighboring center. The linear system was solved by singular value decomposition.

The MLP and k-NN results for *optdigits*, *pendigits*, and *letter* were obtained by Kaynak and Alpaydin [KA00]. MLPs used 10 hidden units and k-NN used $k = 3$ [KA00]. MLP, k-NN and RBF results for *satimage* were obtained from the StatLog book [MST94].

RBF-DDA and k-NN do not depend on parameters initialization for training. On the other hand, the weights of MLPs obtained after training depend on their initial random values. Therefore, it is necessary to run MLP experiments a number of times, usually 10 to 30, in order to obtain the mean and standard deviation of classification errors presented in table 4.2. Notice that standard deviation is not reported for *satimage* because it was not reported in the StatLog book [MST94].

A number of classifiers have been compared in the work of Kaynak and Alpaydin, including machine committees [KA00]. k-NN was the best classifier on the *pendigits* and *optdigits* datasets [KA00]. In this same paper, a cascaded classifier formed by MLP and k-NN performed slightly better than single k-NN on the *letter* dataset. The error obtained by the cascaded classifier was 6.27% which is still worse than the result obtained by our RBF-DDA with θ^- selection classifier (5.30%). The improved RBF-DDA proposed here outperforms MLP in the four datasets considered in table 4.2. Our method also outperforms k-NN on three of these datasets. On *pendigits* the method proposed here obtains a classification error similar to k-NN.

Dataset	θ^+ / θ^-	Tr. epochs	No of RBFs	% of Tr. samples	Tr. set error
optdigits	0.4 / 0.1	4	1,953	51.09%	6.12%
optdigits	0.4 / 10^{-5}	3	3,812	99.71%	0.00%
pendigits	0.4 / 0.1	5	1,427	19.04%	3.10%
pendigits	0.4 / 10^{-5}	4	5,723	76.37%	0.00%
letter	0.4 / 0.1	5	7,789	51.93%	6.51%
letter	0.4 / 10^{-4}	4	12,861	85.74%	0.00%
satimage	0.4 / 0.1	4	2,812	63.40%	8.41%
satimage	0.4 / 10^{-4}	4	4,099	92.41%	0.00%
segment1	0.4 / 0.1	4	110	52.38%	7.14%
segment1	0.4 / 10^{-5}	4	172	81.90%	0.00%
segment2	0.4 / 0.1	5	568	27.05%	17.95%
segment2	0.4 / 10^{-4}	4	1088	51.81%	0.00%

Table 4.4. The RBF-DDA produced by default and selected parameters after training for the image recognition datasets

The *letter* dataset was used previously in other works as well. The original work using this dataset has obtained around 20% classification error on the test set [FS91]. A more recent work has obtained 10.10% classification error on the test set using k-NN and 20.70% using MLP [KGC00]. The best result obtained by Bayesian pairwise classifiers proposed in that work was 12.40% of classification error on the test set [KGC00]. On the other hand, the first method for improving RBF-DDA proposed in this chapter has obtained 5.30% classification error (as shown in table 4.2) and therefore, outperforms also all classifiers considered in that previous work [KGC00]. Our method also outperforms the *Locally Confident Network* (LCN), which achieved 7.40% classification error on the test set [WNC03]. Our method also markedly outperformed the RBF training method of the StatLog book [MST94], which obtained 23.30% classification in this dataset.

The MLP and k-NN results for the *satimage* dataset were obtained also from previous works [MST94, WNC03]. Our method outperforms both these classifiers in this dataset as shown in table 4.2. It also outperforms the LCN classifier, which achieved 9.40% classification error on test set [WNC03]. Our method also outperformed the RBF training method of the StatLog book [MST94], which obtained 12.10% classification in this dataset, as shown in table 4.2.

Table 4.4 shows training results for both RBF-DDA with default parameters and RBF-DDA trained with selected θ^- as in the method proposed in subsection 4.2.1. It can be seen that the number of RBF units in the proposed method is much larger than for RBF-DDA trained with default parameters. Because of that, the proposed method builds RBF-DDA networks that can better cover the training space and obtain 0% classification error on training sets. The use of a validation set in our training method allows DDA to introduce the correct number of RBF units for better coverage of training space and for simultaneously avoiding overfitting.

The computational and space complexities of classifiers can be as important as their

Dataset	RBF-DDA (θ^- selection)	RBF-DDA Default	MLP (no shortcuts)	MLP (shortcuts)	Linear networks
soybean1	10.59%	32.94%	29.40% (2.50%)	9.06% (0.80%)	9.47% (0.51%)
soybean2	8.24%	24.71%	5.14% (1.05%)	5.84% (0.87%)	4.24% (0.25%)
soybean3	13.53%	34.12%	11.54% (2.32%)	7.27% (1.16%)	7.00% (0.19%)
mean across datasets	10.79%	30.59%	15.36%	7.39%	6.90%

Table 4.5. Classification errors on test sets for soybean datasets

accuracy for the selection of a classifier for a given application. For the image recognition tasks considered here our improved RBF-DDA has produced much smaller errors than MLPs. However, MLPs require less memory space than our improved RBF-DDA. On the other hand, our method can be much faster to train.

The improved RBF-DDA proposed here has outperformed k-NN on three of the datasets and had similar performance on the remaining one. k-NN does not need training, but all training patterns must be stored to be used latter for classification of novel patterns. On the other hand, the improved RBF-DDA does not need RBF units for each training pattern. This can be seen in the fifth column of table 4.4 which shows the percentage of training patterns that are stored as prototypes in both RBF-DDA with default parameters and the improved RBF-DDA. Moreover, a trained RBF-DDA is able to classify a given pattern faster than k-NN, which can be an important advantage on many practical applications.

Table 4.5 compares classification errors on test sets for different classifiers on different partitionings of the soybean dataset [Pre94]. The results show that the improved RBF-DDA classifier proposed here considerably outperforms the default RBF-DDA for the *soybean* dataset as well. The improved RBF-DDA classifiers achieve 10.79% mean classification error across the datasets whereas RBF-DDA with default parameters obtains 30.59%. Results are also compared to MLPs and linear networks results reported on Proben1 [Pre94].

It can be seen that the improved RBF-DDA classifier outperforms MLPs without shortcut connections. On the other hand, MLPs with shortcut connections and linear networks obtain better results on these datasets. MLPs without shortcuts connections have only connections between adjacent layers. In addition to these connections, MLPs with shortcuts connections have direct connections between the input and output layers. In the case of networks with two hidden layers there are also connections from inputs to the second hidden layer and from the first hidden layer to the outputs.

The results obtained by MLPs on soybean datasets show that the partitioning of input space provided by MLPs is more appropriate than that of RBFs for this problem. However, it is important to emphasize that MLPs results were obtained after an optimization of the number of hidden layers and hidden units in each layer [Pre94]. This task is carried out manually, that is, by training and comparing the performance of a number of MLP architectures. This can be a very time consuming task. On the other hand, this task is not need in RBF-DDA networks, because the number of RBF units are determined automatically by the DDA training algorithm, since it is a constructive

Dataset	θ^+ / θ^-	Tr. epochs	No of RBFs	% of Tr. samples	Tr. set error
soybean1	0.4 / 0.1	4	221	43.10%	11.50%
soybean1	0.4 / 10^{-5}	3	366	71.34%	0.00%
soybean2	0.4 / 0.1	4	226	44.05%	8.77%
soybean2	0.4 / 10^{-5}	3	359	69.98%	0.19%
soybean3	0.4 / 0.1	4	220	42.88%	9.16%
soybean3	0.4 / 10^{-3}	4	336	65.50%	0.19%

Table 4.6. The RBF-DDA produced by default and select parameter after training for the soybean datasets

training algorithm [BD95, BD98].

Table 4.6 is analogous to table 4.4 and shows training results for both RBF-DDA with default parameters and RBF-DDA improved by using the method proposed here, this time in the soybean datasets. The results present in table 4.6 also show that the number of RBF units in improved RBF-DDA is much larger than in the case of RBF-DDA with default parameters. Once again, the use of the validation set in the proposed method allows DDA to introduce the correct number of RBF units for improving generalization performance. The fifth column of table 4.6 shows that the number of RBF units added by DDA is around 70% of the number of training patterns for the soybean datasets. Alternatively, RBF-DDA with default parameters needs to use only about 45% of the training patterns as prototypes, but its generalization ability is much worse when compared to the improved RBF-DDA, as shown in table 4.5.

4.3.3 Results and Discussion for RBF-DDA Improvement by Weights Adjustment

This section reports experiments using the method based on weights adjustment proposed in subsection 4.2.2. The simulations were carried out using the datasets *optdigits*, *pendigits*, *letter* and *soybean*, described in subsection 4.3.1. We have used the RBF-DDA implementation available on the well known SNNS neural networks simulator [Zel98] in these simulations. This simulator saves the trained networks in ASCII files with .net extension. We have implemented the weights adjustment algorithm in the C language. This program takes the .net file generated by SNNS after DDA training; adjusts the weights according to the algorithm described in subsection 4.2.2, and generates a novel .net file in the SNNS format for the network with weights adjusted. Finally, performance of this network is analyzed using the SNNS tools (*batchman* and *analyze*). This procedure was done in order to guarantee the correctness of the implementation and to assure a fair comparison between the original DDA and our proposed method (using the same tools for the analyzes).

Table 4.7 presents the classification errors on test sets for the OCR datasets. For the RBF-DDA with weights adjustment, the default DDA parameters ($\theta^+ = 0.4$ and $\theta^- = 0.1$) were used in the first phase of training. In the second phase of training – weights adjustment – the learning rate $\eta = 0.1$ was used. This table compares the

Dataset	RBF-DDA (weights adjustment)	RBF-DDA (θ^- selection)	RBF-DDA Default	MLP	k-NN
optdigits	4.45%	2.78%	10.18%	10.95% (1.90%)	3.51%
pendigits	3.43%	2.92%	8.12%	4.83% (0.50%)	2.29%
letter	5.24%	5.30%	15.60%	23.59% (0.90%)	6.62%
mean across datasets	4.37%	3.67%	11.30%	13.12%	4.14%

Table 4.7. Classification errors on test sets for OCR datasets: comparing two of the proposed methods for improving RBF-DDA

Dataset	Training epochs (DDA)	No. of RBFs	% of training patterns	Tr. error (DDA)	Additional training epochs	Tr. error (after weights adj.)
optdigits	4	1,953	51.09%	6.12%	29	3.77%
pendigits	5	1,427	19.04%	3.10%	115	0.61%
letter	5	7,789	51.93%	6.51%	26	1.04%

Table 4.8. Training results before and after weights adjustments for OCR datasets

performance of the weights adjustment method with RBF-DDA trained with default parameters, RBF-DDA with θ^- selection, MLP and k-NN (results reported on table 4.2).

The results of table 4.7 show that the method of weights adjustment also considerably improves performance of RBF-DDA with default parameters. The proposed method also outperforms previous MLP results on all datasets considered [KA00, KGC00, FS91]. It outperforms previous k-NN results on the *letter* dataset and obtains slightly worse results on the two remaining datasets. The θ^- selection method performs better than the weights adjustment one on two of the datasets (*optdigits* and *pendigits*). These methods achieve similar performance on *letter*.

It is important to stress that the weights adjustments were carried out on RBF-DDA networks trained with default parameters. The training epochs, number of hidden RBF units, percentage of training patterns stored as prototypes (RBF units) and training error of the default RBF-DDA on the OCR datasets are shown in table 4.8. This table also presents the additional training epochs used in the weights adjustments and the error on the training set after the adjustment. It can be seen that few additional epochs are needed for weights adjustment (from 26 to 115). Notice that all patterns of the reduced training set are considered for adjusting the weights in one epoch of training. The results also show that the training errors of the RBF-DDA networks are much reduced by weights adjustment. This led to improved networks regarding generalization performance, as the results of table 4.7 have demonstrated.

Although the performance of the RBF-DDA with θ^- selection is better than that of the weights adjustment method, the former method produces much larger networks. The number of hidden units in these networks are compared in table 4.4. This table shows that θ^- selection produces networks with a number of hidden units that correspond to 76.37% to 99.71% of the training patterns for the OCR datasets (87.27% mean across the

θ^-	RBF-DDA default	RBF-DDA weights adjustment	hidden units	% of training patterns
10^{-1}	10.18%	4.45%	1,953	51.09%
10^{-2}	4.79%	2.45%	3,318	86.79%
10^{-3}	3.45%	2.73%	3,697	96.70%
10^{-4}	3.06%	2.68%	3,784	98.98%
10^{-5}	2.78%	2.67%	3,812	99.72%

Table 4.9. Classification errors on *optdigits* test set using RBF-DDA with weights adjustment

three datasets). On the other hand, the weights adjustment method produces networks with 19.04% to 51.93% of the training patterns stored as prototypes in the hidden layer (40.69% mean across the datasets).

As discussed in the previous section, the space requirement of a classifier can be as important as its performance. The RBF-DDA with weights adjustment offers an alternative to RBF-DDA with θ^- selection. The use of one or the other in practice depends on the trade-off between performance and space complexity.

A second set of simulations with the weights adjustment method proposed here for RBF-DDA networks with different values for the θ^- DDA parameter were carried out. The results are shown in tables 4.9, 4.10, and 4.11 for *optdigits*, *pendigits* and *letter* respectively. Each of these tables compares the classification error on the test set before and after the application of the proposed weights adjustment. The tables also show the number of hidden units for each θ^- together with the percentage of training patterns stored as prototypes. For each dataset, θ^- is varied from its default value (10^{-1}) to the near-optimal value obtained by the θ^- selection method (see table 4.4).

The results of these tables show that the weights adjustment method proposed in this work improves RBF-DDA performance for all θ^- values considered, though the improvement is smaller for smaller θ^- values. The tables also show that the size of the resulting network increases with decreasing θ^-. It can be seen that training RBF-DDA networks with $\theta^- = 10^{-2}$ and next adjusting the weights as proposed in work results in networks with good performance and moderate size. Finally, table 4.12 summarizes these results and shows that networks trained with $\theta^- = 10^{-2}$ followed by weights adjustment achieve the best generalization performance in the datasets considered. Moreover, the method based on weights adjustment produces much smaller networks than the method based solely on θ^- selection.

Experiments were also performed using the *soybean* datasets. The RBF-DDAs with weights adjustment were also trained with default parameters ($\theta^+ = 0.4$ and $\theta^- = 0.1$) in the first phase of training. Next, weights were adjusted using the learning rate $\eta = 0.1$. The classification errors on test sets are shown in table 4.13. This table also presents the previous results obtained by RBF-DDA with default parameters (without weights adjustment) and by θ^- selection. RBF-DDA with θ^- selection performs better in two *soybean* partitionings whereas the method based on weights adjustment is better on the remaining one. Both methods considerably improve RBF-DDA performance on these datasets as well. It can also be seen that the mean classification error on test sets

θ^-	RBF-DDA default	RBF-DDA weights adjustment	hidden units	% of training patterns
10^{-1}	8.12%	3.43%	1,427	19.04%
10^{-2}	4.09%	2.74%	3,108	41.47%
10^{-3}	3.29%	2.72%	4,363	58.22%
10^{-4}	2.94%	2.80%	5,172	69.02%
10^{-5}	2.92%	2.78%	5,723	76.37%

Table 4.10. Classification errors on *pendigits* test set using RBF-DDA with weights adjustment

θ^-	RBF-DDA default	RBF-DDA weights adjustment	hidden units	% of training patterns
10^{-1}	15.60%	5.24%	7,789	51.93%
10^{-2}	6.58%	4.96%	11,025	73.50%
10^{-3}	5.44%	5.04%	12,232	81.55%
10^{-4}	5.30%	5.14%	12,861	85.74%

Table 4.11. Classification errors on *letter* test set using RBF-DDA with weights adjustment

Dataset	RBF-DDA Default	RBF-DDA (weights adj., $\theta^- = 10^{-1}$)	RBF-DDA (θ^- selection)	RBF-DDA (weights adj., $\theta^- = 10^{-2}$)
optdigits	10.18%	4.45%	2.78%	**2.45%**
pendigits	8.12%	3.43%	2.92%	**2.74%**
letter	15.60%	5.24%	5.30%	**4.96%**
mean across datasets	11.30%	4.37%	3.67%	**3.38%**

Table 4.12. Classification errors on test sets: comparison of different RBF-DDA classifiers

Dataset	RBF-DDA (weights adjustment)	RBF-DDA (θ^- selection)	RBF-DDA (Default)
soybean1	14.12%	10.59%	32.94%
soybean2	8.82%	8.24%	24.71%
soybean3	9.41%	13.53%	34.12%
mean across datasets	10.78%	10.79%	30.59%

Table 4.13. Classification errors on test sets for soybean datasets: comparing the proposed methods for improving RBF-DDA

Dataset	Training epochs (DDA)	No. of RBFs	% of training patterns	Tr. error (DDA)	Additional training epochs	Tr. error (opt.)
soybean1	4	221	43.10%	11.50%	19	0.97%
soybean2	4	226	44.05%	8.77%	14	1.56%
soybean3	4	220	42.88%	9.16%	20	1.56%

Table 4.14. Training results before and after weights adjustment for soybean datasets

k	θ^+	θ^-	hidden RBF units	% tr. patterns stored	Test set error
No reduction	0.4	0.1	1,953	51.08%	10.18%
No reduction	0.4	10^{-5}	3,812	99.71%	2.78%
27	0.4	10^{-2}	1,628	42.58%	1.78%

Table 4.15. Classification errors on test set of *optdigits* for RBF-DDA classifiers

across the different partitionings is almost the same for the two methods proposed here for improving RBF-DDA (around 10.80%). This is much better than the mean error obtained by the original DDA trained with default parameters (30.59%).

Finally, table 4.14 presents results regarding both training phases of the weights adjustment method on the *soybean* datasets. This table contains the number of epochs of DDA training, the number of hidden RBF units, the percentage of patterns of the training set stored as prototypes, and the classification error on the complete training set. These data concern the first phase of training in which the networks are built by using DDA with default parameters. The table also shows the number of epochs for the second phase of training (weights adjustment) and the final training set classification error. This error is also much decreased by the second phase of training proposed here.

4.3.4 Results and Discussion for the Integrated Method: Data Reduction and Parameter Selection

The training method proposed in subsection 4.2.3 is intended to build RBF networks with improved generalization capabilities without increasing the resulting networks. This subsection reports experiments carried out in order to assess the performance of the proposed method. The first four datasets of table 4.1, namely, *optdigits*, *pendigits*, *letter*, and *satimage*, were used in the following experiments. They were selected for these experiments because they are the larger datasets of table 4.1, and therefore, produce larger RBF networks.

Tables 4.15, 4.16, 4.17, and 4.18 compare performance of integrated method of subsection 4.2.3 with RBF-DDA trained without data reduction for *optdigits*, *pendigits*, *letter*, and *satimage*, respectively. Each of these tables contains k (the parameter of the data reduction algorithm), θ^+, θ^-, the number of hidden units of the RBF network built by DDA, the percentage of training patterns stored as RBF units and the classification error on the test set. Notice that the percentage of training patterns stored as RBF units is computed with respect to the full training set.

k	θ^+	θ^-	hidden RBF units	% tr. patterns stored	Test set error
No reduction	0.4	0.1	1,427	19.04%	8.12%
No reduction	0.4	10^{-5}	5,723	76.37%	2.92%
18	0.4	10^{-2}	1,430	19.08%	4.86%
60	0.4	10^{-2}	2,338	31.20%	2.94%
60	0.4	10^{-4}	3,130	41.77%	2.63%
80	0.4	10^{-5}	3,689	49.22%	2.60%

Table 4.16. Classification errors on test set of *pendigits* for RBF-DDA classifiers

k	θ^+	θ^-	hidden RBF units	% tr. patterns stored	Test set error
No reduction	0.4	0.1	7,789	51.93%	15.60%
No reduction	0.4	10^{-4}	12,861	85.74%	5.30%
12	0.4	10^{-2}	7,826	52.17%	5.18%
16	0.4	10^{-2}	8,582	57.21%	5.12%

Table 4.17. Classification errors on test set of *letter* for RBF-DDA classifiers

The first line of each table presents results obtained by RBF-DDA using default parameters without data reduction. The results of the second line were also obtained by RBF-DDA without data reduction, this time trained with the method based solely on θ^- selection (proposed in subsection 4.2.1) [ONM04c]. The remaining lines present the results obtained by firstly reducing the training set and then training with DDA. The selection of k and θ^- was carried out by cross-validation, as outlined in subsection 4.2.3. For some datasets more than one combination of k and θ^- is presented. The combination which yielded the best performance is shown in the last line of each table.

The results of tables 4.15, 4.16, 4.17, and 4.18 show that the method based solely on θ^- selection considerably improves RBF-DDA performance for the four datasets considered, however, this method always leads to larger networks than those generated by RBF-DDA trained with default parameters [ONM04c]. These tables also show that the integrated method proposed in subsection 4.2.3 achieves better generalization performance than both default RBF-DDA and RBF-DDA trained with selected θ^- [ONM04c]. Notice that, at the same time, the proposed method is able to produce much smaller networks than those produced by the RBF-DDA trained with θ^- selection (without data reduction).

For example, for the dataset *optdigits* (results in table 4.15), an RBF-DDA trained with the full training set and default parameters has achieved 10.18% classification error on test set with 51.08% of the training patterns stored as RBF units. The method based only on θ^- selection (without data reduction) is able to improve the generalization performance, reducing the classification error to 2.78%. This is obtained, however, at the expense of increasing the percentage of stored training patterns to 99.71%. The integrated method proposed in subsection 4.2.3, in contrast, improves performance (the error is decreased to 1.78%) and, at the same time, decreases the network size (the number of training patterns stored is 42.58% in this case) [OMNM04].

Table 4.19 compares the generalization performance of the proposed method with other classifiers for each dataset. This table presents results obtained with RBF-DDA

k	θ^+	θ^-	hidden RBF units	% tr. patterns stored	Test set error
No reduction	0.4	0.1	2,812	63.40%	14.95%
No reduction	0.4	10^{-4}	4,099	92.42%	8.55%
46	0.4	10^{-3}	2,759	62.21%	8.55%
75	0.4	10^{-3}	3,090	69.67%	8.25%

Table 4.18. Classification errors on test set of *satimage* for RBF-DDA classifiers

Dataset	RBF-DDA (integrated method)	RBF-DDA ($\theta^- sel.$)	RBF-DDA Default	MLP	k-NN
optdigits	**1.78%**	2.78%	10.18%	10.95%	3.51%
pendigits	2.60%	2.92%	8.12%	4.83%	**2.29%**
letter	**5.12%**	5.30%	15.60%	23.59%	6.62%
satimage	**8.25%**	8.55%	14.95%	13.90%	9.40%
mean across datasets	4.44%	4.89%	12.21%	13.32%	5.45%

Table 4.19. Classification errors on test sets: comparison of the integrated method with other classifiers

using data reduction; RBF-DDA without data reduction using θ^- selection [ONM04c]; RBF-DDA without data reduction using default parameters; multi-layer perceptrons (MLP) and k-nearest neighbor (k-NN). The k-NN and MLP results were obtained in the literature, as described in subsection 4.3.2. As commented before, the proposed integrated method outperforms RBF-DDA trained with both default parameters and selected θ^-. Furthermore, the proposed method produces much smaller networks than those of RBF-DDA trained with selected θ^- and comparable in size to those of default RBF-DDA.

The integrated method for training RBF networks proposed in this thesis markedly outperforms MLPs in all datasets considered, as can be seen in table 4.19. The proposed method outperformed k-NN on three of the datasets and had similar performance on the remaining (*pendigits*). The proposed method has an important advantage over k-NN. k-NN needs to store all training data whereas the proposed method needs to store only from 42.58% to 69.67%, depending on the dataset (see tables 4.15, 4.16, 4.17, and 4.18).

Finally, table 4.20 compares the results obtained by the integrated method with those of the method based on weights adjustment (of subsection 4.2.2). The methods are compared regarding both the generalization performance and the size of the networks. The results of this table show that the integrated method achieved mean performance slightly better than the method of weights adjustment with $\theta^- = 10^{-2}$ with the advantage of producing smaller networks.

Dataset	RBF-DDA (integrated method)	RBF-DDA (weights adj., $\theta^- = 10^{-1}$)	RBF-DDA (weights adj., $\theta^- = 10^{-2}$)
optdigits	1.78% [1628]	4.45% [1953]	2.45% [3318]
pendigits	2.63% [3130]	3.43% [1427]	2.74% [3108]
letter	5.12% [8582]	5.24% [7789]	4.96% [11025]
mean across datasets	3.18% [4446.7]	4.37% [3723]	3.38% [5817]

Table 4.20. Comparison of the integrated method with the method based on weights adjustment (classification error on test sets and number of hidden RBFs)

4.3.5 Results and Discussion for RBF-DDA-SP: DDA with Selective Pruning and Parameter Selection

This subsection reports experiments carried out using the RBF-DDA-SP training method - which combines selective pruning and parameter selection - proposed in subsection 4.2.4. The experiments considered the six datasets of table 4.1. We consider only the inverted version of the *segment* dataset, which we named *segment2*, and the first partitioning of the *soybean* dataset, *soybean1*.

Figures 4.17 (a) and (b) compare the proposed method (RBF-DDA-SP) with both the original RBF-DDA [BD95] and RBF-DDA-T [Pae04] regarding the generalization performance and the complexity of the networks, respectively. These results were obtained for the *optdigits* dataset. RBF-DDA-SP was training pruning 50% of the neurons which covered only one training sample. The results for RBF-DDA-SP correspond to means over 10 runs of the simulations, because performance depends on the RBFs pruned from the network. In each run of the simulations, 50% of the neurons which covered only one traning sample were removed from the network. The neurons to be removed were randomly selected and each run used a different value for the seed of the random number generator.

It can be observed from figure 4.17 (a) that for $\theta^- = 0.1$, the generalization performance of the methods is similar, with a slight advantage for RBF-DDA. For smaller values of θ^-, the generalization performance improves for both the proposed RBF-DDA-SP and RBF-DDA (up to $\theta^- = 10^{-5}$). On the other hand, as θ^- decreases, performance severely degrades for RBF-DDA-T. This occurs because smaller values of θ^- generate networks with larger number of neurons which cover only one training sample. RBF-DDA-T prunes the network at each training epoch, thereby removing all training samples which cover only one neuron. These training samples are put on an outlier list and are not considered in subsequent training epochs [Pae04]. This results in much severe pruning for small values of θ^-. For the *optdigits* dataset, RBF-DDA-T generated RBF networks with only one hidden unit for $\theta^- = 10^{-9}$ and $\theta^- = 10^{-10}$ with heavily degraded performance, as can be seen in figure 4.17 (a).

On the other hand, the pruning strategy adopted by RBF-DDA-SP removes only a portion of those neurons which are considered to cover only one training sample, thereby

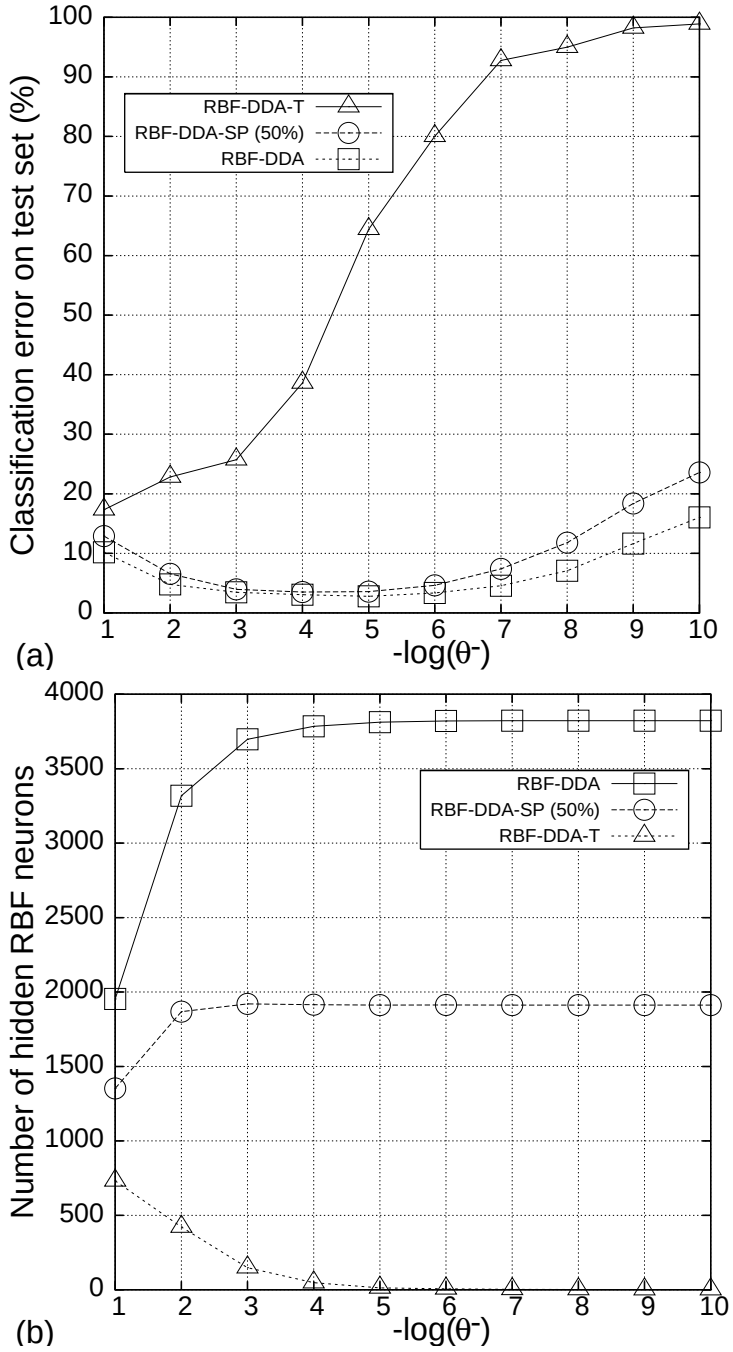


Figure 4.17. Comparison of the proposed method (RBF-DDA-SP) with RBF-DDA and RBF-DDA-T as function of θ^-: (a) classification errors. (b) number of RBFs. Results on *optdigits*.

producing networks with better generalization performance. In addition, RBF-DDA-SP pruning is carried out only at the last training epoch, thereby avoiding premature pruning. A similar behavior to that depicted in figures 4.17 (a) and (b) has also been observed for the other datasets considered in the experiments of this subsection, namely, *pendigits*, *letter*, *satimage*, *segment2* and *soybean1*.

The results of figures 4.17 (a) and (b) show that RBF-DDA-SP offers a much better compromise between generalization performance and complexity than both RBF-DDA and RBF-DDA-T. RBF-DDA-SP achieves classification performance comparable to RBF-DDA for $\theta^- = 10^{-5}$ (3.57% classification error versus 2.78%) using only 1912 hidden units whereas RBF-DDA needed 3812 hidden units. On the other hand, the better results obtained by RBF-DDA-T were 14.75% classification error with only 655 hidden units, for $\theta^- = 0.1$. Smaller values of θ^- heavily degraded RBF-DDA-T performance.

Tables 4.21 and 4.22 compare the classification performance and the complexity of the networks of the proposed method (RBF-DDA-SP) with RBF-DDA trained with default parameters ($\theta^+ = 0.4$ and $\theta^- = 0.1$) [BD95], RBF-DDA-T [Pae04] and RBF-DDA with θ^- selection (see subsection 4.2.1 and [ONM04c]) in each dataset. For each OCR dataset, table 4.21 shows both the classification error on the test set and the number of hidden RBF units, for each training method. Table 4.22 shows the results for the remaining datasets, namely, *satimage*, *segment2*, and *soybean1*.

The experiments reported here considered RBF-DDA-SP with three different percentages of pruning, namely, 30%, 40%, and 50%. For example, in the case of RBF-DDA-SP with 30% of pruning, the method prunes, after DDA training, 30% of the neurons which cover only one training sample. For RBF-DDA-SP, the simulations were carried out ten times for each dataset, since the method randomly selects the neurons to be pruned. Tables 4.21 and 4.22 report both the mean and the standard deviation of the classification errors for RBF-DDA-SP.

RBF-DDA-T simulations were carried out with $\theta^+ = 0.4$ and both $\theta^- = 0.1$ and $\theta^- = 0.2$ for each dataset. Recall that Paetz suggests using $\theta^- = 0.2$ for RBF-DDA-T (see subsection 2.2.2.2 and [Pae04]). Tables 4.21 and 4.22 show only the best RBF-DDA-T classification results obtained for each dataset (which used $\theta^- = 0.2$ for *optdigits* and *satimage*, and $\theta^- = 0.1$ for *pendigits*, *letter*, *segment2* and *soybean1*). The values of θ^- for RBF-DDA with θ^- selection and for RBF-DDA-SP for each dataset were: *optdigits*, *pendigits* and *soybean1*, ($\theta^- = 10^{-5}$); *letter*, *satimage* and *segment2*, ($\theta^- = 10^{-4}$).

The results of tables 4.21 and 4.22 show that RBF-DDA-SP considerably outperforms both RBF-DDA with default parameters and RBF-DDA-T for the three percentages of pruning considered. In addition, it can be observed that the proposed method achieves classification performance more close to that of RBF-DDA with θ^- selection, with the advantage of generating much smaller networks. RBF-DDA-SP performance is higher for smaller amounts of pruning (e.g., 30%). On the other hand, higher pruning rates produce networks with less neurons and a slight degradation in performance. These results show that RBF-DDA-SP offers a better compromise between generalization performance and network complexity than both RBF-DDA and RBF-DDA-T for all six datasets considered in the experiments.

Finally, table 4.23 compares RBF-DDA-SP with the integrated method proposed in

Method	optdigits	pendigits	letter
RBF-DDA (default)	10.18% [1953]	8.12% [1427]	15.60% [7789]
RBF-DDA-T (default)	14.75% [655]	8.43% [978]	25.32% [2837]
RBF-DDA (θ^- sel.)	2.78% [3812]	2.92% [5723]	5.30% [12861]
RBF-DDA-SP (30%, θ^- sel.)	3.13% [2672] (0.23%)	3.04% [4344] (0.14%)	6.54% [9358] (0.18%)
RBF-DDA-SP (40%, θ^- sel.)	3.30% [2292] (0.28%)	3.17% [3884] (0.18%)	7.10% [8191] (0.12%)
RBF-DDA-SP (50%, θ^- sel.)	3.57% [1912] (0.26%)	3.29% [3424] (0.19%)	8.00% [7023] (0.25%)

Table 4.21. Classification errors on test sets and number of hidden RBFs for OCR dataset

Method	satimage	segment2	soybean1
RBF-DDA (default)	14.95% [2812]	18.10% [568]	32.94% [221]
RBF-DDA-T (default)	24.75% [662]	8.10% [347]	39.41% [85]
RBF-DDA (θ^- sel.)	8.55% [4099]	0.00% [1088]	10.59% [366]
RBF-DDA-SP (30%, θ^- sel.)	9.18% [2934] (0.27%)	1.09% [884] (0.67%)	12.38% [279] (1.04%)
RBF-DDA-SP (40%, θ^- sel.)	9.59% [2546] (0.32%)	1.62% [816] (0.75%)	12.41% [250] (1.02%)
RBF-DDA-SP (50%, θ^- sel.)	10.05% [2157] (0.40%)	2.76% [748] (1.30%)	13.82% [220] (1.76%)

Table 4.22. Classification errors on test sets and number of hidden RBFs for remaining datasets

Dataset	RBF-DDA (integrated method)	RBF-DDA-SP (θ^- *sel.*)
optdigits	1.78% [1628]	3.57% [1912]
pendigits	2.60% [3689]	3.29% [3424]
letter	5.12% [8582]	7.10% [8191]
satimage	8.25% [3090]	9.18% [2934]

Table 4.23. Comparison of RBF-DDA-SP to the integrated method of subsection 4.2.3

subsection 4.2.3 (results of subsection 4.3.4). This table shows both the classification error on test sets and the size of the network for each dataset and each method. The RBF-DDA-SP results were obtained from tables 4.21 and 4.22. For each dataset we show only the RBF-DDA-SP result which produced a network with approximately the same size as that of the integrated method. The results of table 4.23 show that the integrated method achieves better generalization performance for all datasets considered (producing networks of approximately the same size as RBF-DDA-SP). On the other hand, RBF-DDA-SP is less expensive computationally since the pruning method employed by it is much simpler than the data reduction algorithm of the integrated method.

4.3.6 Comparison with AdaBoost and SVM

This section compares the generalization performance of the two best methods proposed in this chapter (in terms of generalization performance) - namely, RBF-DDA with θ^- selection (subsection 4.2.1) and the integrated method of subsection 4.2.3 - with AdaBoost and support vector machines. The results of AdaBoost and support vector machines were obtained in recent papers available in the literature [KA00, ASS00, CHBK04, RK04]. The datasets considered in the comparison are *optdigits*, *pendigits*, *letter* and *satimage*.

Table 4.24 compares performance of two of the proposed methods for improving RBF-DDA with AdaBoost and SVM results. The first column of this table shows the best AdaBoost results from MLP ensembles starting from two MLP components to 100 MLP components [KA00]. The second column of the table shows the best AdaBoost results reported in a previous work that compared a number of decoding methods [ASS00]. The results shown here are the best obtained in that study, corresponding to AdaBoost with loss-based decoding using the exponential loss function and all-pairs output codes [ASS00].

It can be seen that the constructive training methods for RBF networks proposed in this thesis outperform AdaBoost for all datasets considered. The classifiers proposed here also outperformed a recent ensemble classifier named DIvote on the *pendigits* datasets [CHBK04]. This classifier obtained 3.60% classification error on the test set [CHBK04] whereas RBF-DDA with θ^- selection achieved 2.92% and the integrated method of subsection 4.2.3 achieved 2.60%.

The SVM results shown in table 4.24 are the best results for these datasets obtained

Dataset	AdaBoost (1)	AdaBoost (2)	SVM (OVA)	SVM (Sparse)	RBF-DDA (θ^- selection)	Integrated met. (subsec. 4.2.3)
optdigits	4.67%	-	2.73%	3.01%	2.78%	1.78%
pendigits	3.17%	2.90%	2.50%	2.70%	2.92%	2.60%
letter	19.91%	7.10%	2.75%	3.55%	5.30%	5.12%
satimage	-	11.40%	7.80%	8.85%	8.55%	8.25%

Table 4.24. Classification errors on test sets: comparison with AdaBoost and SVMs. (1) results from [KA00] and (2) results from [ASS00]

in two previous studies [ASS00, RK04]. This table shows SVM results using both one-vs-all (OVA) and sparse output coding [ASS00, RK04]. The *optdigits*, *letter* and *satimage* results were obtained in the work of Rifkin and Klautau [RK04] whereas results for *pendigits* were obtained in the work of Allwein et al. [ASS00].

Notice that the integrated method proposed in this chapter for improving RBF-DDA outperforms SVM in the *optdigits* dataset, obtains a similar performance in the *pendigits* and *satimage* datasets and obtains worse results in *letter*. The RBF-DDA with θ^- selection achieves comparable results to SVMs in the *optdigits*, *pendigits* and *satimage* datasets. In the *letter* dataset, SVM obtains better generalization performance. It is important to stress that other output coding methods for SVMs considered in previous works lead to worse performance compared to our RBF-DDA methods [ASS00, RK04].

Support vector machines are a powerful class of machine learning algorithms shown to achieve good performance on many classification and regression tasks. However, a recent study showed that simpler methods can be very competitive with SVMs on a number of datasets [MLH03]. The methods proposed in this chapter have built RBF networks that can achieve generalization performance comparable to SVMs in a number of tasks. Moreover, they can be faster to train then SVMs, which can be an important advantage in practical applications.

4.3.7 Experiments Using Cross-Validation

The experiments reported in the last subsections used the *holdout method* to evaluate and compare the performance of the classifiers. Nonetheless, it is argued that the *cross-validation* method should be used to enhance such comparison [DHS00, TK03]. We have used the holdout method because it is possible to compare the results to a number of other results obtained by other methods reported in the literature which used the same training and test sets.

In *n-fold cross-validation* the data available for simulations (training and test patterns used in holdout) are merger to form a single dataset. Next, this dataset is divided into n disjoint subsets. Each subset is called a fold. The process of training and test is carried out n times. Each time, one of the folds is used as test set and the remaining folds are used to form the training set. The *cross-validation error* is the mean of the n classification errors on test sets obtained in this process [DHS00, TK03].

By using cross-validation instead of holdout, it is possible to compute the standard

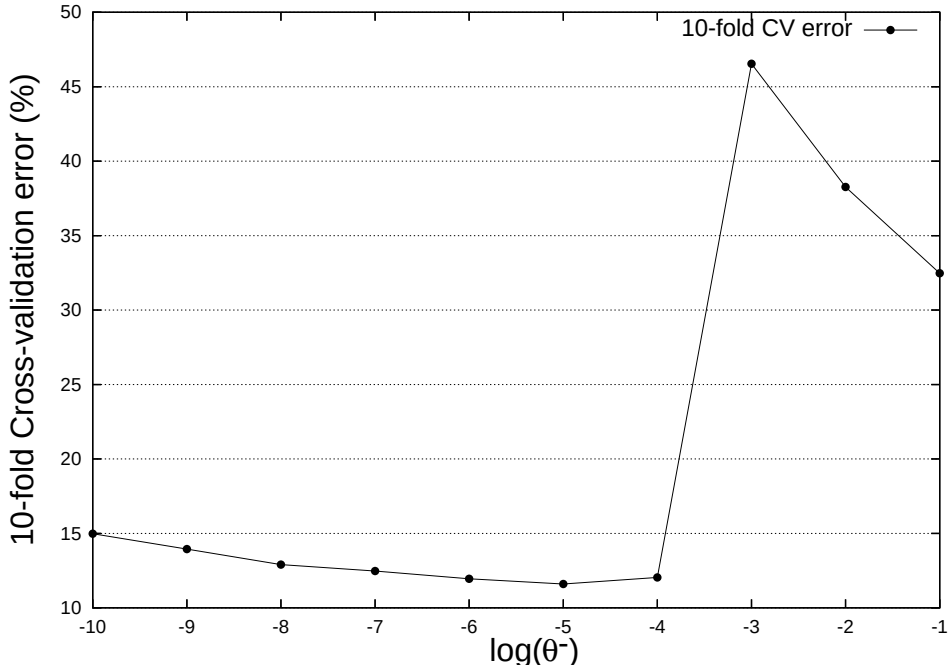


Figure 4.18. 10-fold cross-validation error for the *segment* dataset as a function of θ^-

deviation of the cross-validation error and, more importantly, to use statistical hypothesis tests in order to assess whether the difference in the performance of two classifiers is statistically significant.

We have performed a number of cross-validation simulations using only the two best methods proposed in this chapter (in terms of generalization performance), namely, RBF-DDA with θ^- selection and the integrated method of subsection 4.2.3. All simulations have used 10-fold cross-validation.

Figure 4.18 shows the mean 10-fold cross-validation error for the *segment* dataset. The dataset used in these experiments was formed by merging the training and test datasets of UCI [BM98]. This produced a dataset with 2310 patterns. Thus, each fold had 231 patterns. The results of figure 4.18 are similar to those of the holdout simulations with segment datasets (see figures 4.4 and 4.5). The cross-validation error increases as θ^- is decreased. This takes place up to $\theta^- = 10^{-3}$. For $\theta^- = 10^{-4}$ there is a considerable improvement in performance, which is slightly degraded for smaller values of θ^-. The behavior of the cross-validation error as a function of θ^- for the other datasets is also similar to the respective behavior using the holdout method.

Table 4.25 reports 10-fold cross-validation errors for the *optdigits*, *pendigits*, *letter*, *satimage* and *segment* datasets. This table compares the performance of k-NN, the default RBF-DDA, RBF-DDA with θ^- selection and the integrated method. k-NN experiments were carried out using $k = 3$ for the first four datasets. For *segment*, we have tried $k = 1$, $k = 2$ and $k = 3$. We report only the results for $k = 1$, since these were the best results for this dataset. The results of table 4.25 show that both methods proposed in this chapter considerably improve performance with respect to the original RBF-DDA.

Dataset	k-NN	RBF-DDA (default)	RBF-DDA (θ^- selection)	Integrated method (subsection 4.2.3)
optdigits	2.28% (1.07%)	9.11% (1.03%)	1.87% (0.74%)	1.53% (0.56%)
pendigits	0.77% (0.34%)	4.45% (1.01%)	0.84% (0.27%)	1.05% (0.32%)
letter	4.74% (0.50%)	14.50% (1.03%)	4.68% (0.58%)	4.72% (0.53%)
satimage	11.30% (3.47%)	15.57% (4.71%)	10.46% (3.70%)	10.10% (3.76%)
segment	10.39% (0.81%)	32.47% (18.41%)	12.04% (8.85%)	12.38% (7.69%)

Table 4.25. 10-fold cross-validation errors (means and standard deviations): comparison of classifiers

In addition, they show that these methods achieve performance comparable to k-NN, with the advantage of producing much smaller networks. In this regard, the integrated method offers greater advantages, since we have shown that it is able to produce much smaller networks than those produced by RBF-DDA with θ^- selection (see subsection 4.3.4).

The 10-fold cross-validation experiments are described in more detail in appendix A. This appendix shows the test error obtained in each fold for each dataset and each method. Moreover, it reports on statistical hypothesis tests performed in order to decide whether the differences in performance between the classifiers are statistically significant. The hypothesis tests were carried out using 95% confidence level.

The hypothesis tests of appendix A show that, for the five datasets considered, k-NN and RBF-DDA with θ^- selection achieve the same performance. On the other hand, it is shown that k-NN outperforms RBF-DDA with default parameters for all datasets. Moreover, it is shown in appendix A that both RBF-DDA with θ^- selection and the integrated method outperform RBF-DDA trained with default parameters for all datasets.

Hypothesis tests were also performed for comparing k-NN to the integrated method proposed in this chapter. The results of appendix A show that the integrated method outperforms k-NN for the datasets *optdigits* and *satimage*. On the other hand, k-NN outperforms the integrated method for the dataset *pendigits*. For *letter* and *segment*, the tests have revealed that the difference in performance of these methods is not statistically significant.

Finally, the comparison of the RBF-DDA with θ^- selection and the integrated method detailed in appendix A has shown that the integrated method outperforms RBF-DDA with θ^- selection for the *optdigits* and *satimage* datasets. For the other datasets, the hypothesis tests have shown that there is no statistical difference between these classifiers.

4.4 CONCLUSION

This chapter has proposed and investigated four methods for improving the generalization performance of RBF-DDA. The original DDA algorithm is a fast constructive algorithm that effectively uses all training data for training, that is, does not needs validation data. Training is usually carried out with the recommended default parameters $\theta^+ = 0.4$ and $\theta^- = 0.1$, since it has been show, for some datasets, that these parameters

only slightly influence performance [BD95, BD98].

The first method proposed in this chapter was motivated by the observation that parameter θ^- considerably improves performance of RBF-DDA for some datasets. In this method, we divide training data into training and validation pattern sets and use the performance on the latter set to select the value of θ^-. Next, we use the selected θ^- value to train using the full training data. Finally, generalization performance is evaluated on an unseen test set. This method indeed improves generalization performance, yet it also leads to much larger networks.

We have proposed three alternative methods in order to tackle the drawback of the first method. The second method proposed in this chapter is based on a gradient-descent adjustment of the weights of the connections between hidden and output units. In this method an RBF network is firstly built using DDA. Next, the network weights are adjusted by a gradient-descent algorithm using the reduced training set. Performance on the validation set is assessed during training. Training stops when the validation set SSE increases.

The third method proposed in this chapter integrates a data reduction algorithm with the selection of parameter θ^-. Firstly, the data reduction algorithm is applied to the training set in order to select the most important patterns for discrimination. Next, an RBF network is built by DDA using the reduced training set. The data reduction algorithm employed has a parameter k which determines the reduction rate. Hence, the proposed method has two critical parameter, namely θ^- and k, which are selected via cross-validation.

Finally, the four method, which we named RBF-DDA-SP, combines selective pruning with θ^- selection. The method was inspired by RBF-DDA-T, an extension of RBF-DDA recently proposed in the literature [Pae04]. We have demonstrated that RBF-DDA-T leads to much severe pruning for smaller values of θ^-. On the other hand, RBF-DDA-SP, introduced here, prunes only a portion of the neurons which cover only one training pattern, thereby generating networks which offer much better generalization performance compared to RBF-DDA-T.

The proposed methods have been evaluated on six benchmark datasets obtained from the UCI repository [BM98]. The experiments used both the holdout and the cross-validation methods to compare the classifiers. The experiments have shown that all four methods proposed in this chapter considerably improve generalization performance with respect to the original RBF-DDA.

In addition, the experiments have shown that the first method (RBF-DDA with θ^- selection) improves performance at the expense of generating much larger networks. The three alternative methods proposed in this chapter are able to produce smaller networks than the first method. Nonetheless, the second and fourth method achieve this goal at the expense of a slight degradation in performance (with respect to the first method). On the other hand, the third method (the integrated method) was able to reduce the size of the networks and, simultaneously achieve results comparable or even better than those of RBF-DDA with θ^- selection.

The results obtained by the first and third methods proposed in this chapter have also been compared to results of other classifiers obtained in the literature. We have

compared our methods to results obtained with MLPs, kNN, AdaBoost and SVMs. The comparison has shown that our methods outperforms MLP and AdaBoost in five of the datasets considered (the only exception was the *soybean* dataset). In addition, the results have shown that our methods achieve results comparable to both kNN and SVM in the datasets considered. Moreover, our methods produce smaller classifiers than kNN, which can be an advantage in practical applications.

The main subject of this thesis is the problem of novelty detection in time series. In chapter 5 we present methods proposed to tackle this problem based on classification. During research, we have observed that performance of RBF-DDA in that problem could be considerably improved by carefully selecting the value of θ^-. This has motivated work on methods for improving RBF-DDA performance. This chapter has proposed and investigated four methods that were evaluated using six benchmark classification datasets. This was done in order to demonstrate that the methods apply to more general datasets and in order to compare the results to other results available in the literature. Two of the methods investigated in this chapter are also applied to the problem of novelty detection in time series in chapter 5 and shown to be valuable in that case as well.

CHAPTER 5

METHODS FOR NOVELTY DETECTION IN TIME SERIES BASED ON CLASSIFICATION

5.1 INTRODUCTION

Forecasting-based techniques for novelty detection in time series have been criticized because of the difficulty to establish thresholds for the detection of novelties. In the next chapter we contribute to the solution of this problem by proposing techniques developed for improving forecasting-based novelty detection in time series. In this chapter we present classification-based techniques for novelty detection in time series. The techniques developed here differ from those of recent works (reviewed in chapter 3) in that they were designed for novelty detection in financial time series. In this domain, we are interested in detecting more subtle deviations from normality. This is so because in many cases the frauds in accountancy and payroll time series, for example, represent small deviations. Additionally, the occurrence of short time series is frequent in financial systems.

Classification-based novelty detection techniques for time series are designed to classify a given time series window into *normal* or *novelty*. A time series window is formed by w consecutive data points extracted from the time series. The advantage of classification-based methods over forecasting-based ones is that classification is a task usually easier than forecasting. On the other hand, classification-based techniques are able to detect novelty in a whole window from the time series, whereas forecasting-based techniques can detect novelty on a single data point from the time series. This can be an advantage on real-world applications, for example, in fraud detection in financial systems.

In this chapter, two novel classification-based techniques are proposed for novelty detection in time series. Both techniques are based on the idea, introduced here, of defining an *envelope* around the time series windows. Time series windows within the envelope are considered normal whereas novelties are windows with data points outside the envelope. The rationale for the use of the envelope in these techniques is to take into account the uncertainty about the future behavior of the time series.

The first novelty detection method presented here is based on the negative samples approach [SM04]. In this technique, negative samples are artificially generated and added to the training set in order to represent novelties and enable the classifier to be trained to correctly classify novelties. The method proposed in this work adapts this technique to novelty detection in short time series modifying it by introducing generation of both *novelty* random patterns (negative samples) and *normal* random patterns. These patterns are then added to training and test sets. Normal random patterns represent small deviations from real normal patterns and are created to increase the datasets in the hope of improving neural network classification performance for short time series

[ONM03, ONM04a, ONM04b].

This technique was tested on a number of real-world time series. Performance of the system was evaluated using two different RBF-DDA architectures. The influence of time series differencing and of the number of random patterns added to the training set in classification performance was also evaluated [ONM03]. The influence of different neural networks classifiers on performance was studied as well [ONM04a]. The classifiers considered include MLP; RBF-DDA; two of the improved methods based on RBF-DDA introduced in chapter 4; machine committees of MLPs; machine committees of MLPs and RBF-DDA neural networks and support vector machines [ONM04a, ONM04b].

The performance of the technique based on negative samples depends on the number of artificial negative samples added to the training set. To avoid the problem of defining this number an alternative technique that does not use negative samples is also proposed in this chapter [ONM04d]. The proposed technique is based on RBF-DDA neural networks. For each time series window in the training set only two additional patterns are generated and added to the training set in order to help the classifier to *learn* the envelope. A number of experiments were carried out in order to compared this technique with that based on negative samples and the results are given in this chapter.

5.2 THE USE OF ENVELOPES FOR NOVELTY DETECTION IN TIME SERIES

The time series novelty detection methods proposed in this chapter work by classifying time series windows as normal or novelty. The methods require fixed length windows, with window size w. A window is formed by w consecutive data points extracted from the time series under analysis. Each training and test pattern to be used must have a fixed length, which correspond to the window size w. The first training pattern will have the first w data points from the time series as its attributes values. To obtain the second pattern one starts with the second data point and uses the next w data points. Similarly, the remaining patterns are obtained by sliding the window by one and taking the next w data points. Thus, if the time series has l data points and a window size of w is used, $l - w + 1$ patterns will be generated. For example, given a time series $\{x_1, \ldots, x_l\}$, the following patterns are generated:

$$\begin{aligned}(x_1, x_2, \cdots x_w)\\ (x_2, x_3, \cdots x_{w+1})\\ \cdots\\ (x_{l-w+1}, x_{l-w+2}, \cdots x_l)\end{aligned}$$

These patterns will later be segregated to obtain training and test sets. The time series is supposed to represent the normal behavior and therefore the training set will have only normal patterns.

In order to define the boundaries between normal and novelty regions, the idea of using an *envelope* is introduced. Given a window from the time series, the idea is to define an envelope around it as shown in figure 5.1. Any time series window with all values

inside the envelope is considered normal. Windows with points outside the envelope are considered novelty. A threshold p_1 is used to define the envelope. Normal patterns are defined by establishing the maximum percent deviation p_1 above and below each data point of a given original pattern.

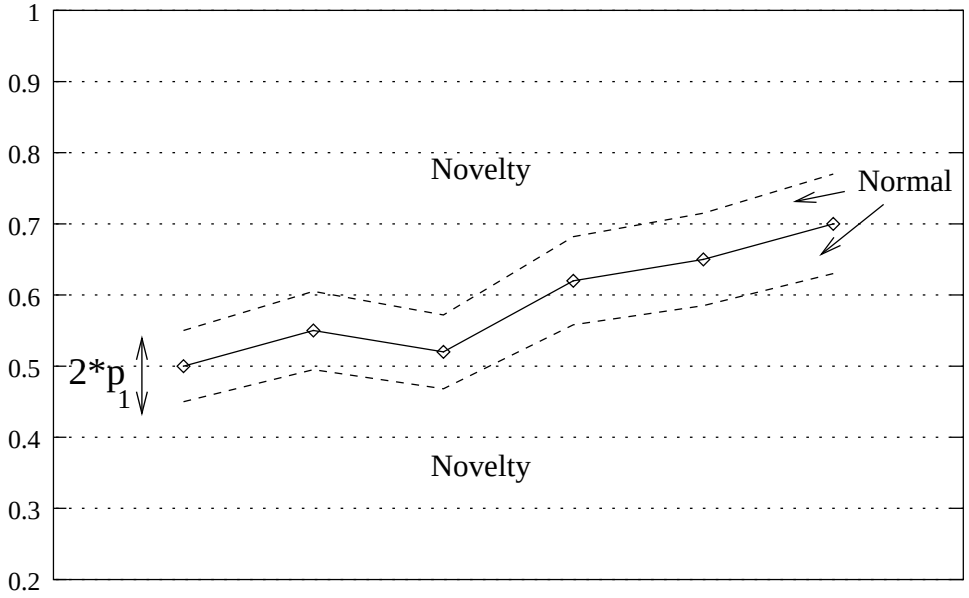


Figure 5.1. The envelope used for the definition of normal and novelty regions for a given time series windows.

The rationale for using the envelope for the definition of normal and novelty regions is to take into account the uncertainty associated with the prediction of future values of the time series. This is an issue that is crucial also in time series forecasting. In this case, it is very important to provide a confidence interval in addition to the prediction of a forecasting system, as discussed in chapter 6 [Mas95].

In order to justify the use of the envelope, a parallel with the theory of stochastic control systems is made. The behavior of most practical systems should be modeled by stochastic models because of the associated uncertainties [May79, WB01]. This is true for example for an aircraft, a chemical process or a national economy [May79]. In control systems, the output of a system can be obtained from its internal state. However, this output is contaminated by measurement noise, inherent to the measurement processes, due mainly to the physical nature of sensors [May79, WB01]. There is also an uncertainty inherent to the system model, because no mathematical system model is perfect [May79]. In stochastic control systems theory uncertainty is modeled by the so-called process noise. In the case of time series forecasting or classification, there is an associated process noise because the models used to learn the time series behavior are not perfect mathematical models. Moreover, the data acquisition process used to obtain the data used to build the time series can introduce errors. Thus, a measurement noise can also appear in the time series novelty detection problem.

The theory of stochastic estimation has been used in time series forecasting [Cha89].

The Kalman filter is the optimal linear estimator used in stochastic estimation. It has been shown that the application of the Kalman filter to the problem of time series forecasting leads to the well known exponential smoothing technique for time series forecasting [Cha89]. In the case of time series novelty detection by classification of time series windows, the envelope is used in order to consider a priori the measurement and process noises that are unavoidable. The process noise in this case is associated with the imperfect models that will be created by the classifier from the training data.

5.3 A METHOD FOR NOVELTY DETECTION WITH NEGATIVE SAMPLES

In this section a method based on the envelope and on negative samples is proposed. In many important applications the occurrence of short time series is frequent. The proposed method is therefore designed to improve detection of novelties in short time series. The use of the procedure described in the previous section to generate patterns from the time series with short time series produces training and test sets with few patterns. In addition, it is assumed that the time series represent the normal behavior and therefore the training set will have only normal patterns. A means to represent novelty is thus needed. The method presented here tackles these problems by adding *random patterns* generated from the normal patterns available in the training set.

Figure 5.2 presents the method proposed here for novelty detection in short time series. The neural network classifier will be trained with an augmented training set containing the real normal patterns extracted from the time series and a number of *normal random patterns* and *novelty random patterns* generated from the real normal patterns. To be considered normal, a random pattern must not deviate much from the original pattern from which it was generated. It must be inside the envelope, as shown in figure 5.1. Besides adding normal random patterns a number of novelty random patterns are also added to the training set, as shown if figure 5.2. Novelty random patterns are patterns that deviate from real normal patterns beyond a given threshold and therefore lie in the novelty regions of figure 5.2. In order to improve classification performance, random patterns should be added in a way that the resulting data set have equal numbers of normal and novelty patterns [Hay98].

Generation of normal and novelty random patterns requires the definition of thresholds. In this work, thresholds were defined as follows: (a) Normal patterns were defined by establishing the maximum percent deviation p_1 above and below each data point of a given original pattern. (b) Novelties also used a second threshold, this time, p_2, to limit the novelty regions in figure 5.1. During training, a pattern is considered a novelty if each of its data points have values between the thresholds p_1 and p_2. This is because in many problems of interests, such as auditing, where novelty is related to possibility of fraud, one is mainly interested in testing network performance in detection of patterns whose deviation from normality is not excessive. For example, in the experiments presented below, the threshold values are $p_1 = 0.1$ and $p_2 = 0.5$, meaning that patterns whose attributes are at most 10% from the normal pattern are considered normal and patterns whose attributes deviates from a normal pattern from 10% to 50% are considered novelty or fraudulent patterns.

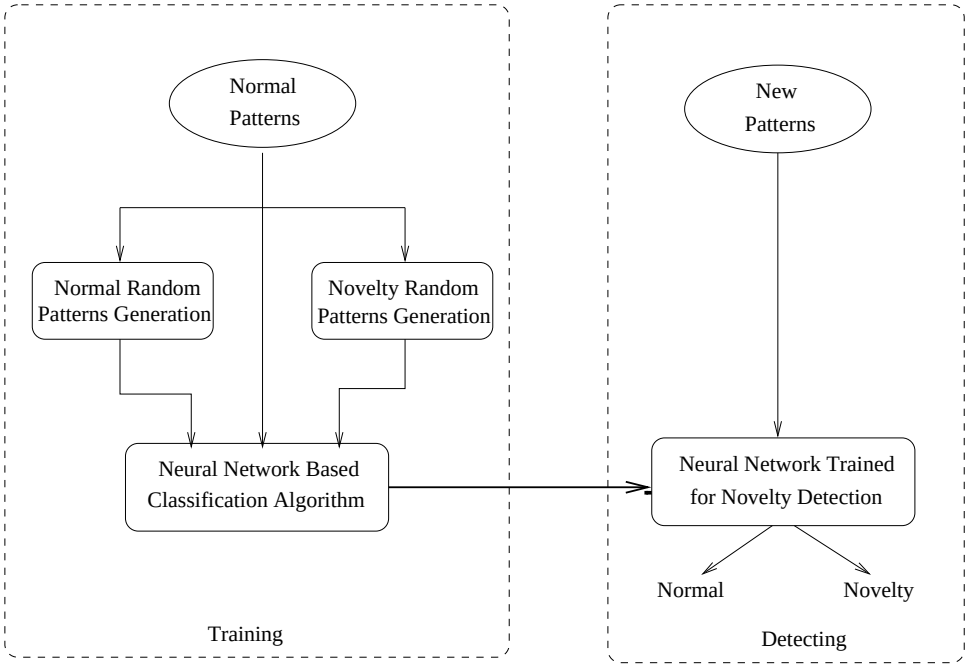


Figure 5.2. The proposed novelty detection method with negative samples. The classifier is trained with an augmented training set formed by the original normal patterns and by normal and novelty random patterns generated based on them.

Any classifier could be used in conjunction with the proposed method, including neural networks classifiers. In this work, experiments were carried out with Radial Basis Functions Networks (RBFs) trained with the DDA algorithm [BD95], RBF-DDA optimized using the first method proposed in chapter 4, MLPs, and machine committees of these classifiers. Nonetheless, other architectures could also be tried, such as support vector machines [Hay98]. The architectures employed here are focused TLFNs (*Time Lagged Feedforward Networks*), in which temporal processing is located only in the network input. That limits the practical use of these architectures to stationary time series [Hay98]. In order to deal with non-stationary time series, it is important to use some pre-processing, such as differencing [Cha89], in order to work with a stationary series. Alternatively one could use *distributed* TLFNs, such as TDNN, FIR neural networks [Hay98] or TDRBF [Ber94], or use a recurrent neural network [Hay98]. In this work the first alternative was adopted, that is, the non-stationary time series are differenced before training and testing the classifier.

To train and test the system, one proceeds as follows:

i) For each normal pattern available, generate n normal random patterns, according to the criterion previously stated;

ii) For each normal pattern available, generate n+1 random novelty patterns, according to the criterion previously stated.

iii) Create an augmented pattern set including the real normal patterns and the normal

and novelty random patterns generated. Divide the resulting augmented pattern set into disjoint training and test sets.

iv) Train a neural network for pattern classification using the augmented training set;

v) Evaluate system performance with using the augmented test set.

It should be noted that an augmented test set is also generate in order to evaluate the performance of the method. This is because the time series used in the experiments reported later present no novelty and therefore novelties have to be simulated by generating augmented test sets in the same way as augmented training sets were generated. The difficulty in finding series with real novelties appears in other works as well [MP03, KLC02, GD02, GDK02, STZ00, Kos00]. These works also tackle this problem by simulating novelties (see chapter 3).

The RBF-DDA classifier does not need a validation set, however, MLPs can need one if early stopping is used as the criterion for stopping training [Hay98]. In this case, part of the training data is used for forming the validation set. Two alternatives for generating validation sets are discussed below.

5.3.1 Generation of Training and Validation Sets

Early stopping is a common method used to avoid overfitting in neural network training (see chapter 2) [Hay98], specially with MLP neural networks. In this technique, data available for training is further divided into training and validation sets. During training, the generalization performance is measured in the validation set; training stops when generalization performance starts to decrease. In time series forecasting it is common to segregate data into training, validation and test sets in time order [YT00, Hay98]. For example, for monthly time series starting in 1996 and ending in 2002, months from 1996 to 2000 would form the training set; validation set would correspond to 2001 months and the test set would be formed from 2002 months.

In order to generate the augmented training and validation sets in time order one proceeds as follows:

i) Divide the original time series into disjoint training, validation and test periods in time order;

ii) For each set, generate normal patterns with window size w length, according to the procedure previously stated;

iii) For each set, for each normal pattern available, generate n normal random patterns, according to the criterion previously stated;

iv) For each set, for each normal pattern available, generate $n + 1$ random novelty patterns, according to the criterion previously stated.

v) For each set, create an augmented pattern set including the real normal patterns and the normal and novelty random patterns generated.

The problem is that division of data in time order can potentially loose important information for training. Thus, a new form of division referred under the name of *distributed division* is proposed here. In distributed division the original time series is divided into training, validation and test sets in time order. However, a number of the random patterns generated from the training set are added to form the augmented validation set and vice versa. In this way, the resulting augmented training and validation sets will have information from all the period available for training. This can result in better classification performance for classifiers that need a validation set for training, as shown in the experiments that follow.

The generation of augmented training and validation sets in distributed division works as follows:

i) Divide the original time series into disjoint training and test periods in time order;

ii) For each set, generate normal patterns with window size w length, according to the procedure previously stated;

iii) For each normal pattern available in the training set, generate n normal random patterns, according to the criterion previously stated. Put a percentage of these patterns in the augmented training set and rest in the augmented validation set;

iv) For each normal pattern available in the training set, generate $n+1$ novelty random patterns, according to the criterion previously stated; Put a percentage of these patterns in the augmented training set and rest in the augmented validation set;

5.3.2 The Classification Algorithms

A number of alternative neural networks classifiers are considered in this work for use in conjunction with the method for time series novelty detection based on negative samples. At first, the use of RBFs trained with the DDA was considered as the classifier [ONM03]. Later, MLPs and committee machines formed either by MLPs only or by MLP and RBF-DDA were also considered as classifiers [ONM04a]. Finally, two of the methods for improving RBF-DDA proposed in chapter 4 are also used in conjunction with the proposed method. All these classifiers have been reviewed in chapter 2, however some remarks are needed here regarding MLPs and committee machines in order to clearly indicate the training methods used.

5.3.2.1 Multi-Layer Perceptrons – MLPs The second kind of classifiers considered in this work are Multi-Layer Perceptron (MLP) neural networks. Four alternative MLP classifiers were used:

- MLP trained with Rprop using time order validation sets;
- MLP trained with Rprop using distributed validation sets;
- MLP trained with RpropMAP using distributed validation sets;

- A committee machine with MLPs trained with RpropMAP.

The two first MLP classifiers are trained with the *resilient backpropagation* (Rprop) [RB93]. Besides being much faster that the original backpropagation, Rprop has only two parameters and its performance is rather insensitive to the parameter's values . In this work the default values for the parameters are used: $\Delta_0 = 0.1$ and $\Delta_{max} = 50.0$ [RB93]. These classifiers are trained with early stopping (see chapter 2). Training stops according to the GL_5 criterion from Proben1 [Pre94], presented in chapter 2.

The third MLP classifier is trained with Rprop with adaptive weight decay (RpropMAP) [Zel98]. This is an extended version of the Rprop algorithm that uses regularization by adaptive weight decay (see chapter 2) [Hay98]. The weighting parameter λ for the weight-decay regularizer is computed automatically within the Bayesian framework, during training [Mac92b]. RpropMAP has two parameters besides those of Rprop: the weighting λ of the weight decay regularizes and the update frequency of the weighting parameter. In the experiments reported below, the default values for these parameters were used. These values are 1 and 50, respectively [Zel98].

RpropMAP does not need a validation set, which is a very important advantage for the novelty detection method. All data available for training are effectively used to adjust the network weights. Training is carried out in two phases. The first phase uses a validation set in order to discover the number of epochs to be used in the second phase. The second phase uses the full training set. In the experiments, the first step divides training data into training and validation sets using distributed validation. Next, network weights and bias are randomly initialized and the network is trained by using RpropMAP and early stopping with the GL_5 criterion. At the end of training, the number of epochs until early stopping, e_{max}, is registered. Finally, the same random weights and bias initialization used in the first training phase are used as the starting values. The network is trained, this time without a validation set, that is, with the full training set. Training stops when the number of epochs reaches e_{max}.

The fourth MLP classifier is a committee machine of MLP classifiers. Experiments with the third classifier are carried out ten times with different random initializations of weights and bias. Next, the mean and standard deviation of the classification error over this executions are obtained. On the other hand, the fourth MLP classifier uses *ensemble mean* to compute the classification error [Hay98]. In this case, for each augmented training set, there will be ten MLPs with two output neurons trained with RpropMAP, as described previously. For each MLP j, the value of output i is given by y_{ij}. Given a pattern p in the test set, the output i of the committee classifier is computed using $\sum_{j=1}^{10} y_{ij}$. Next, pattern p is classified according to the winner-takes-all criterion [Hay98]. This procedure is carried out for each pattern in the test set, in order to compute the classification error in this set.

5.3.2.2 MLP/RBF Committee MLP and RBF networks have different characteristics, mainly due to their different activation functions [Hay98]. The experiments presented below show that these networks have different performance regarding false positive (false alarms) and false negatives (undetected novelty rates). This has motivated the in-

vestigation of the use of an MLP/RBF committee in order to further improve the novelty detection system performance. The committee uses RBF trained with DDA and MLPs trained with RpropMAP.

For each augmented training set, an RBF is trained only once because DDA does not depends on weights initialization. On the other hand, MLPs are trained ten times for each augmented training set. RBF-DDA outputs can be greater than 1. MLPs using sigmoid logistic activation functions produce outputs from 0 to 1. Thus, in order to integrate the information from these classifiers their results must be transformed. The *softmax* transformation is used for this purpose [DHS00]. Firstly, the results of MLPs trained with RpropMAP are combined to form the MLP committee described previously. Next, the softmax transformation

$$g_i = \frac{e^{y_{ij}}}{\sum_{i=1}^{2} e^{y_{ij}}} \tag{5.1}$$

is applied to each committee output i. The softmax transformation is also applied independently to the outputs of the trained RBF-DDA. Finally, the softmax transformed outputs of the MLP committee and of the RBF are combined to obtain an ensemble mean. The winner-takes-all criterion is applied for classification.

5.3.3 Neural Networks Topologies

Two RBF-DDA architectures were considered for used in conjunction with the novelty detection method. Both were tested on the same datasets, in order to compare their performance in this problem. Both architectures have twelve input neurons, corresponding to the window size w. The first architecture, shown in figure 5.3, has only two output neurons and is able to classify a pattern as normal or novelty. The other architecture, presented in figure 5.4 has twenty-four output neurons and can tell not only if a pattern is normal or novelty but also to what month corresponds the first input of the pattern presented to it. In other words, this network has two outputs for each month where one of them indicates a normal pattern whose first value was taken from this month and the other indicates that the pattern had its first value taken from this month and also that it is a novelty.

For MLPs and committee machines, only networks with two outputs were considered. One output is used to indicate normality and the other novelty. The number of inputs of these networks is w and the number of hidden units is optimized by hand.

5.3.4 Pre-processing

The neural networks considered in this chapter are focused TLFNs (*Time Lagged Feedforward Networks*), in which temporal processing is located only in the network input [Hay98]. That limits the practical use of these architectures to stationary time series [Hay98]. In order to use focused TLFNs in conjunction with non-stationary series, it is important to pre-process the time series to make it stationary. In this work the classic technique of differencing is used to obtain stationary versions of the time series [Cha89].

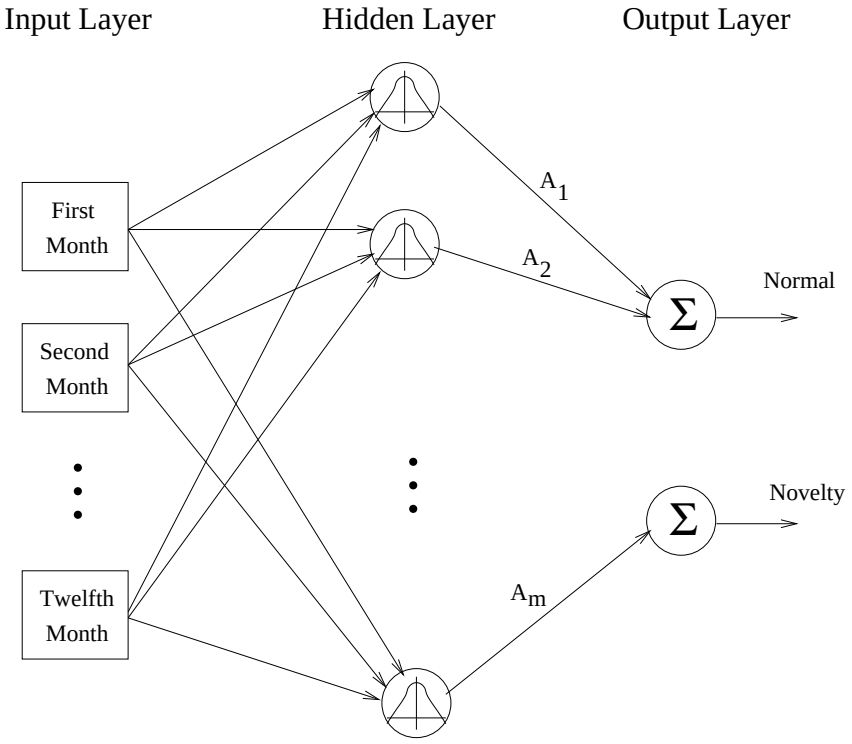


Figure 5.3. RBF network with two output units for classification-based novelty detection.

For each original time series $\{x_1, \ldots, x_N\}$ a differenced time series $\{y_2, \ldots, y_N\}$ is formed by $y_t = x_t - x_{t-1}$. Note that the differenced time series does not have the first data point.

In the experiments presented below both differenced and non-differenced versions of the time series are in fact considered. This is done in order to analyze the impact of time series differencing in novelty detection performance. In either case, each time series has their values normalized between 0 and 1 before training and testing. This is carried out using the expression

$$x_{norm} = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{5.2}$$

where x_{norm} is the normalized value corresponding to the original value x of the series in a given month and x_{min} e x_{max} are, respectively, the minimum and maximum values among the all values available for the series. This normalization is mandatory for using RBF-DDA networks [BD95] and is indicated for using MLP as well [Hay98]. It is important to emphasize that normalization is carried out only after differentiation.

5.3.5 Experimental Results

Experiments using real world data were carried out in order to test the performance of the method on novelty detection. The first set of experiments were carried out using four short time from financial systems. These are non-stationary seasonal time series as shown by their correlograms, presented in appendix B. These series have 84 values

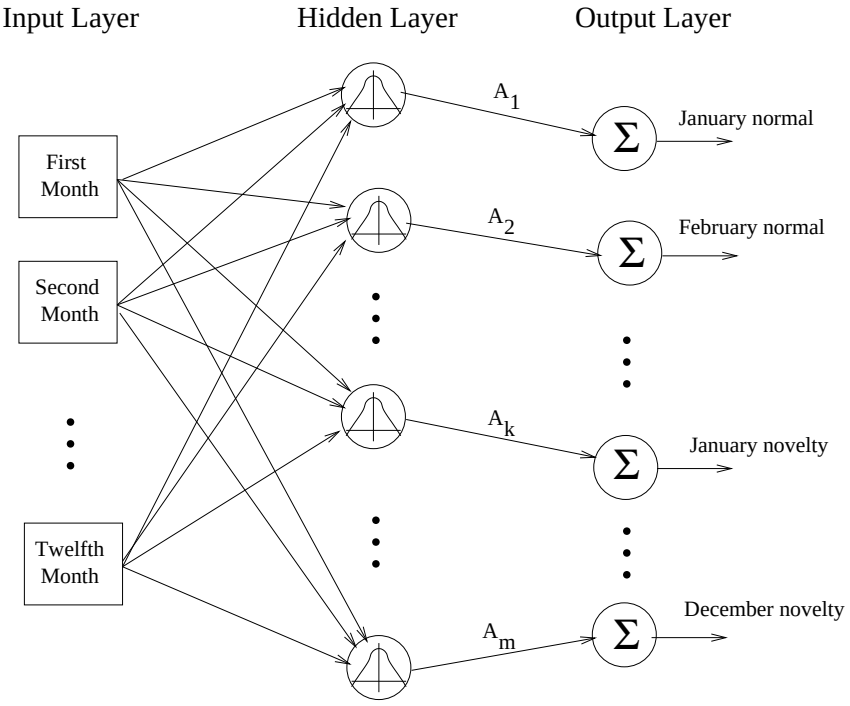
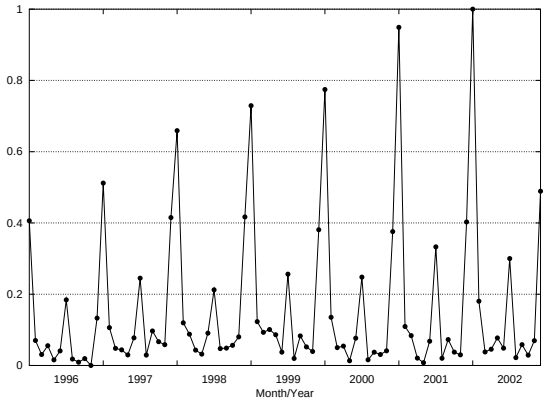


Figure 5.4. RBF network with twenty-four output units for classification-based novelty detection.

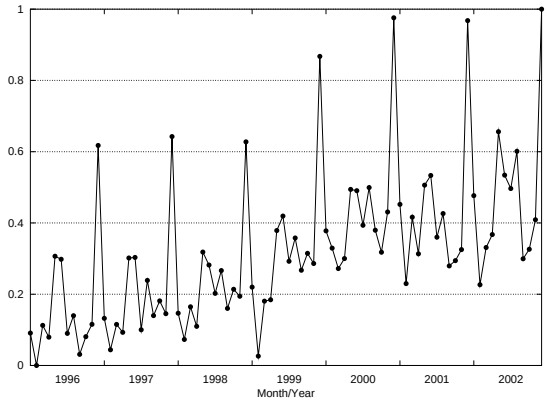
corresponding to the months from January 1996 to December 2002. The first series was extracted from a real Brazilian payroll and was used previously in a study of a payroll auditing system based on neural networks forecasting [OABS03]. The remaining series are sales time series with values in the same period of the first series.

These time series are clearly non-stationary. A visual observation clearly indicates that their mean is variable in time. A technique commonly employed to verify if a time series is stationary is the autocorrelogram [Cha89]. The autocorrelograms of the time series used here are shown in appendix B. They show that these series are seasonal and non-stationary. It is obvious that their period is 12. Therefore, in the first set of experiments, a window size $w = 12$ was used. For the original time series this means that datasets with 73 patterns are generated using the procedure described in section 5.2. For differenced time series, datasets with 72 patterns each having 12 attributes will be generated. For both cases, the last 12 patterns are used as the test set and the remaining patterns as training set.

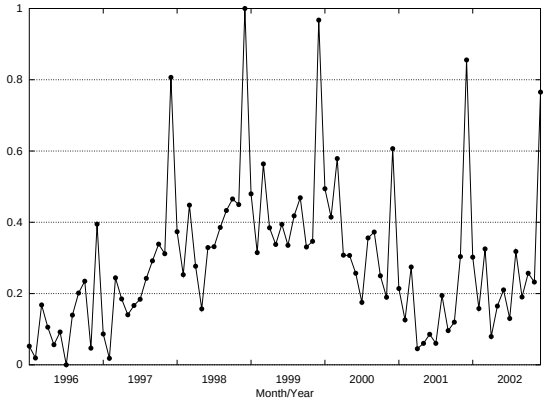
5.3.5.1 Influence of Time Series Differencing and Network Topology The first set of experiments carried out aimed at comparing the two RBF-DDA architectures considered, that is, RBF-DDAs with two and twenty-four outputs. They also aimed at comparing the classification performance when using the original and differenced versions of each time series. Thresholds $p_1 = 0.1$ and $p_2 = 0.5$ were used for defining the envelope (see section 5.2). The generation of random normal and novelty patterns is carried out according to section 5.3. For each original pattern, 9 normal random patterns and 10



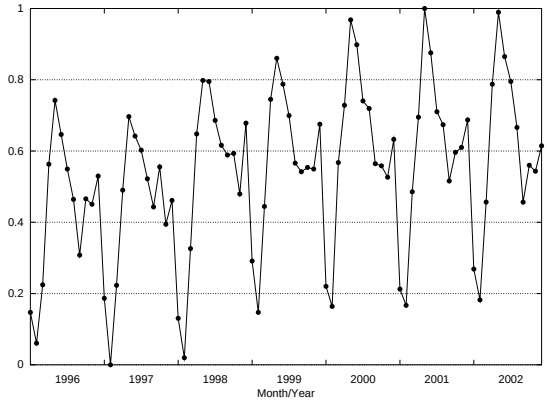
(a) Series 1: earning from a Brazilian payroll.



(b) Series 2: sales from a Brazilian company.



(c) Series 3: USA computer and software store sales.



(d) Series 4: USA hardware stores sales.

Figure 5.5. Time series used in the experiments. Each series has 84 values corresponding to months from January 1996 to December 2002. Values are normalized between 0 and 1. Series 3 and 4 are available at the URL http://www.census.gov/mrts/www/mrts.html.

random novelty patterns are added to the training and test sets to form augmented sets. With this, the training and test sets increases by a factor of 20, having, respectively, 1220 and 240 patterns for original time series and 1200 and 240 patterns for differenced time series. For each time series, RBF-DDA networks are trained with ten different versions of the augmented training set generated from different seeds, to take into account the variability of the random patterns added to form them. Next, the mean and standard deviation of performance measures across these executions is taken.

Results for RBF-DDAs with two outputs are shown in tables 5.1 and 5.2. Tables 5.3 and 5.4 present the results obtained with twenty-four outputs networks. In all cases, RBF-DDAs with default parameters, $\theta^+ = 0.4$ and $\theta^- = 0.1$, were used.

Tables 5.1 and 5.3 present results obtained after training the networks. They contain the mean number of epochs used in training, the mean number of hidden units in the

resulting network and the network performance on its training set, i.e., the classification error, the false alarm rate and the undetected novelty rate. A false alarms happens when the network classifies a normal pattern as novelty. An undetected novelty happens when a novelty pattern is misclassified. In these table, the mean and standard deviation of these results obtained in ten executions with different augmented training sets are presented. Each augmented training set had its own seed for random patterns generation.

Time series	Epochs	Hidden Units	Class. error		False alarm rate		Undetec. novelty rate	
			mean	s.dev	mean	s.dev	mean	s.dev
Series 1 (original)	5	569.2	1.80%	0.24%	0.00%	0.00%	1.80%	0.24%
Series 1 (diff.)	4	623.2	9.23%	2.24%	0.00%	0.00%	9.23%	2.24%
Series 2 (original)	4	646.1	1.28%	0.50%	0.00%	0.00%	1.28%	0.50%
Series 2 (diff.)	4	621.7	5.65%	1.25%	0.00%	0.00%	5.65%	1.25%
Series 3 (original)	4	656.1	0.93%	0.17%	0.00%	0.00%	0.93%	0.17%
Series 3 (diff.)	4	644.3	4.31%	0.38%	0.00%	0.00%	4.31%	0.38%
Series 4 (original)	4	651.5	3.98%	1.25%	0.00%	0.00%	3.98%	1.25%
Series 4 (diff.)	4	644.4	3.87%	0.62%	0.00%	0.00%	3.87%	0.62%

Table 5.1. Trained RBF-DDA networks with two outputs and its performances on training sets.

Time series	Classification error		False alarm rate		Undetected novelty rate	
	mean	s.dev	mean	s.dev	mean	s.dev
Series 1 (original)	32.75%	3.30%	26.13%	3.00%	6.63%	1.31%
Series 1 (diff.)	11.00%	2.86%	0.00%	0.00%	11.00%	2.86%
Series 2 (original)	12.75%	4.20%	0.00%	0.00%	12.75%	4.20%
Series 2 (diff.)	6.50%	1.86%	0.00%	0.00%	6.50%	1.86%
Series 3 (original)	37.88%	6.09%	35.00%	5.97%	2.88%	0.91%
Series 3 (diff.)	8.79%	1.93%	0.00%	0.00%	8.79%	1.93%
Series 4 (original)	11.13%	3.61%	0.00%	0.00%	11.13%	3.61%
Series 4 (diff.)	7.13%	1.32%	0.00%	0.00%	7.13%	1.32%

Table 5.2. Performance of the RBF-DDA networks with two outputs on test sets.

Tables 5.2 and 5.4 present classification performance on test sets after training. Besides classification error, these tables also present the false alarm and undetected novelty rates. Results presented in tables 5.2 and 5.4 show that differencing the time series makes a great improvement in the classification performance. These results also show that for series 1 the network with twenty-four outputs has obtained the best classification performance while for series 2, 3 and 4, networks with two outputs have achieved the best performance.

Tables 5.1 and 5.3 show that there is no false alarm on training sets, that is, all classification errors happen on novelty patterns. This also happens for test sets for all differenced time series and for half original series, as shown in tables 5.2 and 5.4. Recall that normal random patterns have all attribute values in a limited area of the hyperspace within the envelope, with values between -10% and 10% of a real pattern values. On the other hand, novelty random patterns have their attributes values deviating from a real pattern from -50% and -10% or from 10% to 50%. RBFs activation functions are

Time series	Epochs	Hidden Units	Class. error		False alarm rate		Undetec. novelty rate	
			mean	s.dev	mean	s.dev	mean	s.dev
Series 1 (original)	5	574.6	1.81%	0.28%	0.00%	0.00%	1.81%	0.28%
Series 1 (diff.)	4	632.6	1.86%	0.56%	0.00%	0.00%	1.86%	0.56%
Series 2 (original)	4	647.7	2.30%	0.52%	0.00%	0.00%	2.30%	0.52%
Series 2 (diff.)	4	622.6	6.88%	1.38%	0.00%	0.00%	6.88%	1.38%
Series 3 (original)	4	657.8	1.94%	0.35%	0.00%	0.00%	1.94%	0.35%
Series 3 (diff.)	4	645	6.53%	0.53%	0.00%	0.00%	6.53%	0.53%
Series 4 (original)	4	652.8	4.97%	1.25%	0.00%	0.00%	4.97%	1.25%
Series 4 (diff.)	4	645.6	5.04%	0.66%	0.00%	0.00%	5.04%	0.66%

Table 5.3. Trained RBF-DDA networks with twenty-four outputs and its performances on training sets.

Time series	Classification error		False alarm rate		Undetected novelty rate	
	mean	s.dev	mean	s.dev	mean	s.dev
Series 1 (original)	33.00%	3.41%	26.13%	3.00%	6.88%	1.25%
Series 1 (diff.)	7.92%	2.43%	0.00%	0.00%	7.92%	2.43%
Series 2 (original)	25.67%	4.29%	0.00%	0.00%	25.67%	4.29%
Series 2 (diff.)	7.71%	2.12%	0.00%	0.00%	7.71%	2.12%
Series 3 (original)	38.13%	4.15%	25.08%	5.02%	13.04%	2.63%
Series 3 (diff.)	13.83%	3.55%	0.00%	0.00%	13.83%	3.55%
Series 4 (original)	21.25%	4.01%	0.00%	0.00%	21.25%	4.01%
Series 4 (diff.)	9.04%	1.98%	0.00%	0.00%	9.04%	1.98%

Table 5.4. Performance of the RBF-DDA networks with twenty-four outputs on test sets.

localized and so they can cover well the region of normal patterns and classify correctly all these patterns. However, novelty patterns can have attribute values on two different regions (see figure 5.1) and this leads to misclassification of some of these patterns.

5.3.5.2 Influence of Augmented Training Set Size Experiments were also carried out in order to analyze the classification performance of the system as function of the augmented training set size. Experiments were carried out only with differenced time series, because the previous experiments have indicated that a much better classification performance is achieved using differentiation. For each series, an augmented test set with much more patterns was generated. Augmented test sets with 240 patterns were used before. This time augmented test sets with 2400 patterns are considered. To generate these larger augmented test sets, for each original pattern in the test set 99 normal random patterns and 100 random novelties are included in the augmented test set, together with the original normal patterns. For each series networks were trained using augmented training sets with 240, 600 and 1200 patterns, with ten different versions of the respective augmented training sets. Each augmented training set was generated with a different seed for the random number generator and the larger augmented test set had also a seed of its own. For time series 1, the RBF-DDA network with twenty-four outputs was used whereas the RBF-DDA with two outputs was used for series 2, 3 and 4. This was done because these architectures have performed better for the respective time series.

Table 5.5 shows the classification error on the test sets as a function of training set

Training	Class. error series 1		Class. error series 2		Class. error series 3		Class. error series 4	
set size	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev
240	26.70%	2.50%	23.14%	3.28%	29.35%	2.76%	33.97%	3.30%
600	20.58%	3.30%	17.04%	2.81%	17.12%	2.44%	20.81%	2.28%
1200	17.21%	3.46%	12.52%	1.58%	11.20%	1.27%	14.39%	1.39%

Table 5.5. Classification error on test sets as function of training set size. Results consider differentiated time series and used twenty-four output networks for series 1 and two output networks for series 2,3 and 4. The augmented test sets have 2400 patterns.

size. It is clear from these results that the size of the training set augmented by random patterns is very important for classification performance. Thus, in practice, one should use augmented training set as large as possible. This is also a limitation of novelty detection techniques for classification problems based on negative samples. The problem is that the random negative samples introducing during training may not adequately cover all the novelty space, as discussed in chapter 3 [Vas95, MS03b].

5.3.5.3 Influence of Classifiers on Performance Additional experiments were carried out in order to compare the performance of a number of different classifiers described in section 5.3.2 when used in conjunction with the novelty detection method [ONM04a]. In this subsection we consider only classifiers with two outputs, one to indicate normality and the other, novelty. The number of inputs of each network was also $w = 12$ and corresponds to the window size used. Recall that the number of hidden units in the RBF classifiers is obtained automatically during training by the DDA algorithm [BD95]. In the experiments, MLPs with one hidden layer were used. The number of units in the hidden layer has a great impact on classifier performance. For MLPs, experiments were carried out with 2, 6, 12, 18, 24, 36 and 48 units in the hidden layer. Only the results corresponding to the topology that has performed better are presented.

In these experiments, each classifier is also trained with ten different versions of the augmented training set generated from different seeds, to take into account the variability of the random patterns added to form them. Each time series is first differenced and then normalized before its use by the novelty detection method. DDA algorithm for RBF training is constructive and its result does not depend on the initial values of the networks weights and therefore for each augmented training set, experiments are executed only once. On the other hand, MLPs performance depends on weights and bias initializations thus, in this case, for each augmented training set, the MLP network is trained and tested ten times, to take into account the weight initialization influence on results.

The generation of augmented training and test sets is carried out as in section 5.3.5.1. For each original pattern, 9 normal random patterns and 10 random novelty patterns are generated to form the augmented training set together with the original normal patterns. With this, training and test sets increases by a factor of 20, having, respectively, 1200 and 240 patterns for differenced time series. For classifiers that need a validation set, training data is further divided into training and validation sets using either time order or distributed validation, according to section 5.3.1. In both cases, the training set will have 80% of the patterns (960) and the validation set will have 20% of them (240 patterns).

Classifier	Epochs	Hidden Units	Class. error		False alarm rate		Undetec. novelty rate	
			mean	s.dev	mean	s.dev	mean	s.dev
Time Series 1								
MLP (1)	224.10	6	30.08%	3.53%	21.52%	2.75%	8.55%	2.09%
MLP (2)	584.35	24	10.10%	1.68%	8.60%	2.08%	1.50%	1.44%
MLP (3)	3552.65	48	1.37%	0.56%	0.91%	0.44%	0.46%	0.24%
MLP (4)	3552.65	48	0.04%	0.13%	0.00%	0.00%	0.04%	0.13%
MLP/RBF	3552.65/4	48/623.2	3.25%	1.33%	0.00%	0.00%	3.25%	1.33%
RBF-DDA	4	623.2	11.00%	2.86%	0.00%	0.00%	11.00%	2.86%
Time Series 2								
MLP (1)	150.10	12	33.17%	1.57%	22.86%	1.96%	10.31%	1.93%
MLP (2)	988.75	12	22.41%	1.56%	19.80%	2.01%	2.61%	0.97%
MLP (3)	7033.55	36	7.09%	2.26%	6.44%	2.21%	0.65%	0.34%
MLP (4)	7033.55	36	3.58%	2.42%	3.37%	2.34%	0.21%	0.41%
MLP/RBF	7033.55/4	36/621.7	3.08%	1.27%	0.00%	0.00%	3.08%	1.27%
RBF-DDA	4	621.7	6.50%	1.86%	0.00%	0.00%	6.50%	1.86%
Time Series 3								
MLP (1)	450.55	12	29.90%	1.41%	25.24%	1.95%	4.66%	0.90%
MLP (2)	710.95	24	29.26%	3.00%	26.65%	3.28%	2.61%	0.92%
MLP (3)	7869.75	48	8.35%	2.88%	6.78%	2.74%	1.57%	0.38%
MLP (4)	7869.75	48	4.54%	3.29%	3.71%	2.90%	0.83%	0.52%
MLP/RBF	7869.75/4	48/656.1	3.04%	0.92%	0.00%	0.00%	3.04%	0.92%
RBF-DDA	4	656.1	8.79%	1.93%	0.00%	0.00%	8.79%	1.93%
Time Series 4								
MLP (1)	205.00	48	20.36%	3.22%	13.43%	1.86%	6.93%	2.53%
MLP (2)	640.45	48	11.46%	2.71%	8.48%	2.87%	2.98%	1.03%
MLP (3)	6948.30	36	2.09%	0.58%	1.02%	0.47%	1.07%	0.45%
MLP (4)	6948.30	36	0.79%	0.69%	0.21%	0.35%	0.58%	0.53%
MLP/RBF	6948.30/4	36/644.4	2.75%	1.09%	0.00%	0.00%	2.75%	1.09%
RBF-DDA	4	644.4	7.13%	1.32%	0.00%	0.00%	7.13%	1.32%

Table 5.6. Comparing performance of the negative samples novelty detection method on test sets for each time series with different classifiers.

Table 5.6 presents results obtained after training the networks for series 1, 2, 3, and 4 from figure 5.5. It contains the mean number of epochs used in training, the mean number of hidden units in the resulting network and the network performance on its test set, i.e., the classification error, the false alarm rate and the undetected novelty rate.

In table 5.6 MLP(1) means MLP trained with Rprop using time order validation sets; MLP(2) means MLP trained with Rprop using our distributed validation sets generation; MLP(3) means MLP trained with RpropMAP; MLP(4) means MLP committee with ensemble mean trained with RpropMAP; MLP/RBF is the classifier built by combining MLP and RBF results. Finally, the previous results obtained by RBF-DDA are presented for comparison [ONM03].

Results show that the use of the proposed distributed validation set approach presented in section 5.3.1 improves MLP performance on test sets for all series considered. It can also be seen that using all data available for training to form the training set and training with RpropMAP greatly improves performance. Results also show that the committee of MLPs and the MLP/RBF committee further improve classification performance

on test sets, for all time series considered. These committee machines outperform both MLP and RBF-DDA. RBF-DDA obtained 8.36% mean classification error across the four series, while the MLP committee and the MLP/RBF committee obtained, respectively, 2.24% and 3.03%.

Table 5.7 summarizes the results of table 5.6 in order to better compare those results. The mean classification error on test set obtained by each classifier across the four series is presented.

Classifier	Test set class. error
MLP (1) - Rprop/time order validation	28.38%
MLP (2) - Rprop/distributed order validation	18.31%
MLP (3) - RpropMAP/distributed order validation	4.73%
MLP (4) - RpropMAP committee machines	2.24%
MLP/RBF committee machines	3.03%
RBF-DDA	8.36%

Table 5.7. Mean classification error on test sets across the four series for different classifiers.

The MLP and MLP/RBF committees have produced comparable results for the classification error. MLP committee outperformed MLP/RBF committee in two series and produced worse results for the remaining two series. However, these classifiers produced very different results regarding false alarm and undetected novelty rates. RBF-DDA and the MLP/RBF committee have produced 0% false alarm on all experiments. On the other hand, pure MLP classifiers tend to produce undetected novelty rates greater than false alarm rates. This is due to the different nature of MLPs and RBFs. MLPs build global input-output mappings while RBFs build local mappings as discussed in chapter 2 [Hay98].

5.3.5.4 Improving RBF-DDA Results Through Parameters Adjustment Finally, in this section the improved RBF-DDA introduced in subsection 4.2.1 of this thesis is applied in conjunction with the novelty detection method based on negative samples. Recall that this method has been applied to six benchmark classification datasets and has considerably improved RBF-DDA generalization performance in all of them.

The same four series used in the previous experiments and shown in figure 5.5 are used now. Each time series is first differentiated and then normalized as before. Only networks with two outputs are considered. The number of inputs in each network is also $w = 12$. The augmented pattern sets used are created as in section 5.3.5.3 and therefore have the same number of patterns as in that section. The validation sets used here are created in time order (see section 5.3.1).

Figures 5.6, 5.7, 5.8 and 5.9 show the classification error on test and validation sets as function of the parameter θ^- of the DDA algorithm [BD95], for series 1, 2, 3, and 4, respectively. It can be seen that performance on the validation sets can, indeed, be used to select the value of θ^- that optimizes or nearly optimizes RBF-DDA performance on the respective test set, as was the case for the datasets of chapter 4. In table 5.8 the optimal value of $theta^-$ predicted by the validation error (θ^-_{opt}) is compared with the θ^-

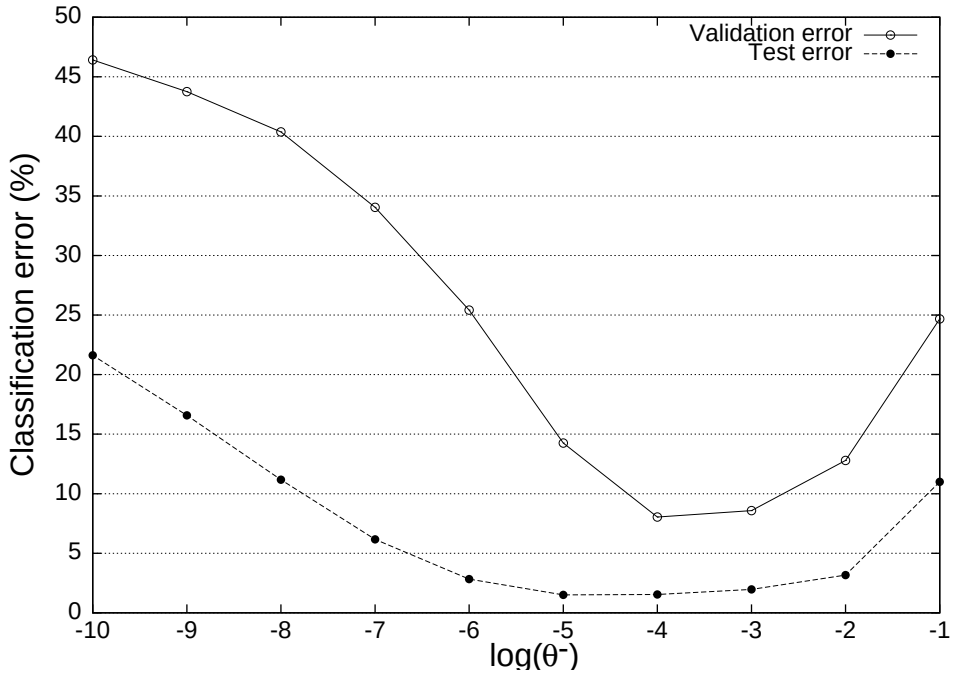


Figure 5.6. Influence of θ^- on classification performance on test set for series 1. θ^+ is fixed on 0.4.

that really minimizes test error (θ^-_{min}) for each time series. It can be seen that the values of θ^-_{opt} and θ^-_{min} are the same for series 2 and 3. However, they are slightly different for series 1 and 4. In these cases, θ^-_{opt} leads to a classification error on test set only near the minimum. Even so, the use of this method for improving RBF-DDA networks regarding generalization considerably improves RBF-DDA performance when compared to DDA trained with default parameters. This is shown by the results presented in table 5.9.

Time series	θ^-_{opt} Val.	θ^-_{min} Test	Test set class. error (with θ^-_{opt} Val.)	Test set class. error (with θ^-_{min} Test)
Time series 1	10^{-4}	10^{-5}	1.54%	1.50%
Time series 2	10^{-4}	10^{-4}	0.63%	0.63%
Time series 3	10^{-3}	10^{-3}	3.92%	3.92%
Time series 4	10^{-3}	10^{-5}	1.04%	0.50%

Table 5.8. Comparing optimal results on test set with results predicted by the RBF-DDA with θ^- selection method.

Finally, table 5.10 compares the performance of all classifiers considered in this chapter for use in conjunction with the proposed novelty detection method. The table reproduces the results from table 5.7 and adds the mean classification error across the four series obtained by the improved RBF-DDA an presented in detail in table 5.9. The results show that the improved RBF-DDA proposed in this work outperforms, in the mean, all the other classifiers considered here in task.

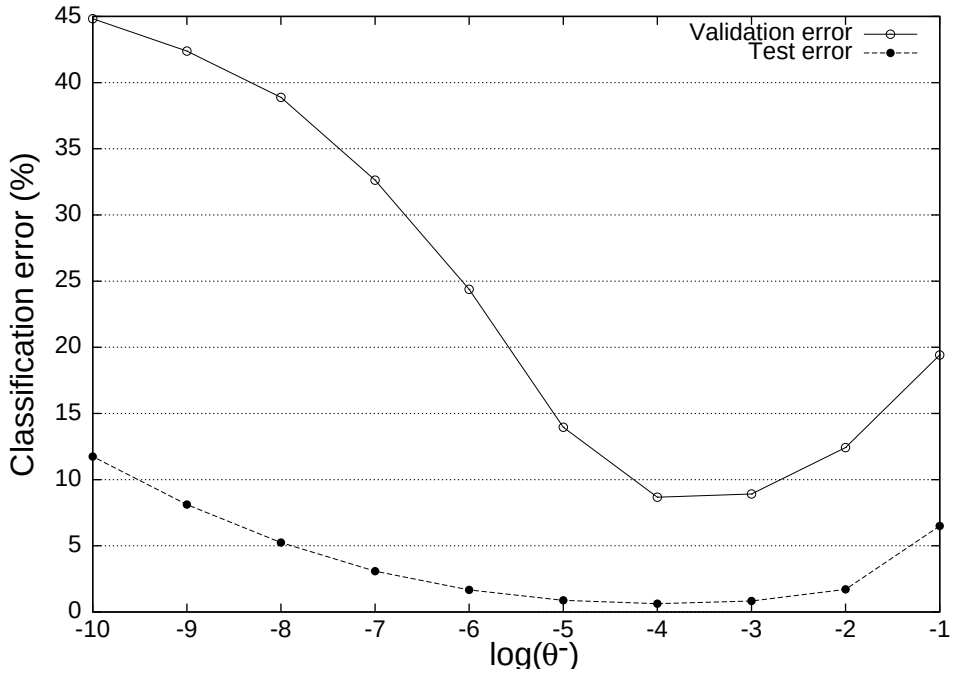


Figure 5.7. Influence of θ^- on classification performance on test set for series 2. θ^+ is fixed on 0.4.

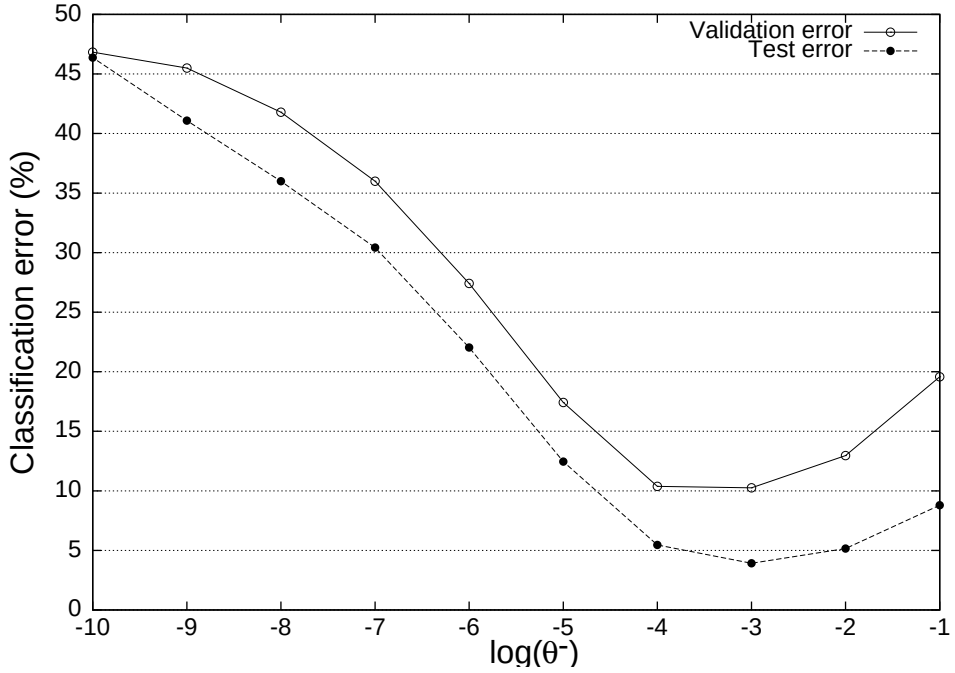


Figure 5.8. Influence of θ^- on classification performance on test set for series 3. θ^+ is fixed on 0.4.

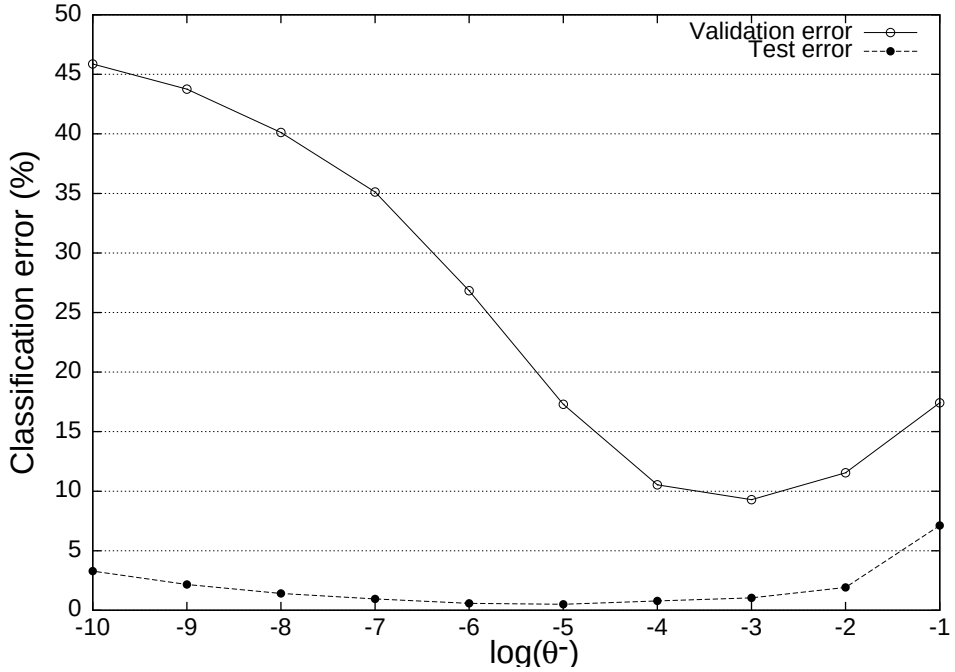


Figure 5.9. Influence of θ^- on classification performance on test set for series 4. θ^+ is fixed on 0.4.

RBF-DDA Method	θ^-	**Epochs**	**Hidden units**	**Class. error**	
				mean	**s.dev**
Time series 1					
Default parameters	0.1	4	623.2	11.00%	2.86%
Selected parameters	10^{-4}	4	728.2	1.54%	0.74%
Time series 2					
Default parameters	0.1	4	621.7	6.50%	1.86%
Selected parameters	10^{-4}	4	724.1	0.63%	0.59%
Time series 3					
Default parameters	0.1	4	644.3	8.79%	1.93%
Selected parameters	10^{-3}	3.6	671.4	3.92%	0.97%
Time series 4					
Default parameters	0.1	4	644.4	7.13%	1.32%
Selected parameters	10^{-3}	3.7	672.5	1.04%	0.53%

Table 5.9. Comparing RBF-DDA with default and selected parameters performance on test sets.

Classifier	Test set class. error
MLP (1) - Rprop/time order validation	28.38%
MLP (2) - Rprop/distributed order validation	18.31%
MLP (3) - RpropMAP/distributed order validation	4.73%
MLP (4) - RpropMAP committee machines	2.24%
MLP/RBF committee machines	3.03%
RBF-DDA	8.36%
RBF-DDA with θ^- selection	**1.78%**

Table 5.10. Comparing RBF-DDA with θ^- selection with other classifiers.

	Test set error		
Time series	RBF-DDA with θ^- selection	MLP committee	MLP/RBF committee
Time series 1	1.54%	0.04%	3.25%
Time series 2	0.63%	3.58%	3.08%
Time series 3	3.92%	4.54%	3.04%
Time series 4	1.04%	0.79%	2.75%

Table 5.11. Comparing RBF-DDA with θ^- selection and machine committee classifiers on each series.

A comparison between this classifier and the committee classifiers per series is provided in table 5.11. It can be seen that RBF-DDA with θ^- selection is the best classifier on series 2, the MLP committee is the best on series 1 and 4 and the MLP/RBF committee is the best on series 3.

5.3.5.5 Influence of Windows Size Additional experiments were carried out in order to evaluate the influence of the windows size w on the performance of the method. Two novel series were used for this purpose. The first series was obtained from a time series library[1] and is depicted on figure 5.10. This series has 159 data points and was also used in chapter 6. The second series is the well-known Mackey-Glass chaotic time series [GDK02], with 1200 data points, depicted in figure 5.11. The series used in the previous experiments were all seasonal and non-stationary [ONM03, ONM04a]. The two series used in this section were selected to show that the novelty detection method also works in different kinds of series. The autocorrelogram of these series are presented and discussed in appendix B. Again, differentiation and normalization are carried out before applying the novelty detection method in each time series.

In the experiments reported below the normal and novelty regions were also defined using thresholds $p_1 = 0.1$ and $p_2 = 0.5$. The augmented training, validation and test sets were also generated as in section 5.3.5.3, that is, each augmented dataset was formed by adding 9 normal patterns and 10 random novelty patterns together with the original normal patterns. For the *empper* series, the last 12 data points were used as test data whereas for the Mackey-Glass series the last 192 points were used as test data.

[1] Available at http://www-personal.buseco.monash.edu.au/~hyndman/TSDL.

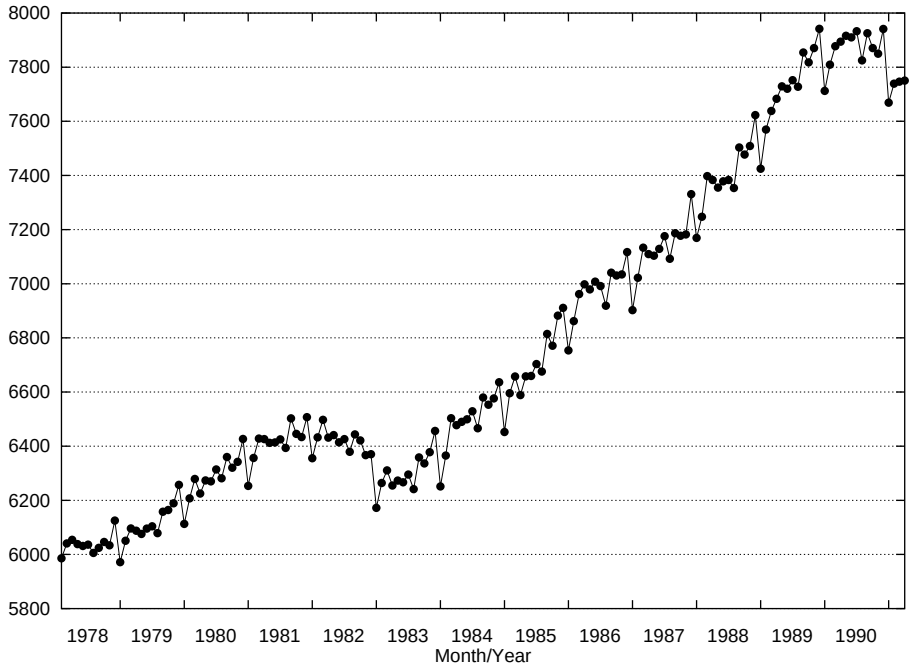


Figure 5.10. Empper Time Series: monthly number of employed persons in Australia. Feb 1978 – Apr 1991.

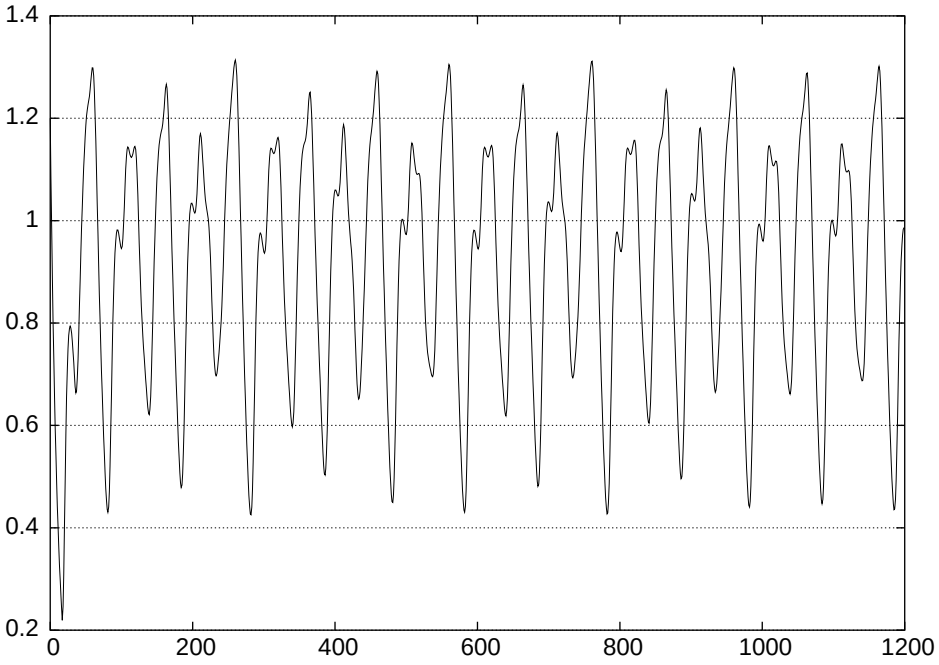


Figure 5.11. Chaotic Mackey-Glass Time Series.

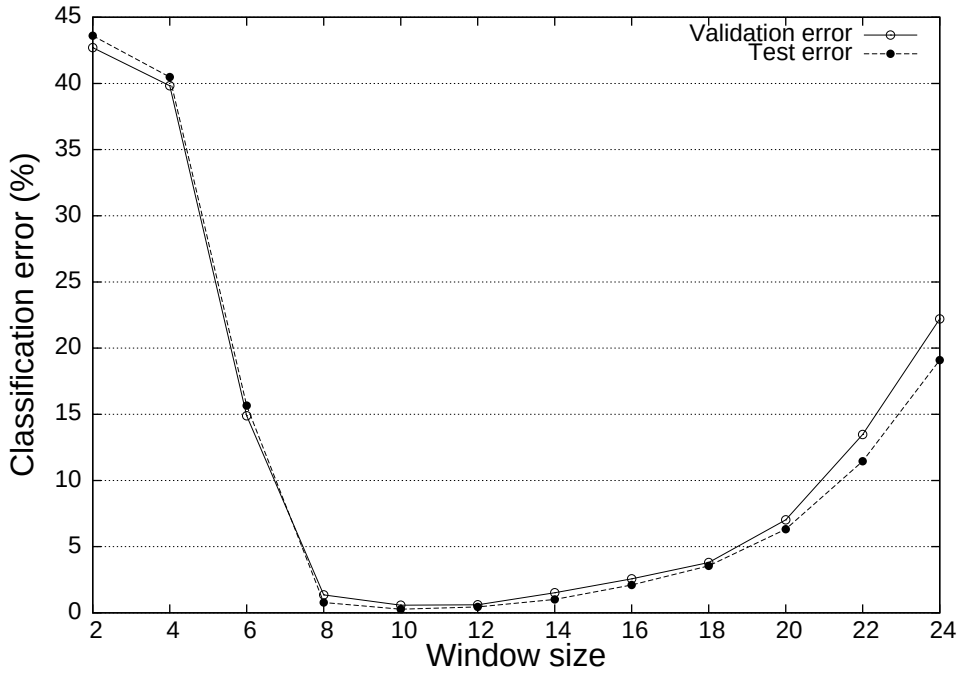


Figure 5.12. Results for Mackey-Glass series on test set with RBF-DDA.

The first set of experiments were carried out using the chaotic Mackey-Glass time series. The results depicted in figure 5.12 were obtained by using RBF-DDA default parameters and varying the window size. The chart shows that the validation set results can also be used to select the optimum window size. It also shows that performance greatly depends on window size. The optimum window size was $w = 10$, which has produced 0.27% mean classification error on test set.

A second set of experiments were carried out using the *empper* time series. Figure 5.13 compares classification performance on test set using RBF-DDA with default parameters with RBF-DDA with parameters optimized using the first method proposed in chapter 4, that is, the method based on θ^- optimization. Results show that the proposed method improves performance for all window sizes. It can also be observed that the window size which optimizes performance is different for the two methods. For default parameters $w = 8$, whereas for optimized parameters $w = 15$. Notice that the selection of the the windows size and the use of the optimized RBF-DDA had great influence on the novelty detection performance.

5.3.5.6 Simulations Using RBF-DDA-SP This subsection reports experiments carried out using the RBF-DDA-SP training method introduced in subsection 4.2.4 of this thesis [OMM05a]. Recall that RBF-DDA-SP is intended to improve RBF-DDA performance without increasing the size of the networks. The method combines selective pruning with θ^- selection. The simulations of this subsection were carried out using time series 1 to 4 (depicted in figure 5.5) and the *empper* time series (figure 5.10). The division of the time series into training and test sets and the generation of augmented training

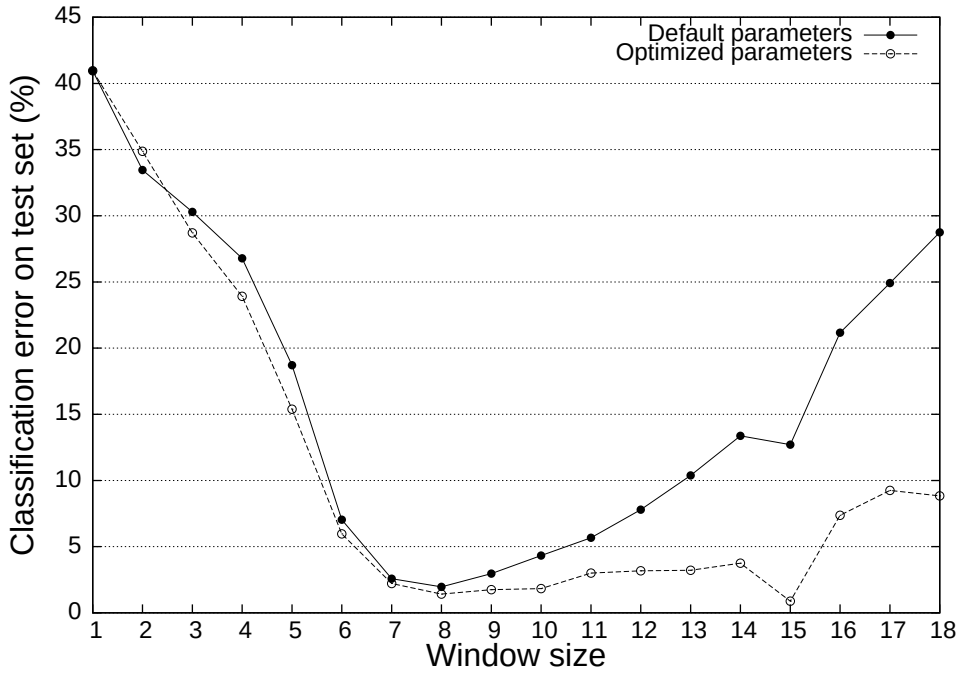


Figure 5.13. Results for empper series on test set with RBF-DDA.

and test sets were the same used in the experiments of subsections 5.3.5.1 and 5.3.5.5. For time series 1 to 4, the window size was $w = 12$ whereas for *empper* the window size was $w = 15$.

Table 5.12 compares performance of RBF-DDA-SP with 10%, 15% and 20% pruning percentages to both RBF-DDA trained with default parameters and RBF-DDA with θ^- selection for time series 1 to 4. This table shows both the classification errors on test sets (mean and standard deviation) and the mean number of hidden units for each series and each classifier considered. RBF-DDA with θ^- selection and RBF-DDA-SP simulations were carried out using $\theta^- = 10^{-4}$ for series 1 and 2 and $\theta^- = 10^{-3}$ (as in table 5.9).

The results of table 5.12 show that RBF-DDA-SP outperforms RBF-DDA trained with default parameters for the four time series considered. Moreover, it produces networks whose sizes are comparable to those of RBF-DDA trained with default parameters.

Simulations for the *empper* time series used a window size $w = 15$ and $\theta^- = 10^{-4}$ for RBF-DDA with θ^- and for RBF-DDA-SP. We considered RBF-DDA-SP with 30%, 40% and 50% of selective pruning. The results of these simulations are shown in table 5.13. These results also show that RBF-DDA-SP is able to considerably improve performance with regard to the default RBF-DDA and, simultaneously, producing much smaller networks.

5.3.5.7 Simulations Using Support Vector Machines This subsection compares performance of SVM with that of other classifiers already considered in this chapter in the problem of novelty detection in time series via classification. SVMs, a recent class of powerful classifiers, have been reviewed in section 2.3 of this thesis. The time series used

Method	series 1	series 2	series 3	series 4
RBF-DDA (default)	11.00% [623.2] (2.86%)	6.50% [621.7] (1.86%)	8.79% [644.3] (1.93%)	7.13% [644.4] (1.32%)
RBF-DDA (θ^- sel.)	1.54% [728.2] (0.74%)	0.63% [724.1] (0.59%)	3.92% [671.4] (0.97%)	1.04% [672.5] (0.53%)
RBF-DDA-SP (10%, θ^- sel.)	2.39% [664.80] (0.62%)	1.38% [660.40] (0.59%)	4.48% [610.90] (0.98%)	1.71% [611.50] (0.55%)
RBF-DDA-SP (15%, θ^- sel.)	2.94% [633.20] (0.71%)	1.82% [628.70] (0.52%)	4.78% [580.50] (1.01%)	2.14% [581.30] (0.64%)
RBF-DDA-SP (20%, θ^- sel.)	3.61% [601.60] (0.66%)	2.28% [597.00] (0.46%)	5.13% [550.20] (1.00%)	2.62% [550.70] (0.64%)

Table 5.12. Classification errors on test sets and number of hidden RBFs for series 1 to 4

Method	*empper time series*
RBF-DDA (default)	20.59% [1259.40] (10.63%)
RBF-DDA (θ^- sel.)	0.00% [1372.20] (0.00%)
RBF-DDA-SP (30%, θ^- sel.)	0.28% [975.90] (0.32%)
RBF-DDA-SP (40%, θ^- sel.)	0.42% [843.90] (0.43%)
RBF-DDA-SP (50%, θ^- sel.)	0.59% [711.50] (0.52%)

Table 5.13. Classification errors on test sets and number of hidden RBFs for the *empper* time series

Classifier	Epochs	Hidden Units	Classification error	
			mean	s.dev
MLP committee	3552.65	48	0.04%	0.13%
RBF-DDA (default)	4	623.2	11.00%	2.86%
RBF-DDA (θ^- sel.)	4	728.2	1.54%	0.74%
SVM	1642.50	608.2	**0.00%**	**0.00%**

Table 5.14. Performance of the novelty detection approach on test set for time series 1. Comparing results from different classifiers.

in the experiments of this subsection were time series 1 to 4 (depicted in figure 5.5). The division of the time series into training and test sets, the generation of augmented training and test sets and the window size ($w = 12$) were the same used in the experiments of subsection 5.3.5.1.

The simulations using SVMs reported here were carried out using the LIBSVM, a freely available implementation of SVMs [CL01, HCL04]. For each time series used in the experiments, ten different augmented training and test sets are generated in order to take into account the variability of the random samples generated to represent normal and novelty patterns (as in subsection 5.3.5.1). Each version of the augmented training sets is used to train an SVM independent of the others. For each augmented training set, the selection of parameters C and γ is carried out as outlined in section 2.3.

Tables 5.14, 5.15, 5.16 and 5.17 compare SVM with other classifiers already considered in this work regarding number of training epochs, number of hidden units (complexity) and generalization performance. The number of training epochs and the number of hidden units for each classifier correspond to means of ten executions of the novelty detection method. Each table shows both the mean and standard deviation of the classification error on test sets over ten runs of the simulations (using different augmented training and test sets) for each classifier.

The results of tables 5.14, 5.15, 5.16 and 5.17 show that SVM is the best classifier regarding generalization performance for time series 1 and 4. On the other hand, the RBF-DDA with θ^- selection classifier proposed in this thesis achieved the best classification performance in time series 2 and 3. The mean classification errors across the four time series obtained by each classifier is presented in table 5.18. This table shows that RBF-DDA with θ^- selection obtains the best mean results when compared to all the other classifiers considered in this thesis.

SVM build classifiers with smaller size than RBF-DDA with θ^- selection for the first three series, as shown in tables 5.14, 5.15 and 5.16. On the other hand, the latter classifier builds smaller networks than the former for time series 4 (see table 5.17).

The results of these tables also show that the time complexity of RBF-DDA with θ^- selection is much smaller than that of SVMs. The number of training epochs for SVMs in much greater than for RBF-DDA with θ^- selection for all time series considered. The DDA training algorithm is much simpler than that of SVM, hence, each training epoch of RBF-DDA training is faster than each training epoch of SVM. Furthermore, RBF-DDA with θ^- selection needs to select only one parameter, whereas when using SVMs one

Classifier	Epochs	Hidden Units	Classification error	
			mean	s.dev
MLP committee	7033.55	36	3.58%	2.42%
RBF-DDA (default)	4	621.7	6.50%	1.86%
RBF-DDA (θ^- sel.)	4	724.1	**0.63%**	**0.59%**
SVM	1515.80	534.40	2.58%	1.05%

Table 5.15. Performance of the novelty detection approach on test set for time series 2. Comparing results from different classifiers.

Classifier	Epochs	Hidden Units	Classification error	
			mean	s.dev
MLP committee	7869.75	48	4.54%	3.29%
RBF-DDA (default)	4	656.1	8.79%	1.93%
RBF-DDA (θ^- sel.)	3.6	671.4	**3.92%**	**0.97%**
SVM	1725.40	550.8	7.17%	2.82%

Table 5.16. Performance of the novelty detection approach on test set for time series 3. Comparing results from different classifiers.

needs to carefully select two parameters (C and γ). In the RBF-DDA experiments we have used the same value of θ^- for each augmented training set of a given time series, whereas the model selection for SVMs was performed separately for each augmented training set. This was done because we have observed that the values of C and γ were critical for performance and the best values for these parameters were different for each version of the augmented training set of a given series. This increases even more the training process of SVMs for novelty detection in time series.

The results, therefore, show that if training time is an important issue, RBF-DDA with θ^- selection should be selected as the classifier for the novelty detection in time series problem. On the other hand, if classification performance is the only important concern, the classifier should be carefully selected using a validation set or cross-validation, since the simulations results have shown that SVM achieved the best results on two time series and RBF-DDA with θ^- selection achieved the best results on the remaining two.

Classifier	Epochs	Hidden Units	Classification error	
			mean	s.dev
MLP committee	6948.30	36	0.79%	0.69%
RBF-DDA (default)	4	644.4	7.13%	1.32%
RBF-DDA (θ^- sel.)	3.7	672.5	1.04%	0.53%
SVM	1768.50	799.90	**0.04%**	**0.13%**

Table 5.17. Performance of the novelty detection approach on test set for time series 4. Comparing results from different classifiers.

Classifier	Test set class. error
MLP committee	2.24%
RBF-DDA (default)	8.36%
RBF-DDA (θ^- sel.)	**1.78%**
SVM	3.26%

Table 5.18. Comparison of different classifiers for novelty detection in time series. Means over the results of the four time series considered.

5.4 A METHOD BASED ON RBF-DDA WITHOUT NEGATIVE SAMPLES

In many practical problems the time series available represent the normal behavior of the system. Thus, all patterns generated from these series represent only normality. The method proposed in section 5.3 is based on negative samples in order to represent the novelty regions outside the envelope. The problem with this method is that the number of negative samples added to the augmented training set has an important influence on classification performance [ONM03].

In this section, an alternative method for novelty detection in time series that does not need negative samples is proposed [ONM04d]. The method uses RBF neural networks to classify time series windows as normal or novelty. It is based on the dynamic decay adjustment algorithm, originally proposed for training RBF networks for classification [BD95]. Results of experiments using the same four real-world time series used to evaluate the previous method (figure 5.5) are given. The aim of these experiments was to compare the proposed method with the method based on negative samples.

The method proposed in this section is also based on the idea of the envelope, discussed in section 5.2. The patterns used in this method are also generated from the time series according to section 5.2. The method also requires a fixed time series window size w. For each original time series window pattern, two additional patterns are generated. The first corresponds to the upper envelope and the second, to the lower one. That is, the first additional pattern is obtained from the original by adding the percent deviation p_1 that defines the upper envelope. The second additional pattern is obtained by subtracting p_1 from the original pattern. Therefore, an augmented training set three times larger than the original training set is generated.

Notice that no negative samples were introduced in the augmented training set. The upper and lower envelope patterns are added to the augmented training set in order to help the DDA algorithm adjust the standard deviation σ_i of RBF units associated with normal patterns properly. In this way, after training, standard deviations are adjusted so that patterns in the novelty region produces low values for the normal output and high values for the novelty output of the classifier. It is expected that patterns with all values inside the envelope will be classified as normal patterns.

The two RBF-DDA architectures depicted in figures 5.14 and 5.3 are considered for use in conjunction with this method. In the first case there are three outputs. The first output is associated with normal patterns, that is, original time series windows. The second and the third outputs are associated with upper and lower envelope patterns,

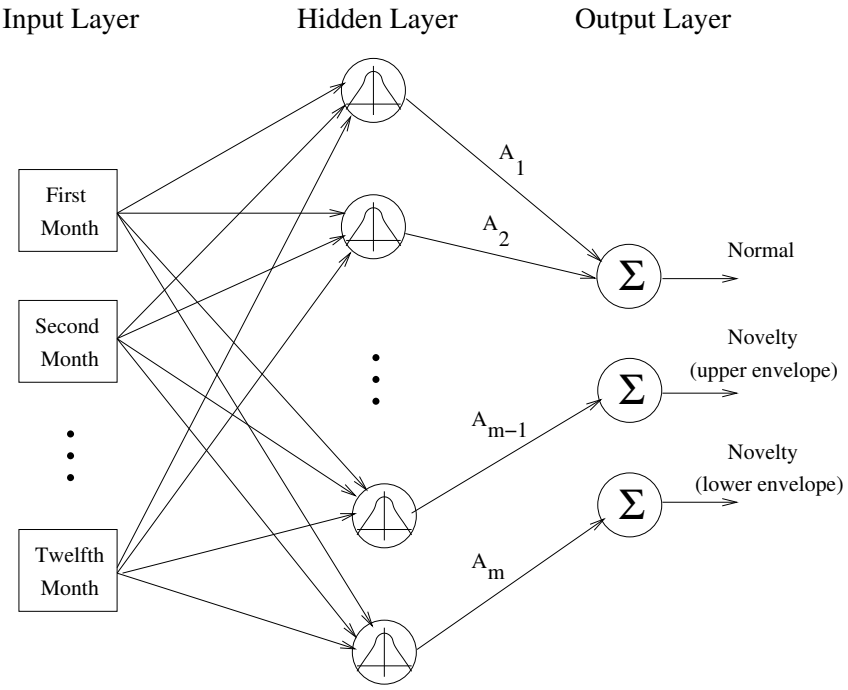


Figure 5.14. RBF network with three output units for classification-based novelty detection.

respectively. The second architecture has only two outputs, one for normal patterns and the other for novelty patterns. In the training phase the novelty output will be associated with both upper and lower envelop patterns from the augmented training set.

After training, the network classifies a given pattern as follows. If all outputs have values below 10^{-6}, the pattern is classified as novelty. If this is not the case, the winner-takes-all rule is used to classify patterns as either normal or novelty.

5.4.1 Experiments

A number of experiments using real-world time series were carried out in order to compare the performance of the method proposed in this section with the negative samples method. The times series used in the experiments are those used in section 5.3 and depicted in figure 5.5. Once again, they are firstly differentiated and then normalized.

The method proposed in this section is trained with training sets augmented by a factor of three. This is because only two additional patterns per original pattern are needed in order to represent the envelope, as discussed before. The additional patterns are generated considering an envelope created with threshold $p_1 = 0.1$. Recall that the time series used in the experiments have 84 data points and the last 12 are use to form test sets. The window size used here is $w = 12$ and therefore augmented training sets with 180 patterns will be generated for the time series considered here.

For the negative samples approach, the normal and novelty regions were defined using thresholds $p_1 = 0.1$ and $p_2 = 0.5$ as before The generation of augmented training sets for this method works by the addition of $n-1$ normal random patterns and n random novelty

patterns for each original pattern, as discussed before. In the experiments, $n = 10$ and $n = 20$ were used. With this, the training sets increase by factor of 20 and 40, respectively. Thus, for the series considered, experiments were carried out with augmented training sets having 1200 or 2400 patterns.

The series depicted in figure 5.5 are supposed to represent the normal behavior. However, it is interesting to evaluate the generalization performance of the method and, therefore, an unseen test set with both normal and novelty patterns is needed for this purpose. Hence, augmented test sets with original patterns, obtained from the last 12 data points from the series, together with normal random patterns and novelty random patterns are generated from the original test sets.

For each series, two different augmented test set sizes were considered. The first is generated from the original test set by adding 9 normal random patterns and 10 random novelty patterns for each original pattern in the test set. In this way, the augmented test set gets increased by a factor of 20. The second alternative increases the test set by a factor of 200 by adding 99 normal random patterns and 100 random novelty patterns for each original pattern. Thus, in the former case there will be 240 patterns in the augmented test sets and in the later case there will be 2400 patterns.

For the method proposed in this section, the network for each series is trained only one time, because RBF-DDA does not depend on weights initialization. Moreover, the training set is fixed. In contrast, the augmented training set used in conjunction with the method based on negative samples depends on random samples and therefore is variable. The augmented test set used to test both method is also variable because of the random patterns used to form it. Hence, each trained network is tested ten times in order to take into account the variability of the random patterns added to form the augmented test sets. On the other hand, for the negative samples approach each network is trained with ten different versions of the training set generated from different seeds, to take into account the variability of the random patterns added to form them. The network was also tested ten times, for the same reason. In the experiments RBF-DDA with default parameters were used in conjunction with both novelty detection methods, in order to perform a fair comparison between them.

5.4.1.1 Results Table 5.19 presents results obtained after training the networks for series 1, 2, 3, and 4. It contains the mean number of epochs used in training, the mean number of hidden units in the resulting network and the mean and standard deviation of the classification error on the test set. Table 5.20 presents the mean and standard deviation for the false alarm rate and the undetected novelty rate on the test set.

The negative samples method uses training sets with 1200 patterns whereas the method proposed in this section has 180 patterns on training sets. Tables 5.19 and 5.20 present the mean and standard deviation across ten executions. For each time series, augmented test sets with 240 and 2400 patterns were used in order to study the influence of test set size on the performance of the methods. Those results are also depicted in figures 5.15, 5.16, 5.17, and 5.18 in order to help the visual comparison of the results.

The results presented in table 5.19 and figures 5.15 and 5.16 show that the proposed method produces classification error smaller than the negative samples approach for all

Method	Epochs	Hidden Units	Class. error	
			mean	s.dev
Time series 1, test set with 240 patterns				
Without negative samples (3 outputs)	3	150	1.17%	0.58%
Without negative samples (2 outputs)	3	149	1.21%	0.57%
Negative samples	4	623.2	11.00%	2.86%
Time series 1, test set with 2400 patterns				
Without negative samples (3 outputs)	3	150	1.53%	0.18%
Without negative samples (2 outputs)	3	149	1.56%	0.19%
Neg. samples	4	623.2	17.67%	2.86%
Time series 2, test set with 240 patterns				
Without negative samples (3 outputs)	3	171	5.33%	1.48%
Without negative samples (2 outputs)	3	170	5.50%	1.70%
Neg. samples	4	621.7	6.50%	1.86%
Time series 2, test set with 2400 patterns				
Without negative samples (3 outputs)	3	171	6.22%	0.39%
Without negative samples (2 outputs)	3	170	6.36%	0.44%
Neg. samples	4	621.7	12.78%	1.43%
Time series 3, test set with 240 patterns				
Without negative samples (3 outputs)	3	181	2.92%	1.11%
Without negative samples (2 outputs)	3	180	3.17%	1.23%
Neg. samples	4	644.3	8.79%	1.93%
Time series 3, test set with 2400 patterns				
Without negative samples (3 outputs)	3	181	4.30%	0.43%
Without negative samples (2 outputs)	3	180	4.61%	0.46%
Neg. samples	4	644.3	10.87%	0.87%
Time series 4, test set with 240 patterns				
Without negative samples (3 outputs)	2	181	3.79%	1.08%
Without negative samples (2 outputs)	2	180	3.88%	1.09%
Neg. samples	4	644.4	7.13%	1.32%
Time series 4, test set with 2400 patterns				
Without negative samples (3 outputs)	2	181	3.98%	0.33%
Without negative samples (2 outputs)	2	180	4.05%	0.33%
Neg. samples	4	644.4	14.10%	1.69%

Table 5.19. Classification performance of the novelty detection methods on test sets for each time series

Method	False alarm		Undetec. novelty	
	mean	s.dev	mean	s.dev
Time series 1, test set with 240 patterns				
Without negative samples (3 outputs)	0.75%	0.47%	0.42%	0.44%
Without negative samples (2 outputs)	0.79%	0.50%	0.42%	0.44%
Neg. samples	0.00%	0.00%	11.00%	2.86%
Time series 1, test set with 2400 patterns				
Without negative samples (3 outputs)	0.95%	0.19%	0.58%	0.06%
Without negative samples (2 outputs)	0.98%	0.19%	0.58%	0.06%
Neg. samples	0.00%	0.00%	17.67%	2.86%
Time series 2, test set with 240 patterns				
Without negative samples (3 outputs)	4.58%	1.08%	0.75%	0.61%
Without negative samples (2 outputs)	4.75%	1.27%	0.75%	0.61%
Neg. samples	0.00%	0.00%	6.50%	1.86%
Time series 2, test set with 2400 patterns				
Without negative samples (3 outputs)	5.20%	0.42%	1.02%	0.19%
Without negative samples (2 outputs)	5.34%	0.46%	1.02%	0.19%
Neg. samples	0.00%	0.00%	12.78%	1.43%
Time series 3, test set with 240 patterns				
Without negative samples (3 outputs)	2.46%	0.89%	0.46%	0.36%
Without negative samples (2 outputs)	2.71%	1.02%	0.46%	0.36%
Neg. samples	0.00%	0.00%	8.79%	1.93%
Time series 3, test set with 2400 patterns				
Without negative samples (3 outputs)	3.48%	0.40%	0.82%	0.17%
Without negative samples (2 outputs)	3.79%	0.42%	0.82%	0.17%
Neg. samples	0.00%	0.00%	10.87%	0.87%
Time series 4, test set with 240 patterns				
Without negative samples (3 outputs)	2.29%	0.88%	1.50%	0.71%
Without negative samples (2 outputs)	2.38%	0.90%	1.50%	0.71%
Neg. samples	0.00%	0.00%	7.13%	1.32%
Time series 4, test set with 2400 patterns				
Without negative samples (3 outputs)	2.37%	0.28%	1.61%	0.23%
Without negative samples (2 outputs)	2.44%	0.27%	1.61%	0.23%
Neg. samples	0.00%	0.00%	14.10%	1.69%

Table 5.20. False alarm and undetected novelty rates on test sets for each time series

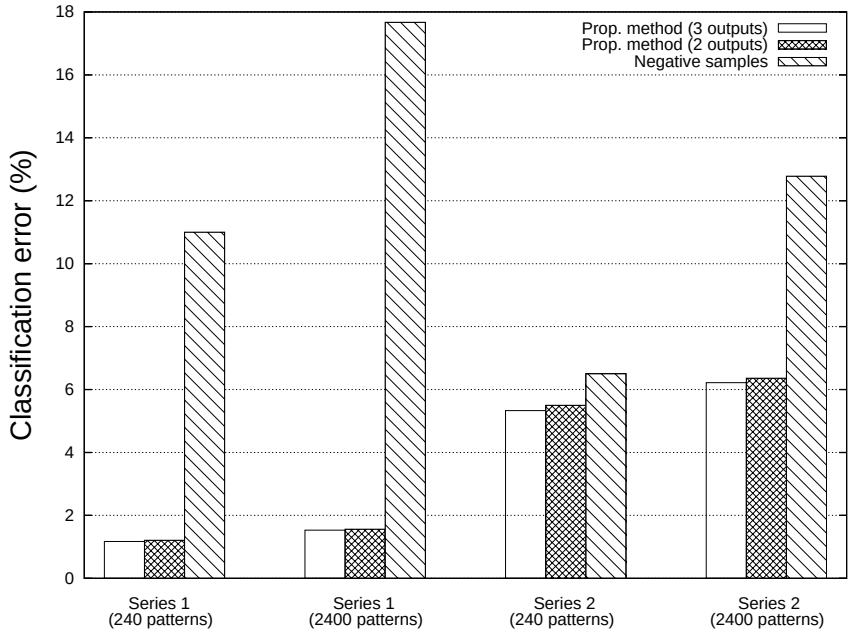


Figure 5.15. Classification error on test sets for series 1 and 2.

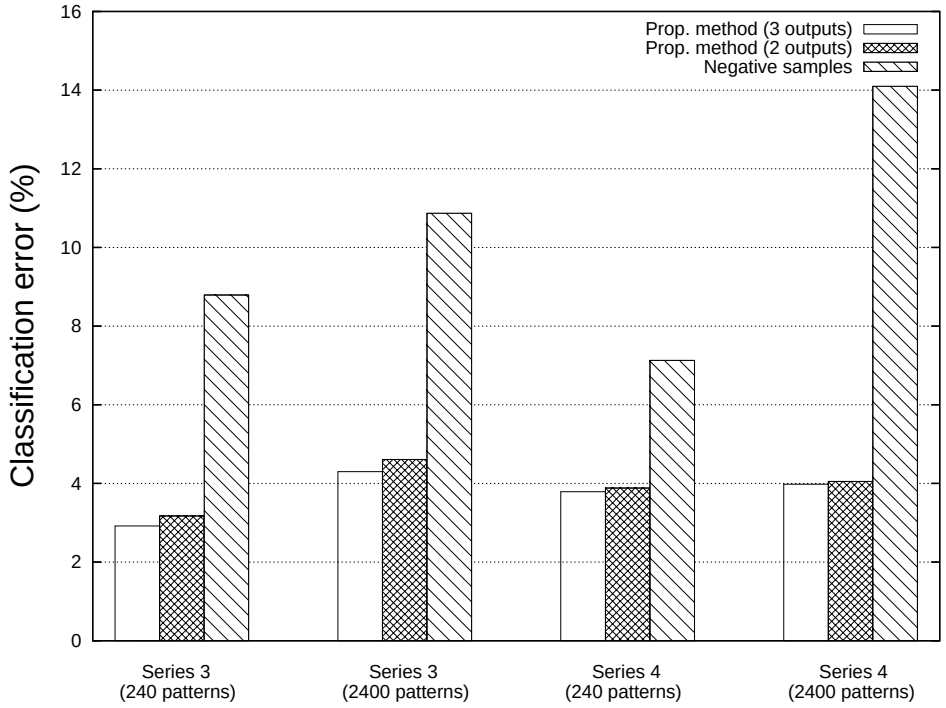


Figure 5.16. Classification error on test sets for series 3 and 4.

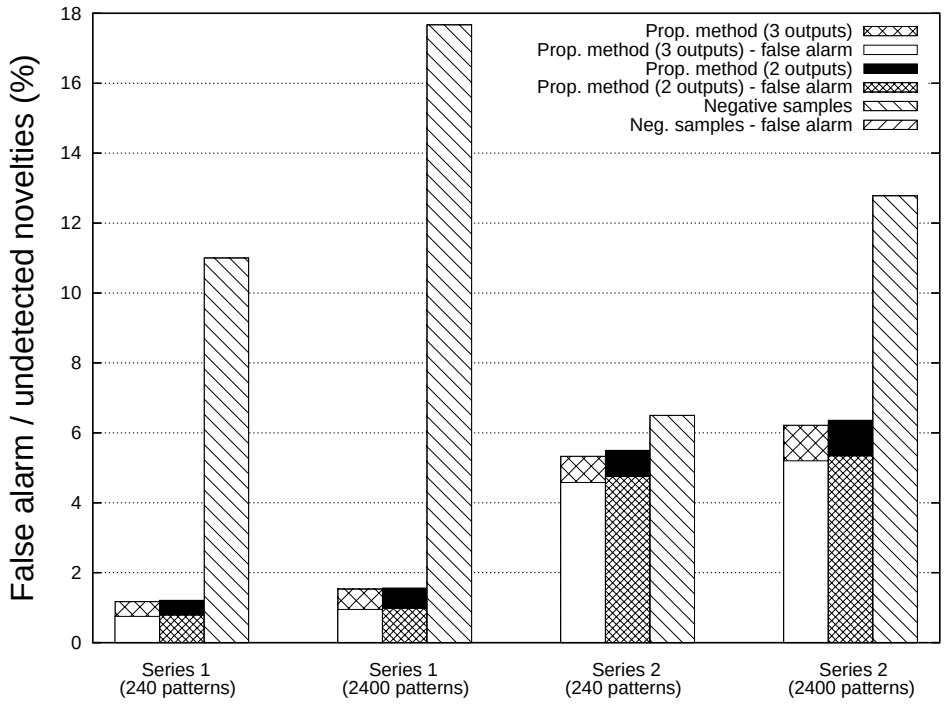


Figure 5.17. False alarm and undetected novelties rates on test sets for series 1 and 2.

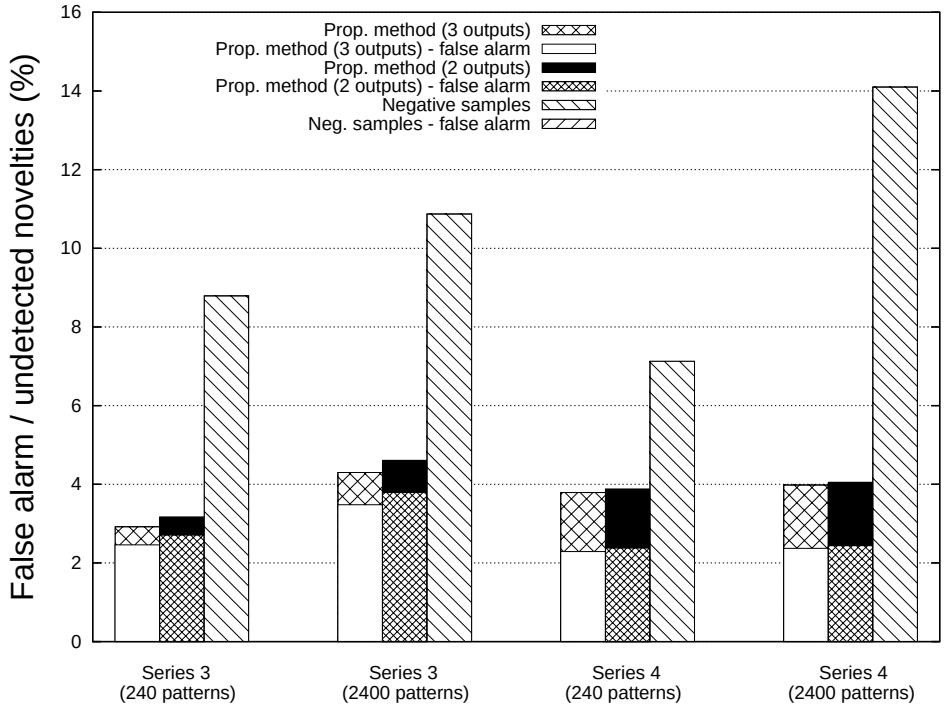


Figure 5.18. False alarm and undetected novelties rates on test sets for series 3 and 4.

Method	**Mean Class. Error**
Test sets with 240 patterns	
Without negative samples (3 outputs)	3.30%
Without negative samples (2 outputs)	3.44%
Negative samples	8.36%
Test sets with 2400 patterns	
Without negative samples (3 outputs)	4.01%
Without negative samples (2 outputs)	4.15%
Negative samples	13.86%

Table 5.21. Mean classification errors across the four time series for each method.

time series considered. This is true for both architectures used in conjunction with the proposed method (figures 5.3 and 5.14). The performance gain is variable across the series, however, it can be very high, for example, for series 1 with 2400 patterns on test sets, the classification error decreases from 17.67% (for the negative samples approach) to 1.53% (for the proposed method with three outputs). These results also show that the RBF architecture with three outputs produces slightly better results than the two outputs architecture. It can also be noted that the negative samples approach is much more sensible to the increase in test set size. For this approach, when test sets increase from 240 to 2400 patterns the classification error increases by a factor of almost two for some series.

The method proposed in this section and the negative samples method behave quite differently with respect to false alarm and undetected novelty rates, as shown in table 5.20. The negative samples method always produces 0% false alarm rate. All the misclassifications produced by this method happen with novelty patterns. In contrast, the proposed method produces more false alarms than undetected novelties.

The results from table 5.19 are summarized in table 5.21. This table presents the mean classification errors across the four time series for each method. It clearly shows that the proposed method performs much better and that its performance is much less dependent on test set size. The relative increase in classification error when the test sets increases from 240 to 2400 patterns are: 21.5% for the proposed method with three outputs; 20.6% for the proposed method with two outputs; and 65.8% for the negative samples method.

Finally, additional experiments with the negative samples method from section 5.3 using RBF-DDA with default parameters were performed. This time the augmented training sets will have more patterns. Recall that results presented in tables 5.19 and 5.20 where obtained with augmented training sets with 1200 patterns. Table 5.22 presents the mean classification errors on test sets for the four time series using augmented training sets with 2400 patterns. The test sets also have 2400 patterns. The results show that the classification errors on test sets decreases when the training sets get increased. Even so, the method proposed in this work still produces much better results as can be seen in tables 5.19 and 5.21. It has the additional advantage of using a fixed number of patterns in the training set, because it does not depend on negative samples.

Series	Class. Error	
	mean	s.dev
Time series 1	8.71%	1.76%
Time series 2	8.84%	1.36%
Time series 3	6.91%	1.02%
Time series 4	8.74%	2.09%
Mean	8.3%	0.93%

Table 5.22. Mean classification errors for the negative samples method with 2400 patterns on both training and test sets.

5.5 CONCLUSION

This chapter has presented two novel methods for the detection of novelties in time series based on classification. Both methods are capable of classifying a given time series window as normal or novelty. In contrast to other classification-based methods described in the literature, the methods proposed here were designed for fraud detection in financial systems. The two methods are based on the idea of the envelope used to define the regions of normality and novelty. The main difference between the proposed methods is that one of them needs to use negative samples in order to represent novelties during training whereas the other works without the need for these samples. A number of experiments with real-world time series were carried out in order to compare the proposed methods.

The method based on negative samples employs a classifier that is trained with an augmented training set formed by patterns corresponding to time series windows and by artificially generated samples. Two kinds of artificially generated samples are added to the training set: normal random samples, which lie inside the envelope, and novelty random samples (negative samples), which lie outside the envelope. Subsequently, the classifier trained with the augmented training set will be able to classify a given time series windows as normal or novelty. The classifiers compared is this task were: MLPs, RBF-DDA, two of the improved RBF-DDA classifiers proposed in chapter 4, committee machines of these classifiers and support vector machines (SVMs). Results show that the proposed methods for improving RBF-DDA considerably improve RBF-DDA in this task as well. It was also shown that the RBF-DDA with θ^- selection classifier outperformed, in the mean, all the other classifiers considered in this chapter.

The comparison between RBF-DDA with θ^- and SVM has shown that each method obtains better performance on two out of the four time series considered. In three of the time series considered, SVM produced slightly smaller classifiers. Nonetheless, we have shown that SVM takes much more time to train, since its training algorithm is more complex than DDA and its performance depends on two parameters, whereas DDA has only one parameter to be selected (θ^-). For a given time series, the results of this chapter have shown that it is important to carry out model selection - for example, using cross-validation - in order to select the best model. We recommend to use RBF-DDA with θ^- and SVM in this selection.

The problem with the first method introduced in this chapter is that it relies on

negative samples in order to represent novelties during training. Therefore, the number of negative samples introduced in the training set influences the performance of the method, as the results of the experiments presented in this chapter have shown. This problem has motivated the proposal of an alternative method that still uses the idea of the envelope to define normal and novelty regions, but does not need negative samples to represent novelties. The method is based on RBF-DDA. The RBF units are used to cover the normal region in the training phase. After training, the system is able to detect novel patterns because of the localized nature of radial basis functions. Experiments with real-world time series have shown that this method outperforms the method based on negative samples and that its performance is almost independent on the test set size. On the other hand, the generalization performance of the other method degrades when the number of patterns on the test set is increased.

CHAPTER 6

METHODS FOR NOVELTY DETECTION BASED ON TIME SERIES FORECASTING

6.1 INTRODUCTION

The most straightforward method for detecting a novelty or outlier in a time series is based on forecasting. Initially, a model is built based on the historical values of the time series by using a forecasting algorithm such as the ones reviewed in chapter 2. Next, the model is used to predict future values of the time series. If the difference between predicted and observed values is beyond a *pre-defined threshold* a novelty would be detected. This technique has been used for detecting frauds in accountancy systems [Kos00, Kos03] and payroll systems [OABS03].

The definition of the value of the threshold for detecting novelties is the main problem of novelty detection based on forecasting. In a previous work the use of relative thresholds has been proposed for accountancy auditing [Kos00]. In this chapter the use of such thresholds are reviewed and compared to the use of absolute thresholds. The latter has been proposed for use in a payroll auditing application [OABS03].

The difficulties of forecasting-based novelty detection has motivated the proposal of methods based on classification of time series windows such as those reviewed in chapter 3 and those proposed in chapter 5 of this thesis. The problem with these methods is that they are not able to detect a novelty in a particular point of the series, since they classify time series of windows of a given length w. On the other hand, forecasting-based methods are able to detect a novelty in a single point of a time series, which can be an advantage for fraud detection applications.

This chapter proposes the use of *robust confidence intervals* as a suitable means for defining the thresholds for detecting novelties in time series and therefore improve performance of forecasting-based time series novelty detection [ONM04e]. Robust confidence intervals for forecasting are obtained without making assumptions about the distributions of the errors [Mas95]. They are directly computed from the prediction errors of the forecasting algorithm for the series under analysis. In many time series forecasting works only the predictions are given. Ideally, however, confidence intervals should also be given for each prediction in order to express the confidence in that prediction [Mas95]. In other works, confidence intervals are computed assuming that the errors follow some distribution defined by a number of parameters. The errors are then used to estimate the parameters and compute the confidence. The problem is that in practice the distribution assumed for the errors may not be correct. Robust confidence intervals are more general because they do not assume nothing about the errors.

A number of experiments using real-world time series were performed in order to evaluate the proposed method. The experiments reported here were performed for short-

term forecasting (one step ahead). The results show that the real values on the test set of each time series lie within the confidence intervals. Thus, these intervals can be used as reliable thresholds for the detection of novelties.

6.2 THE PROPOSED METHODS

6.2.1 Difficulties of Time Series Novelty Detection Based on Forecasting

The method presented in [Kos00] is based on relative differences between predicted and real values. The main problem of this method is the difficulty to establish the thresholds for the errors beyond which the method should indicate novelty (which is interpreted as a fraud here). In contrast, a method based on absolute errors is investigated in this chapter. Besides using absolute errors instead of relative ones, the proposed method tries to learn its threshold based on its previous forecasting performance on the same time series.

Forecasting-based novelty detection relies on the forecasting capabilities of the algorithm used. The idea is to use the algorithm to predict the future values of the series for a given period (window). If the predicted value deviates significantly from the real value, the algorithm alerts the user. This approach was used in the analysis of monthly balances of a manufacturing firm by Koskivaara [Kos00].

Koskivaara applied MLPs as the forecasting method and has performed multivariate analysis, that is, used only one MLP to simultaneously predict the values of nine financial accounts. Each account had only 66 months available - from May 1990 to December 1995. The first 54 months were used for training while the 12 months of year 1995 were used for testing. Two MLP models were used: model 1 used two months to predict a third one and model 2 used four months to predict a fifth one (see section 6.3.3). The results obtained in the test set in that work are reproduced here in table 6.1. The test set used in this work had only normal data points [Kos00] therefore this table shows the forecasting performance of the method and the relative differences between predicted and real (target) values obtained by the neural networks. For example, the third line of table 6.1 shows that 24% of the test months had a difference between predicted and target outputs less than 5% (for model 1). For model 2, 26% of the months of the test set had a difference between predicted and target outputs less than 5%. Notice that there are $12 \times 9 = 108$ points in the test sets because they had 9 time series and 12 months in the test set.

Model 2 obtained slightly better results. Note that there are 3% and 4% (for models 1 and 2, respectively) data points for which the relative difference between predicted and real values is greater than 30%. The author does not discusses what threshold should be used to distinguish between normal and fraudulent behavior. If we set this threshold to 10%, that is, if we admit frauds above this value, then we can see from the table that the system would provide false alarms for around half of the months (to be exact, for model 1 the false alarm rate would be 58%, whereas for model 2 it would be 48%). In practice, we would like this threshold to be small so that we could detect small frauds. But this has lead to great false alarm rates in this case. An alternative would be to use 30% relative

	Model 1	*Model 2*
Test set size (months)	12	12
RMSE	0.15225	0.13942
Difference between output and target: $< 5\%$	24%	26%
Difference between output and target: $< 10\%$	42%	52%
Difference between output and target: $< 30\%$	97%	96%
Average difference	12%	10%

Table 6.1. Results from table 3 of [Kos00], comparing two MLP models for monthly balances forecasting.

error as threshold. In this case, we would get much smaller false alarm rates: 3% for model 1 and 4% for model 2. The problem is that the system would permit frauds up to 30%.

The approach describe above was also studied in our paper in the context of a payroll auditing application [OABS03]. The results from this work are also presented in this chapter. We will show that in this case a similar difficulty arises regarding the definition of thresholds. This has motivated us to propose a slightly different approach still based on forecasting but using absolute instead of relative errors. Moreover, in this alternative approach the system tries to learn the threshold based on its previous performance. Further research has lead us to proposed an enhanced method based on robust confidence intervals and discussed below.

6.2.2 Method Based on Robust Confidence Intervals

In many time series forecasting works, especially those using neural networks for forecasting, only the predicted values are provided. In practice, however, it is very important to provide an indication of the reliability of these prediction. This can be done by using confidence intervals. In many cases these intervals are computed by supposing that the errors follow a normal (Gaussian) distribution or another more general distribution with a number of parameters. Nevertheless, it is argued that in practice this assumption rarely hold and therefore it can lead to wrong confidence intervals. Robust confidence intervals were proposed to overcome this limitation, since they do not make assumptions about the distributions of the errors [Mas95].

Robust confidence intervals are generic and can be computed for predictions made be any model, such as exponential smoothing, ARIMA and neural networks models. They can be computed for both univariate and multivariate time series forecasting. They also work for both short-term and long-term forecasting.

Robust confidence intervals are computed from errors collected from a set of predictions within a known data set. Ideally, this data set should be different from the training set. Nonetheless, experience indicates that reuse of the training set may be acceptable if care is taken to avoid overfitting [Mas95]. This can be achieved by using models whose number of parameters is much smaller that the training set size. Instead, we proposed to use early stopping method discussed in chapter 2. In this case, a model with larger num-

ber of parameters can be used and overfitting is avoided by computing the generalization loss on a validation set different from the training set and stopping training when the generalization loss increases. This is the method used to avoid overfitting in this chapter.

6.2.2.1 Collecting prediction errors In this chapter the errors are collected on both the training and validation sets after training the neural networks. Suppose that p previous data points from the time series are used to predicted the following $p+1$ point. For a given data set $x_1 \cdots x_n$, the first short-term prediction error is computed by feeding the trained network with the first p values from the set (training or validation) and taking the difference between the predicted and known correct values, $\hat{x}_{p+1} - x_{p+1}$. The sign of the computed errors must be preserved. To compute the next error, p values from the data set, starting from the second one, are fed to the network to obtain the predicted value $\hat{x_{p+2}}$. The second error is $\hat{x}_{p+2} - x_{p+2}$. This process is carried out up to the end of the data set.

Errors for long-term forecasting are computed in a way similar to that used for short-term forecasting. The difference is that now predicted values are also used to feed the network. This is done in order to simulate the real use of the model. For example, suppose that prediction for a distance of two time slots in the future is needed. Suppose also that the model employs three past values of the series to predict a fourth one. In this case, the first three values of the series are fed to the network to compute the prediction which is then compared to the known correct value to give the prediction error at a distance of one time slot into the future. Now, the second and third known values of the series along with the *just predicted* value for the fourth one are fed to the network in order to obtain a prediction at a distance of two time slots ahead. This recursive process can be repeated recursively to find errors for larger distances. In this chapter only short-term forecasting is considered.

6.2.2.2 Computing robust confidence intervals A robust confidence interval for predictions is computed from the errors collected from the training and validation sets. The intuition behind robust confidence intervals consists in that errors of a given size will tend to occur with about the same frequency that they have occurred previously. In other words, the frequency of occurrence of errors of a given size in the training and validation sets is assumed to be the same when the network is used for forecasting. For example, suppose that about one-quarter of the errors collected from the training and validation sets exceeds 2.3. Then, it is reasonable to expect that when the model is used for forecasting the errors will exceed 2.3 about one-quarter of the times. As commented previously, this is true only if the model does not overfit training data.

In order to build robust confidence intervals, the collection of n errors measurements is initially sorted in ascending order, $x_1 \cdots x_n$. Next, the *sample distribution function* of the errors, $S_n(x)$, is computed according to equation 6.1.

$$S_n(x) = \begin{cases} 0, & \text{if } x < x_1 \\ r/n, & \text{if } x_r \leq x < x_{r+1} \\ 1, & \text{if } x_n \leq x \end{cases} \tag{6.1}$$

If the collection of errors used to compute $S_n(x)$ is representative of the errors that will be encountered in practice, and if n is large enough, then $S_n(x)$ can be assumed to be close to $F(x)$, the true error distribution. In this case, the confidence intervals for upcoming prediction errors are computed simply by keeping as much of the interior of $S_n(x)$ as is desired and by using the limits of this truncated collection to define the confidence interval. In practice it is usually desirable to have intervals symmetric in probability, even though this implies that it will not generally be symmetrical in the error.

For very large error collections and moderate values of p, np values should be discarded from each extreme in order to build the confidence intervals. For smaller samples, however, the recommended amount to be discarded from each extreme is $np-1$ [Mas95]. If the result is a real number it should be truncated. p is the fraction of probability in each tail of the distribution function. For example, if a 90% confidence interval is desired, $p = (1.0 - 0.9)/2 = 0.05$.

The following example illustrates the procedure used to compute robust confidence intervals. Suppose there are $n = 10$ collected errors. This collection is firstly sorted to give $\overrightarrow{x} = \{-0.3, -0.3, -0.1, 0.0, 0.0, 0.0, 0.1, 0.1, 0.2, 0.4\}$. Suppose one wants to build a 60% confidence interval based on this error collection. Notice that this is an unrealistically small collection used only to illustrate the procedure. To build a 60% confidence interval, $p = (1.0-0.6)/2 = 0.2$, which gives $np-1 = 1$. Thus, one sample must be discarded from each extreme of the error collection. Therefore, the confidence intervals for the error lie between -0.3 and and 0.2. These values are summed to future predicted values in order to build the confidence intervals for the prediction. Hence, the 60% confidence interval for a future predicted value $\hat{x}_p$ would be $\{\hat{x}_p - 0.3, \hat{x}_p + 0.2\}$.

6.2.2.3 Increasing the error collection In order to build robust confidence intervals it is important to collect a reasonable number of errors on training and validation sets [Mas95]. The problem is that in many practical financial systems, the time series of interest are quite short. For example, time series with only 66 months were analyzed by Koskivaara in an accountancy problem [Kos00]. In our work regarding payroll auditing we have analyzed time series with 84 months [OABS03].

In the payroll auditing problem, suppose we reserve 12 months for test and use four months in order to predict a fifth. Then, we are left with only 68 months for training and validation. Thus our error collection will have only 68 points. It would be desirable to have more errors in order to build more reliable confidence intervals.

We propose the system shown in figure 6.1 to tackle this problem. The idea consists in using a machine committee composed of several neural networks forecasters. Firstly, the time series is pre-processed. This step may include simple differentiation or seasonal differentiation, depending on the characteristics of the time series under analysis. Subsequently, several neural networks forecasters are trained independently with the same time series for one-step ahead forecasting. Finally, the prediction furnished by this system will be the mean of the predictions of the neural networks forecasters for the same month. The error collection is formed by considering the forecasting error of each neural network forecaster. Thus, if we use 10 forecasters, for example, we are able to increase

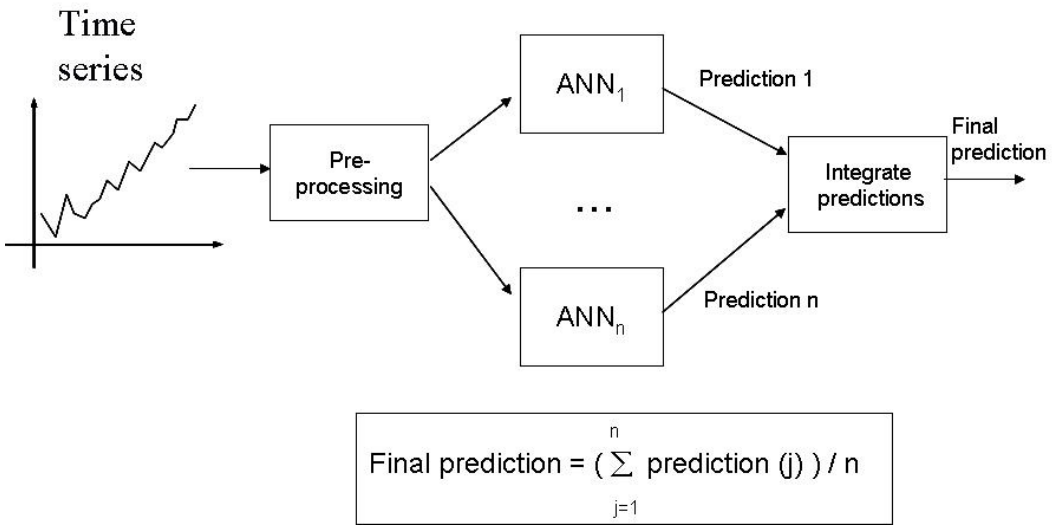


Figure 6.1. System based on machine committee used for increasing the error collection for computing robust confidence intervals.

our error collection by a factor of 10. In practice one could use any forecasting model in the committee of figure 6.1. In this work we consider only the use of MLPs or Elman networks as forecasters.

6.2.2.4 Detecting novelties with robust confidence intervals The robust confidence intervals discussed here provide a suitable threshold for novelty detection in time series. The forecasting-based novelty detection method proposed now states that a future value from a time series is to be considered a novelty if it lies beyond the confidence interval of the respective prediction. This definition is quite natural because it takes into account the previous network forecasting performance in order to establish the threshold for detection of novelties in the future. If the difference between predicted and observed values is within the confidence interval, one assumes that this difference does not correspond to a novelty, but is due to the inherent uncertainty of the forecasting model.

The novelty detection method with robust confidence intervals can be used in practical auditing applications such as payroll and accountancy auditing. The former application

was one of the motivations for carrying out this work, as discussed in chapter 1. In this case there are maybe a few hundred time series corresponding to earnings and deductions. Each time series is supposed to be free of frauds, because they were previously audited. A neural network forecasting model would be built for each time series based on historical values. Auditing would take place monthly, after the payroll is computed. Firstly, the one step ahead forecast would be computed along with the respective robust confidence intervals. Next, the value furnished by the payroll system for this month would be compared to the predicted value with confidence intervals. If the furnished value is outside the confidence intervals for the forecasting the system would detect a novelty. The auditors would focus their work on those earnings/deductions for which a novelty was detected.

6.3 EXPERIMENTS

6.3.1 Datasets

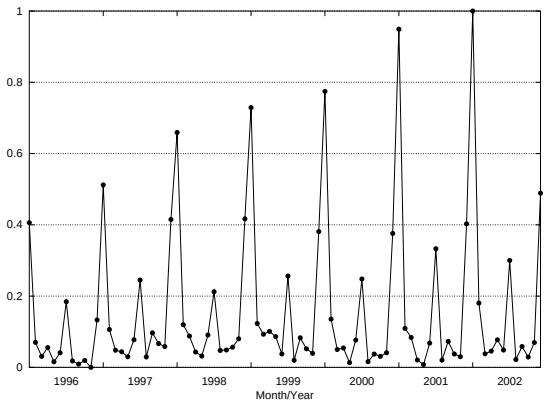
Experiments using real-world time series were carried out in order to test the performance of the forecasting-based novelty detection methods. Six real-world financial time series are used in the experiments. Five of them are shorter time series, with 84 data points.

Figure 6.2 depicts four of the time series used here. These series were also used to evaluate the classification-based novelty detection methods of chapter 5. These series are shown again here to facilitate reference. These are non-stationary seasonal time series as shown by their correlograms, presented in appendix B. These series have 84 data points corresponding to the months from January 1996 to December 2002. The first series was extracted from a real Brazilian payroll and was used previously in a study of a payroll auditing system based on neural networks forecasting [OABS03]. This series represents a monthly earning from a payroll. The remaining series are sales time series with values in the same period of the first series. Henceforth these series are referred as series 1, 2, 3 and 4, respectively.

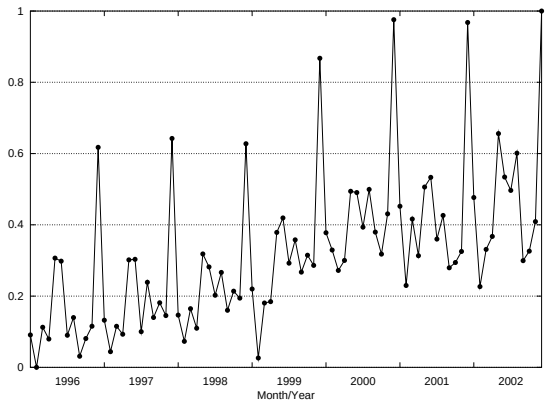
Figure 6.3 depicts the *empper* time series, which records the monthly number of employed persons in Australia from February, 1978 to April, 1991. This series has, therefore, 159 data points. It is available in a time series library available at the URL http://www-personal.buseco.monash.edu.au/~hyndman/TSDL. This series does not exhibit seasonality and it is non-stationary as shown by its correlogram, in appendix B.

Figure 6.4 shows the *earning 2* time series. This series was also extracted from a real Brazilian payroll. It also has 84 data points. It is a monthly time series with data points from January, 1996 to December, 2002. This series is also non-seasonal and non-stationary, as can be observed in its graphics and in its correlogram presented in appendix B.

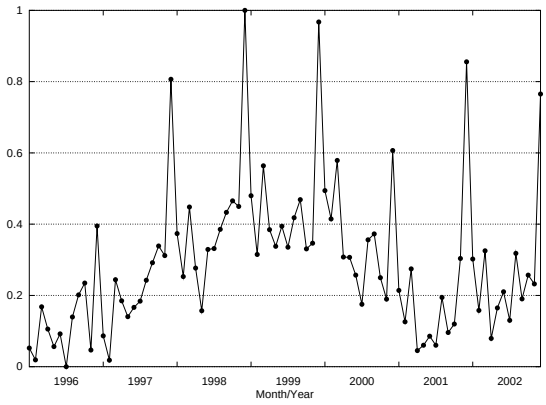
In the experiments reported in this chapter using series 1, 2, 3, 4 and *earning 2*, the first 72 months are used as training set and the last 12 months as test set. The training set is small and this makes neural networks training harder, and can lead to poor forecasting for some months. This situation is common in practice in many auditing problems. For



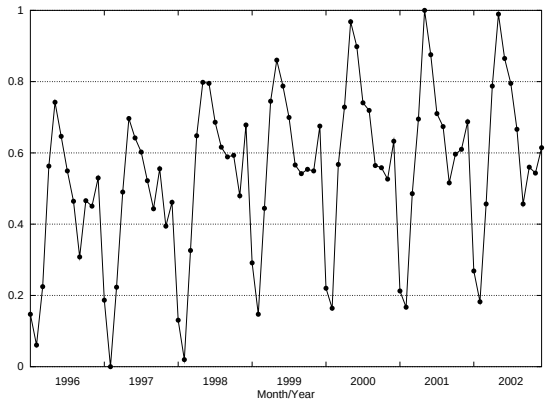
(a) Series 1: earning from a Brazilian payroll.



(b) Series 2: sales from a Brazilian company.



(c) Series 3: USA computer and software store sales.



(d) Series 4: USA hardware stores sales.

Figure 6.2. Time series used in the experiments. Each series has 84 values corresponding to months from January 1996 to December 2002. Values are normalized between 0 and 1. Series 3 and 4 are available at the URL http://www.census.gov/mrts/www/mrts.html.

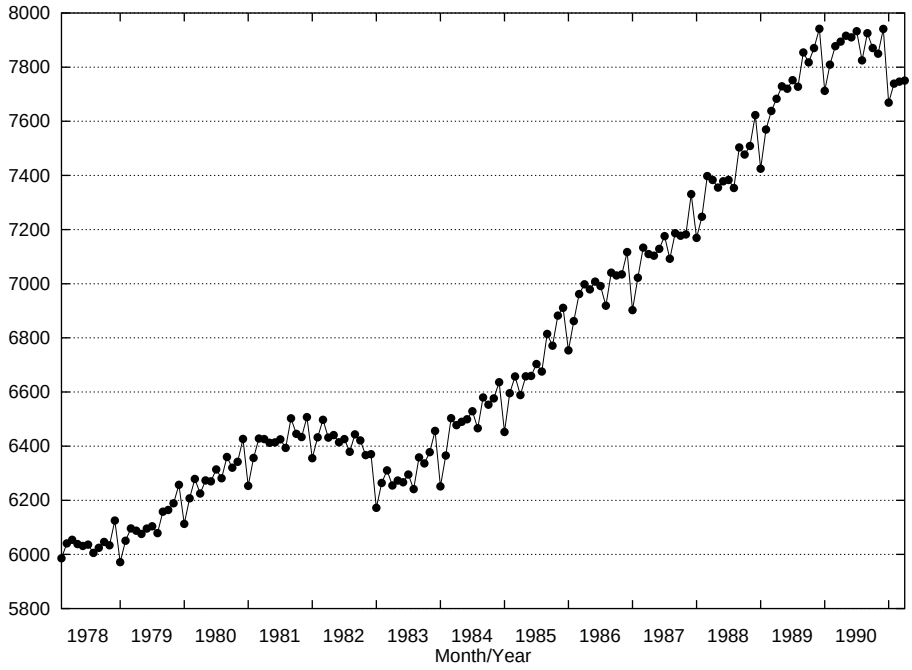


Figure 6.3. Empper Time Series: monthly number of employed persons in Australia. Feb 1978 – Apr 1991.

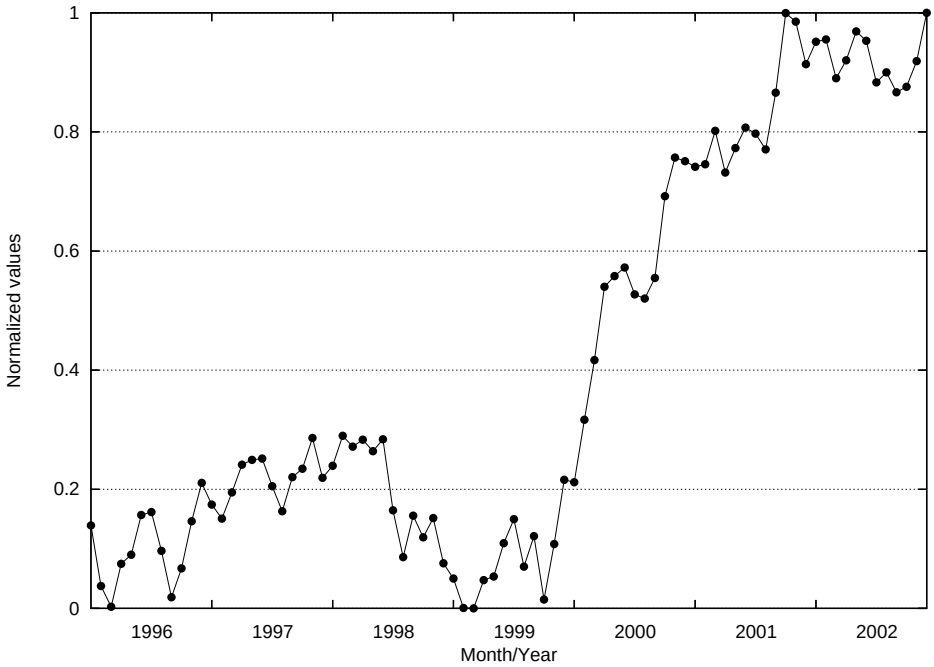


Figure 6.4. Earning 2 time series: monthly normalized values from January 1996 to December 2002.

example, the experiments reported in an accountancy auditing paper used time series with only 66 months [Kos00]. In practice, in Brazil's context at least, it will not be possible to obtain much larger data sets already audited, because payroll government systems have been automated recently and computer aided auditing of these system is even more recent.

6.3.2 Pre-Processing

Two pre-processing schemes were compared in this chapter. In the first, a simple normalization between 0 and 1 is carried out because it is known that this improves neural networks performance [Hay98]. Each series is normalized between 0 and 1 using the expression

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{6.2}$$

where x_{norm} is the normalized value correspondent to the original value x of the sum of the earning values in a given month and x_{min} e x_{max} are, respectively, the minimum and maximum values among the all values available for the series.

The second pre-processing alternative consists in differencing the time series and then applying the above normalization between 0 and 1. Differencing was reviewed in chapter 2 and aims at obtaining a stationary version of a non-stationary time series. This can improve performance of focused TLFN neural networks forecasting (see chapter 2).

6.3.3 MLP Experiments

In order to use Multi-Layer Perceptron (MLP) neural networks for time-series forecasting, a sequence of past values is used as the network input and the next value – to be predicted by the network – is used as network output in the training phase. The number of past values to be used as input is called *order of delay line*, or *window size*. Thus, for example, using an MLP with window size 2, months January and February are used as input and March as the corresponding output; months February and March are used as input and April is the corresponding output, and so on.

Neural Networks Topologies MLP neural networks with one and two hidden layers were used. All networks had only one node in the output layer, which provided the forecast. Hidden layers nodes used *sigmoidal activation function* whereas the output layer used *linear activation function*. The topologies contained all possible feed-forward connections between adjacent layers. Topologies with 3, 7, 14 and 28 nodes in the hidden layer were used for one hidden layer networks. Networks with two hidden layers had 2, 3, 4 and 5 nodes in each hidden layer. Delay line of orders 2, 4, 6 and 12 were used for each of these topologies.

Division of Data into Training, Validation and Test Sets The experiments with MLP networks used only two out of the six time series previously presented: series 1 and

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00044	0.00036	0.00374	0.00248	0.00698	0.00582	15	4
1 / 7	0.00008	0.00011	0.00569	0.00227	0.01154	0.00286	13	4
1 / 14	0.00001	0.00002	0.00662	0.00311	0.01474	0.00654	8	3
1 / 28	2.0×10^{-6}	3.0×10^{-6}	0.00740	0.00254	0.01371	0.00387	5	1
2 / 2	0.00417	0.01081	0.01143	0.02309	0.01531	0.02474	20	9
2 / 3	0.00052	0.00042	0.00344	0.00206	0.00743	0.01057	18	8
2 / 4	0.00041	0.00057	0.00369	0.00114	0.00697	0.00188	20	10
2 / 5	0.00015	0.00005	0.00417	0.00129	0.00644	0.00368	16	2

Table 6.2. Mean and standard deviation for the results of MLP with 12 order delay line experiments using series 1

earning 2. Recall that each of these series has 84 data points. When an MLP with p order delay line is used, and considering a total of 84 months available for each earning, the data set gets reduced to $84-p$ months, because the first p months are used to forecast month $p+1$, and therefore there is no possibility to forecast these initial p months.

In order to build a neural network for forecasting, data sets division into training, validation and test sets is made in time order, that is, data of the earlier period are used for training, data of a later subsequent period are used for validation, and data of the latest period are used for testing [KLSK96, YT00]. For each time series and topology considered here, the $84-p$ months available were segregated this way: the last 12 were used as test set (corresponding to year 2002), the 12 months of 2001 were used as validation set, and the $60-p$ initial months composed the training set.

Training Method The MLP networks were trained using a version of the *Levenberg-Marquadt method* [Fle87]. For each topology, 10 runs were performed with different and random initializations of weights. Training stops if the GL_5 criterion from *Proben1* [Pre94] is satisfied twice (to avoid stopping training because of initial oscillations in validation error); if the *training progress criterion* from *Proben 1* [Pre94] is satisfied, with $P_k(t) < 0.1$, or if the maximum number of 300 epochs is achieved. GL_5 criterion implements *early stopping*, one of the techniques most used to avoid overfitting, an excessive neural network adjustment to training data [Hay98, DHS00]. These techniques were reviewed in chapter 2.

Results The error measure used is the *mean square error* (MSE). The results reported here are: mean square error on training set, validation set and test set at the end of training, and number of epochs executed until the end of training. Tables 6.2 and 6.3 presents the mean and standard deviation of the results for time series 1 and *earning 2*, respectively. In these experiments the time series were not differentiated, only normalization was carried out before training and testing. This is an alternative for seasonal time series with period p when used in conjunction with networks with p delay line order [ZPH98].

The mean square error on the validation set can be used in practice to select the best

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00231	0.00024	0.00928	0.00313	0.02338	0.02459	11	2
1 / 7	0.00207	0.00032	0.05419	0.08051	0.16067	0.25810	9	3
1 / 14	0.00141	0.00030	0.05137	0.05231	0.19705	0.41316	8	1
1 / 28	0.00128	0.00034	0.08955	0.11553	0.16156	0.16004	8	2
2 / 2	0.00299	0.00142	0.00723	0.00300	0.01200	0.00897	27	30
2 / 3	0.00271	0.00040	0.01419	0.01335	0.05094	0.07068	10	3
2 / 4	0.00386	0.00549	0.01191	0.00933	0.01966	0.02042	10	3
2 / 5	0.00234	0.00027	0.01221	0.01349	0.05524	0.06371	12	4

Table 6.3. Mean and standard deviation for the results of MLP with 6 order delay line experiments using *earning 2* time series

topology for a given time series. Thus, the best network topology for a given series is the one that minimize this error. For each series, only the results obtained with the best order of delay line are presented. This is done for each different number of hidden layers and number of nodes in each hidden layer considered. For series 1, 12 order delay line has produced the best results, whereas for earning 2, the best results were obtained by a 6 order delay line. These results are shown in tables 6.2 and 6.3, respectively.

For series 1, table 6.2 shows that the best MLP among the architectures considered had 12 order delay line and two hidden layers, with three nodes in each hidden layer. For this architecture, the mean MSE in test set in ten executions was 0.00743. For *earning 2*, table 6.3 shows that the best MLP obtained had 6 order delay line, two hidden layers, and two nodes in each hidden layer. In this case, the mean MSE in the test set was 0.01200.

6.3.4 Elman Networks Experiments

Elman neural networks are partial recurrent networks developed for temporal processing [Hay98], [KLSK96]. These networks have *context units* whose nodes make recurrent connections between hidden and input layers. Recurrent connections provide hidden nodes inputs with past values of their own outputs. This adds memory to the network and provides better temporal processing performance. Originally, Elman networks had only one hidden layer, however, extensions with more than one hidden layer are now available [Zel98].

Experiments with Elman networks used the six series presented previously in this chapter. Division of data into training, validation and test sets was the same used for MLP networks for the five time series with 84 data points. For the larger *empper* time series, the last 12 points are used as test set, the previous 12 points for the validation sets and the remaining points for the training set.

Elman networks used the same topologies as MLP networks of section 6.3.3, except, of course, for the context units. This was done in order to compare performance of these networks in the data sets available.

The training algorithm used for Elman networks was a version of the *resilient backpropagation* (RPROP) [RB93], with parameters $\Delta_0 = 0.1$, $\Delta_{max} = 50$. The choice of

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00121	0.00046	0.00294	0.00133	0.00443	0.00191	163	95
1 / 7	0.00189	0.00055	0.00249	0.00068	0.00286	0.00064	73	33
1 / 14	0.00205	0.00067	0.00384	0.00337	0.00319	0.00227	72	35
1 / 28	0.00139	0.00054	0.00724	0.01311	0.00521	0.00222	84	21
2 / 2	0.00100	0.00080	0.00330	0.00133	0.00515	0.00172	225	246
2 / 3	0.00086	0.00038	0.00313	0.00187	0.00463	0.00226	85	31
2 / 4	0.00102	0.00064	0.00319	0.00187	0.00465	0.00222	110	100
2 / 5	0.00170	0.00118	0.00286	0.00168	0.00449	0.00228	69	16

Table 6.4. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 1

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00145	0.00036	0.00428	0.00186	0.00363	0.00098	259	140
1 / 7	0.00122	0.00018	0.00292	0.00089	0.00239	0.00067	285	61
1 / 14	0.00131	0.00047	0.00353	0.00130	0.00335	0.00137	314	165
1 / 28	0.00171	0.00058	0.00406	0.00137	0.00391	0.00164	171	101
2 / 2	0.00136	0.00067	0.00486	0.00214	0.00508	0.00248	426	297
2 / 3	0.00091	0.00019	0.00409	0.00080	0.00405	0.00069	408	162
2 / 4	0.00110	0.00023	0.00399	0.00133	0.00384	0.00119	301	144
2 / 5	0.00106	0.00044	0.00401	0.00124	0.00410	0.00141	385	283

Table 6.5. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 1 (with differentiation)

these parameters' values was suggested by [RB93] and it is not critical to the algorithm. This is one of its advantages over the original backpropagation algorithm, the other being its faster convergence. The stopping criteria were the same used for MLP networks and described in section 6.3.3, with the exception that here the maximum number of epochs allowed for training was 2,000.

Experiments with Elman networks were carried out with both original and differenced versions of each time series. In all cases, the series is normalized before training and testing.

Results The error measure used in this case was also the mean square error (MSE) and the results reported were the same used for MLP networks. For Elman networks, architectures with delay line orders of 2, 4, 6 and 12 using the number of hidden layers and nodes in hidden layers as for MLPs were used. The best networks achieved had 12 order delay line for series 1, 2, 3,4 and for the *empper* time series. For *earning 2* time series 6 order delay line obtained the best results, as in the case of MLP networks.

Tables 6.4 and 6.5 show the Elman networks results for time series 1 for the original and differenced versions, respectively. The results for *earning 2* time series are presented in tables 6.6 and 6.7; for series 2 in tables 6.8 and 6.9; for series 3 in tables 6.10 and 6.11; for series 4 in tables 6.12 and 6.13; and for the *empper* time series in tables 6.14 and 6.15.

Table 6.4 shows that the best Elman network topology achieved in this work for series

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.02230	0.01544	0.04908	0.04425	0.05728	0.05703	23	22
1 / 7	0.02145	0.02329	0.03030	0.04279	0.01794	0.02636	26	14
1 / 14	0.00805	0.00344	0.01061	0.01628	0.00576	0.00594	30	10
1 / 28	0.00757	0.00251	0.00792	0.00538	0.00508	0.00367	32	7
2 / 2	0.03291	0.01129	0.12269	0.10173	0.06999	0.07056	34	60
2 / 3	0.03135	0.01400	0.04490	0.06294	0.03454	0.04048	25	30
2 / 4	0.02936	0.01263	0.06923	0.05625	0.06999	0.06822	19	11
2 / 5	0.02402	0.01149	0.07708	0.08422	0.10761	0.11968	23	12

Table 6.6. Mean and standard deviation for the results of Elman networks with 6 order delay line experiments using the *earning 2* time series

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.04819	0.00257	0.05662	0.00688	0.03870	0.01201	43	23
1 / 7	0.04858	0.00264	0.05613	0.00853	0.04134	0.01335	40	14
1 / 14	0.04358	0.00560	0.06057	0.00651	0.06181	0.01505	96	67
1 / 28	0.04998	0.00476	0.06275	0.00623	0.06755	0.01992	56	25
2 / 2	0.04743	0.00227	0.05385	0.00079	0.03196	0.00193	39	6
2 / 3	0.04756	0.00102	0.05362	0.00129	0.03056	0.00095	37	6
2 / 4	0.04913	0.00161	0.05289	0.00217	0.03014	0.00154	34	5
2 / 5	0.04924	0.00165	0.05427	0.00318	0.03277	0.00358	33	5

Table 6.7. Mean and standard deviation for the results of Elman networks with 6 order delay line experiments using the *earning 2* time series (with differentiation)

1 has one hidden layer with seven nodes, 12 order delay line and has produced a value of 0.00286 for the mean MSE in the test set. This result was obtained without differencing the series. Differencing results in very similar performance for this series. This result is much better than the result obtained by the best MLP architecture, which achieved a value of 0.00743 for the mean MSE on test set. Furthermore, comparing tables 6.2 and 6.4, one observes that Elman networks achieved mean MSE in the test set smaller than those of MLP networks for every architecture considered.

For the *earning 2* time series, table 6.6 shows that the best Elman network architecture obtained a value of 0.00508 for the mean MSE on test set. It has one hidden layer with 28 nodes and a 6 order delay line. The best result was also obtained without differencing the time series. This error is also much lower than the best result obtained by MLP, whose value was 0.01200. In contrast to series 1, for *earning 2* Elman networks produced better results than MLP networks only for half of the topologies considered.

6.3.4.1 Detecting Novelties In this section, results of forecasting-based novelty detection experiments for two of the time series considered are given. The series considered here are series 1 and the *earning 2* time series. The relative error approach also used by [Kos00] was applied. Experiments with the approach based on absolute errors are also reported and the results compared with the relative errors one.

Firstly we trained again the better Elman networks topology obtained for each of

Layer(s)/	Training MSE		Validation MSE		Test MSE		Epochs	
Nodes	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00551	0.00177	0.01781	0.00687	0.01551	0.00833	42	27
1 / 7	0.01582	0.02223	0.02863	0.02111	0.03291	0.03381	32	14
1 / 14	0.00531	0.00227	0.02086	0.01047	0.01398	0.01453	43	16
1 / 28	0.03709	0.04207	0.05234	0.04309	0.04297	0.04104	41	23
2 / 2	0.02021	0.01473	0.02885	0.01058	0.03049	0.01379	26	12
2 / 3	0.00991	0.01059	0.02369	0.00856	0.02101	0.01671	41	17
2 / 4	0.01094	0.01133	0.02339	0.00853	0.02258	0.01422	37	16
2 / 5	0.01117	0.01149	0.02177	0.00879	0.01963	0.01209	43	20

Table 6.8. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 2

Layer(s)/	Training MSE		Validation MSE		Test MSE		Epochs	
Nodes	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00284	0.00229	0.00847	0.00393	0.00707	0.00312	404	387
1 / 7	0.00270	0.00101	0.00809	0.00145	0.00712	0.00191	210	320
1 / 14	0.00235	0.00171	0.00757	0.00313	0.00931	0.00333	492	849
1 / 28	0.00230	0.00091	0.00842	0.00242	0.00719	0.00265	178	202
2 / 2	0.00219	0.00122	0.00782	0.00290	0.00678	0.00236	355	264
2 / 3	0.00220	0.00144	0.00843	0.00195	0.00820	0.00355	334	361
2 / 4	0.00249	0.00123	0.00828	0.00348	0.00707	0.00262	199	192
2 / 5	0.00146	0.00088	0.00613	0.00224	0.00648	0.00147	370	277

Table 6.9. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 2 (with differentiation)

Layer(s)/	Training MSE		Validation MSE		Test MSE		Epochs	
Nodes	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 /3	0.01833	0.00628	0.05439	0.00975	0.02662	0.02256	21	7
1 / 7	0.01993	0.01222	0.04984	0.00964	0.03234	0.01906	29	12
1 / 14	0.01657	0.02188	0.05497	0.02463	0.04218	0.04238	86	122
1 / 28	0.02122	0.02044	0.06110	0.03214	0.04280	0.02745	40	12
2 / 2	0.02745	0.00747	0.05086	0.00514	0.02927	0.00962	19	6
2 / 3	0.02844	0.01015	0.04889	0.00615	0.03281	0.01231	21	12
2 / 4	0.02455	0.00755	0.05419	0.00946	0.03182	0.01341	21	8
2 / 5	0.02338	0.01507	0.05145	0.01291	0.03096	0.01370	36	33

Table 6.10. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 3

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 /3	0.00299	0.00136	0.00817	0.00228	0.00620	0.00224	296	248
1 / 7	0.00245	0.00096	0.00750	0.00122	0.00775	0.00282	260	159
1 / 14	0.00266	0.00142	0.00884	0.00222	0.00681	0.00250	255	170
1 / 28	0.00764	0.01164	0.01729	0.02291	0.01925	0.02990	109	108
2 / 2	0.00257	0.00048	0.00694	0.00169	0.00469	0.00208	230	143
2 / 3	0.00238	0.00097	0.00667	0.00189	0.00623	0.00347	224	138
2 / 4	0.00379	0.00280	0.00836	0.00411	0.00805	0.00713	166	133
2 / 5	0.00258	0.00127	0.00717	0.00193	0.00615	0.00243	210	112

Table 6.11. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 3 (with differentiation)

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00606	0.00169	0.00581	0.00260	0.00588	0.00442	62	24
1 / 7	0.00538	0.00119	0.00553	0.00160	0.00433	0.00118	68	14
1 / 14	0.03161	0.08034	0.01429	0.02374	0.01407	0.02518	65	33
1 / 28	0.01684	0.01408	0.01299	0.01009	0.01115	0.01174	48	25
2 / 2	0.00556	0.00118	0.00565	0.00142	0.00538	0.00153	91	33
2 / 3	0.00672	0.00147	0.00653	0.00197	0.00677	0.00258	60	29
2 / 4	0.00466	0.00121	0.00525	0.00114	0.00467	0.00071	110	58
2 / 5	0.00584	0.00222	0.00557	0.00153	0.00545	0.00133	80	42

Table 6.12. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 4

Layer(s)/ Nodes	Training MSE		Validation MSE		Test MSE		Epochs	
	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.00621	0.00121	0.00358	0.00180	0.00707	0.00231	65	23
1 / 7	0.00465	0.00063	0.00348	0.00119	0.00652	0.00120	83	24
1 / 14	0.00967	0.01015	0.00900	0.01167	0.01165	0.01123	70	39
1 / 28	0.01765	0.03886	0.01224	0.02421	0.01336	0.01735	62	22
2 / 2	0.00652	0.00203	0.00392	0.00146	0.00731	0.00201	67	28
2 / 3	0.00560	0.00125	0.00387	0.00115	0.00639	0.00132	64	13
2 / 4	0.00619	0.00175	0.00420	0.00355	0.00799	0.00266	66	17
2 / 5	0.00495	0.00055	0.00336	0.00160	0.00662	0.00200	66	13

Table 6.13. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using series 4 (with differentiation)

Layer(s)/	Training MSE		Validation MSE		Test MSE		Epochs	
Nodes	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 / 3	0.01593	0.01559	0.03264	0.04138	0.02952	0.03328	22	12
1 / 7	0.00278	0.00154	0.00471	0.00378	0.00389	0.00228	36	17
1 / 14	0.00477	0.00691	0.00830	0.01460	0.01203	0.02080	42	24
1 / 28	0.00828	0.00802	0.01224	0.01547	0.01559	0.01722	33	12
2 / 2	0.03455	0.02060	0.03503	0.06073	0.03518	0.05600	19	7
2 / 3	0.02598	0.01238	0.08812	0.08546	0.08784	0.07933	17	6
2 / 4	0.02278	0.00981	0.03112	0.03623	0.03078	0.03134	21	20
2 / 5	0.02542	0.01566	0.02704	0.03473	0.02727	0.03050	24	26

Table 6.14. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using the *empper* time series

Layer(s)/	Training MSE		Validation MSE		Test MSE		Epochs	
Nodes	mean	s.dev	mean	s.dev	mean	s.dev	mean	s.dev.
1 /3	0.00724	0.00070	0.00407	0.00102	0.00964	0.00258	87	45
1 / 7	0.00678	0.00099	0.00405	0.00212	0.01160	0.00276	123	60
1 / 14	0.00917	0.00893	0.00503	0.00293	0.01125	0.00354	150	88
1 / 28	0.01723	0.01698	0.00806	0.00450	0.01641	0.00920	104	85
2 / 2	0.00999	0.00873	0.00816	0.01187	0.01675	0.01314	86	31
2 / 3	0.00724	0.00138	0.00443	0.00137	0.01140	0.00129	97	38
2 / 4	0.00937	0.00776	0.00782	0.01257	0.01624	0.01413	98	97
2 / 5	0.00628	0.00068	0.00411	0.00090	0.01085	0.00090	118	31

Table 6.15. Mean and standard deviation for the results of Elman networks with 12 order delay line experiments using the *empper* time series (with differentiation)

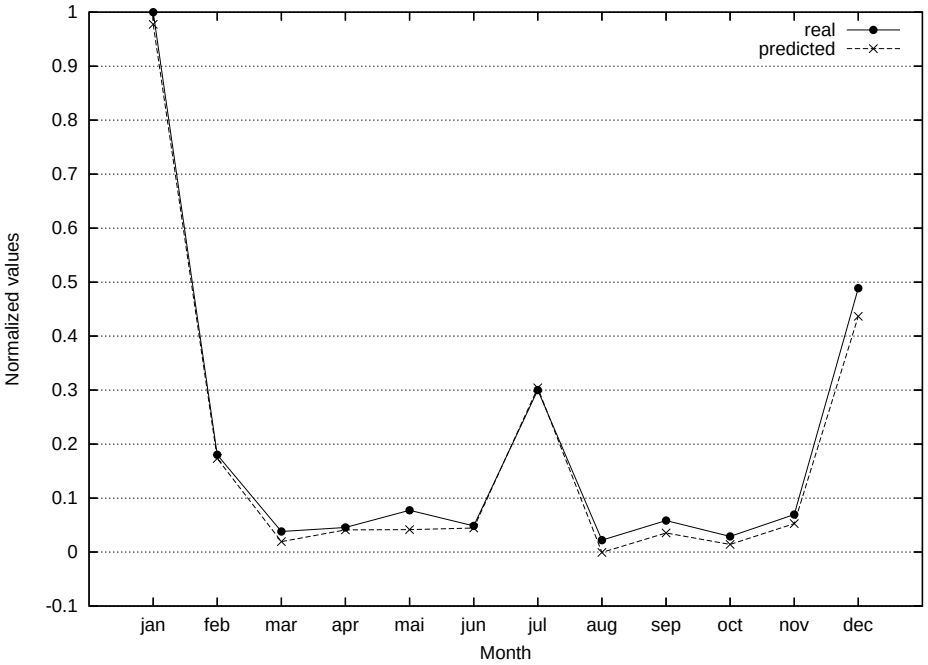


Figure 6.5. Comparing real values with values predicted by the best Elman network for series 1 in test set (year 2002).

the two series in last section, but this time these networks have been trained 30 times with different and random weights initializations. Trained networks were tested in two ways. The first used the test sets vectors in the way described in section 6.3.3. The other way used for testing was based on dynamic test set generation. For example, for a 2 order delay line, the values of months November and December, 2001 are used to forecast the value of January, 2002. To forecast the value of February, 2002, the real value of December, 2001 and the value of January, 2002 predicted by the network are used. This procedure is used to generate dynamically the others vectors of the test sets. Test sets generated dynamically obtained better results for both series.

Figure 6.5 compares real and mean predicted values for each month in the test set for series 1, using the best Elman architecture and a test set generated dynamically. The figure shows that the network visually achieves satisfactory predictions for the test set. Table 6.16 presents the relative differences between mean predicted values and real values for the test sets, for series 1 and for the *earning 2* series. It shows that there is a large error variation for different months for series 1, with very low and very high relative errors. This makes it difficult to establish a threshold for novelty detection for this series. For *earning 2*, on the other hand, we could say that a percentage difference above 8% can be considered fraud.

Difficulty to establish a threshold for relative forecasting error novelty detection has lead us to propose a different novelty detection rule based on the results presented on table 6.17. This table presents absolute difference values for each predicted month. The rule we propose establishes that the system will alarm if the absolute difference between

Month/Year	Differences for Series 1	Differences for *Earning 2*
January	2.25%	4.86%
February	3.93%	4.86%
March	48.79%	1.95%
April	9.60%	0.07%
May	46.09%	5.77%
June	8.23%	3.92%
July	1.41%	4.77%
August	102.85%	2.18%
September	39.43%	7.33%
October	50.79%	5.40%
November	24.43%	0.39%
December	10.67%	6.38%

Table 6.16. Relative percentage differences between mean values predicted by the best Elman network for each series and the respective real values, on test sets (year 2002)

Month/Year	Differences for Series 1	Differences for *Earning 2*
January	0.02250	0.04626
February	0.00709	0.04642
March	0.01854	0.01740
April	0.00437	0.00068
May	0.03563	0.05589
June	0.00400	0.03739
July	0.00424	0.04210
August	0.02273	0.01966
September	0.02299	0.06352
October	0.01473	0.04728
November	0.01700	0.00360
December	0.05216	0.06375

Table 6.17. Absolute differences between mean values predicted by the best Elman network for each series and the respective real values, on test sets (year 2002)

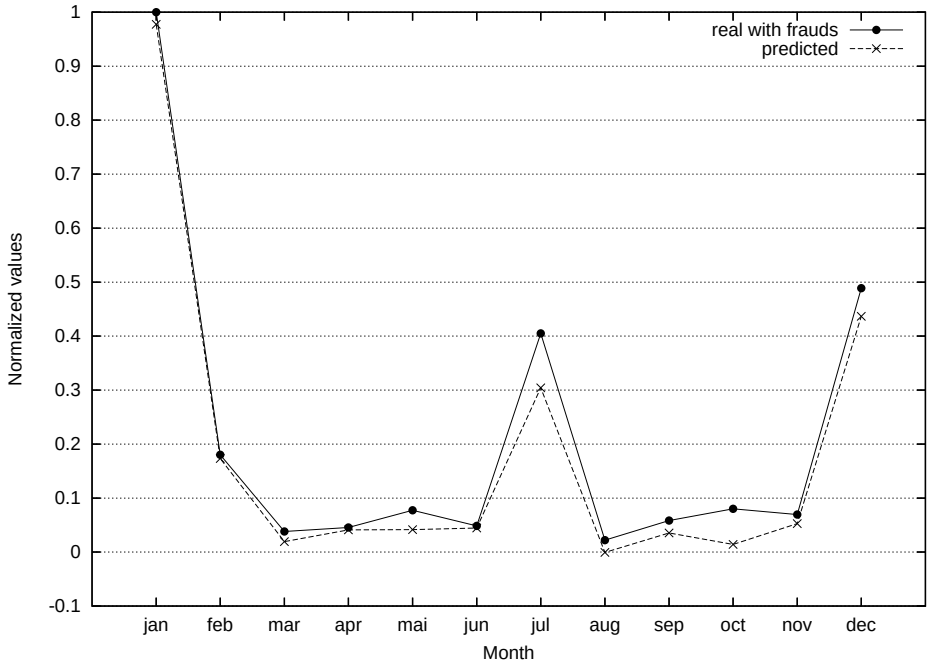


Figure 6.6. Comparing values predicted by the best Elman network for series 1 in the test set (year 2002) with values obtained during auditing. Months July and October show fraudulent values.

predicted and real value in a given month is greater than the greatest difference shown in table 6.17 for each series. This rule is proposed admitting that the network will be able to predict the values of the new year under auditing within the same forecast error of the previous period. For example, if the network was capable to predict values of year 2002 with forecast error of, at most, 0.05216, than it is expected that in year 2003 it will have a similar performance. Therefore, if in 2003 there is a month with difference between predicted and real values greater than this threshold, the system will alarm the auditor. Figure 6.6 shows that it is possible, even visually, to identify months July and October as possibly fraudulent. In this figure, frauds are simulated in these months, making the differences between predicted and real values become greater than the threshold, in this simulation the differences were 0.1 for July and 0.065 for October.

6.3.5 Experiments with Robust Confidence Intervals

This section reports experiments with the method for novelty detection in time series based on robust confidence intervals. This method was applied to the six series presented previously in this chapter. It has been applied only in conjunction with the best Elman neural network topology obtained in the previous experiments. In all experiments 95% confidence intervals for one step ahead predictions were built. This means that $p = (1 - 0.95)/2 = 0.025$. Each network is trained 10 times, as proposed in the system shown in figure 6.1. In this case, each ANN forecaster is an Elman network with the same topology. The difference between these forecasters lies in the different initializations of

Time series	Elman hidden layers & neurons	Order of delay line	Pre-processing	Number of errors (n)	$np-1$
Series 1	1 layer / 7 neurons	12	norm.	600	14
Series 2	2 layers / 5 neurons	12	diff. + norm.	590	13
Series 3	2 layers / 3 neurons	12	diff. + norm.	590	13
Series 4	2 layers / 5 neurons	12	diff. + norm.	590	13
Empper	1 layer / 7 neurons	12	diff. + norm.	1340	32
Earning 2	1 layer / 28 neurons	6	norm.	660	15

Table 6.18. The best Elman topology for each series and the characteristics of the error collections for robust confidence intervals computations

the weights and biases. The prediction for each month in the test set is given by the mean prediction across the 10 runs. The robust confidence intervals are computed by using both the training and validation sets. Elman networks are trained using the modified Rprop algorithm with default parameters, as before. Early stopping with the GL_5 criterion was also used here to avoid overfitting.

Table 6.18 shows the best Elman topology and pre-processing for each time series along with the number of errors collected to build the robust confidence intervals (n) and the number of sample errors discarded from each extreme of the ordered error collection ($np-1$). In each case, the errors were collected from both training and validation sets in 10 runs (10 machines in the committee of figure 6.1) of the experiments with different weights and biases initializations. For example, for series 1, which has 84 data points, using delay line of order 12 and 12 points for the test set, results in $84-12-12=60$ point for the training and validation sets. The number of sample errors in 10 runs is, therefore, 600 in this case. The number of samples discarded from each extreme for 95% confidence intervals is $600*0.025-1=14$.

Figures 6.7, 6.8, 6.9, 6.10, 6.11 and 6.12 depict the real and predicted values in the test sets for series 1, series 2, series 3, series 4, the *empper* series, and the *earning 2* time series, respectively. In each of these graphics the robust confidence intervals for the predictions are also depicted. Recall that the test sets for all series do not contain novelties. These graphics show that the real values from the test sets all lie within the confidence intervals boundaries for the predictions, as desired. Some of the real values are very near the respective predicted value, but other are more distant, being in the limits created by the confidence intervals.

A novelty detection system based on robust confidence intervals would indicate a novelty if the real value is outside the confidence intervals. The experiments reported here showed that these intervals, built from the error on training and validation sets, can correctly bound the region of normality. Furthermore, these intervals could be used to indicate the level of suspicion associated with a given point in the future. This level is directly related to the distance from the value in that point to the limit established by the confidence intervals (if the value is outside of the normality region). In fraud detection applications, the auditor can use these suspicions levels in order to prioritize the investigations. The auditor would prioritize investigations on points of time series

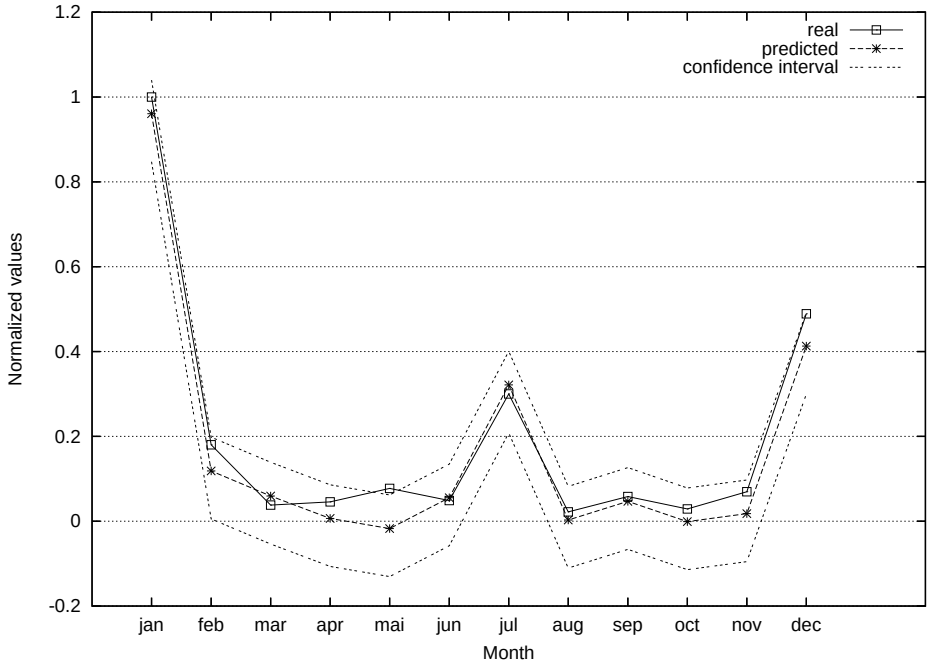


Figure 6.7. Robust confidence intervals for predictions in the test set of time series 1 (year 2002)

with greater suspicions levels.

6.4 CONCLUSIONS

This chapter has reviewed time series novelty detection with forecasting and has proposed a method for improving it based on robust confidence intervals. In order to detect novelties in a time series, the past values of the series are used to train a forecasting model such as neural network. Next, the future values predicted by the model are compared to the observed values. If the difference between these values exceeds a given threshold, a novelty is detected in that point of the series. A previous work that applied this technique to accountancy auditing proposed to pre-define a relative threshold beyond which a novelty would be indicated. In this chapter an alternative was proposed: the use of absolute errors in order to define the thresholds. The proposed method uses previous values of the errors in order to estimate the threshold to be used for future forecasting.

Robust confidence intervals are a refinement of the above idea. The idea is the same, yet robust confidence intervals approach the problem in a more formal way. These intervals are built from the previous forecasting errors of the model on the same time series. Robust confidence intervals are more reliable than other kinds of confidence intervals because no assumption on errors distributions is made. These intervals provide a suitable threshold for novelty detection in time series. A novelty is indicated if a future value of the time series is beyond the low or high limits defined by the confidence interval.

In this chapter 95% robust confidence intervals for one step ahead forecasting were built. Experiments were carried out using six real-world non-stationary time series. Four

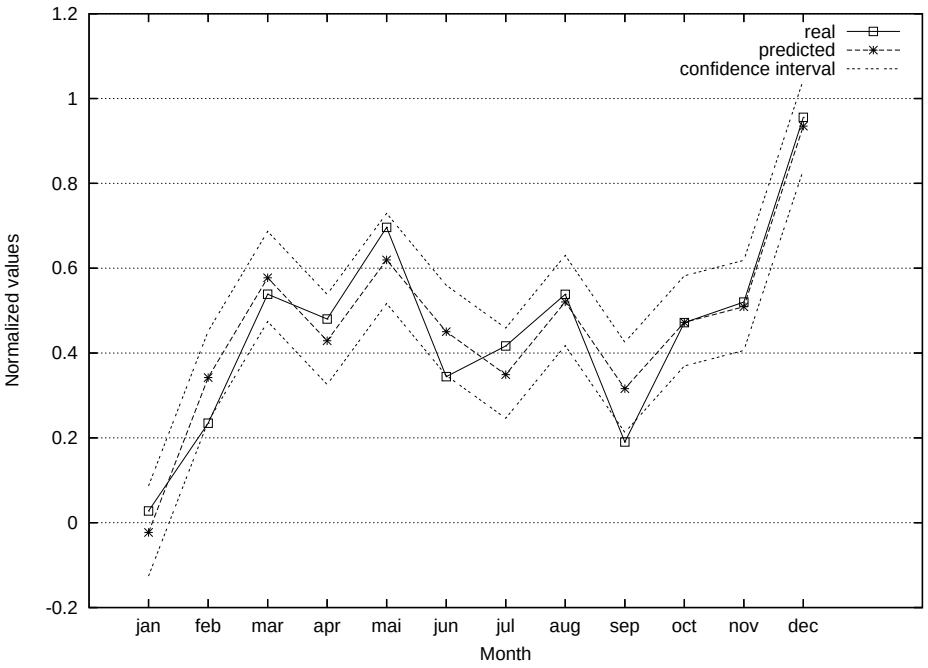


Figure 6.8. Robust confidence intervals for predictions in the test set of time series 2 (year 2002)

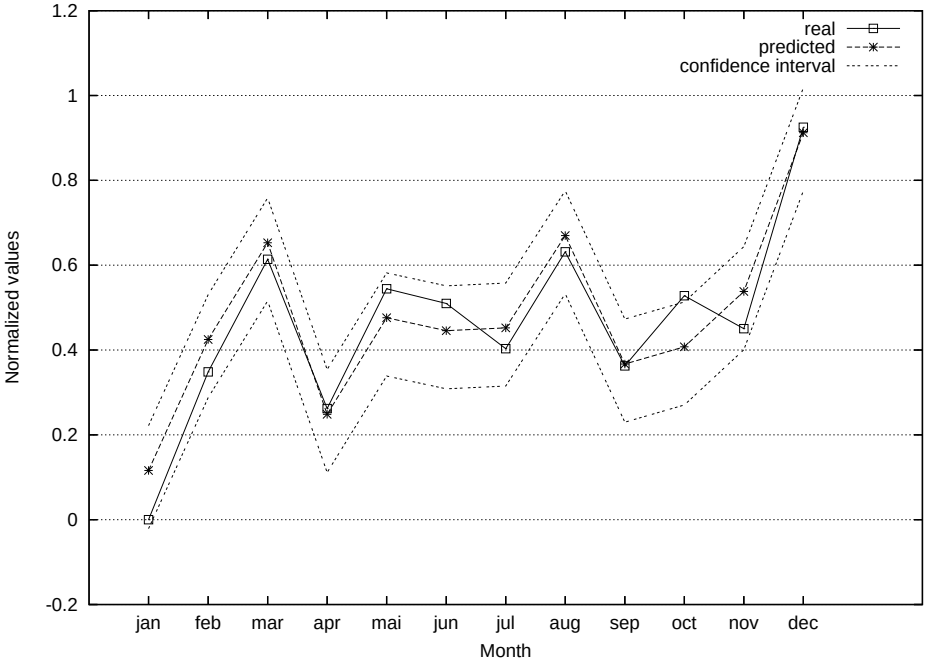


Figure 6.9. Robust confidence intervals for predictions in the test set of time series 3 (year 2002)

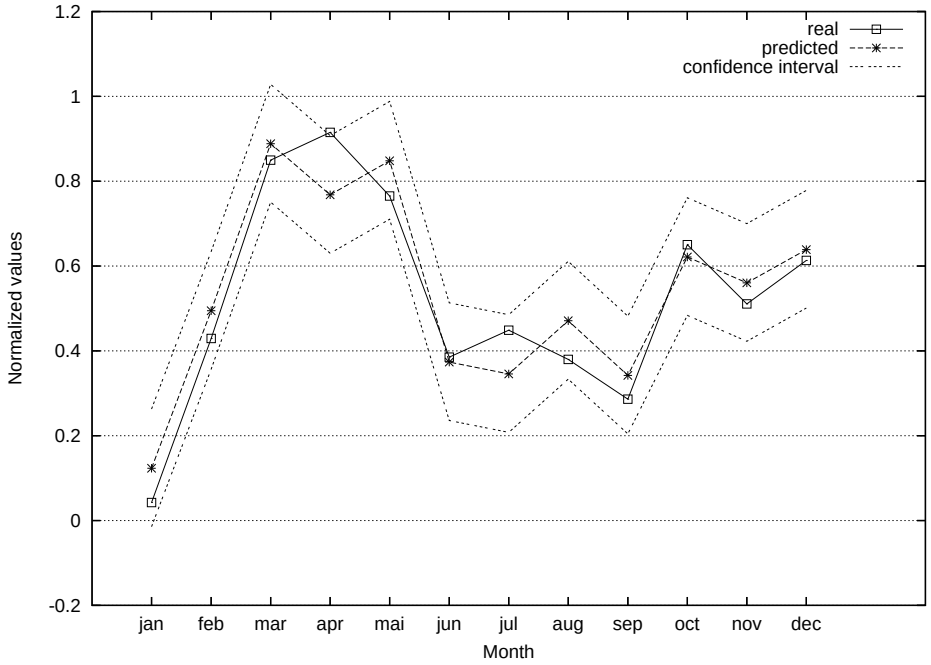


Figure 6.10. Robust confidence intervals for predictions in the test set of time series 4 (year 2002)

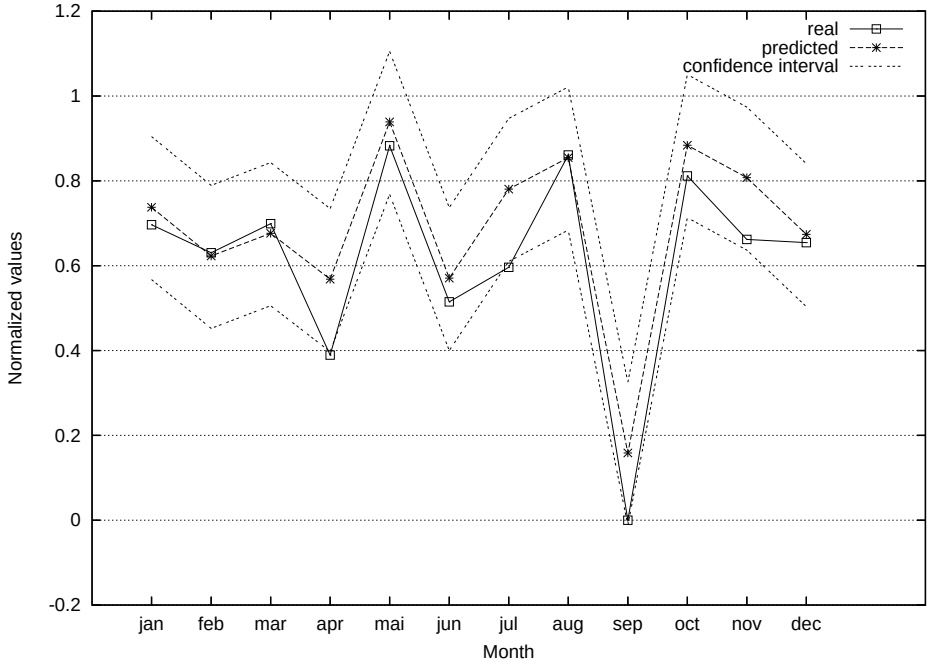


Figure 6.11. Robust confidence intervals for predictions in the test set of *empper* time series

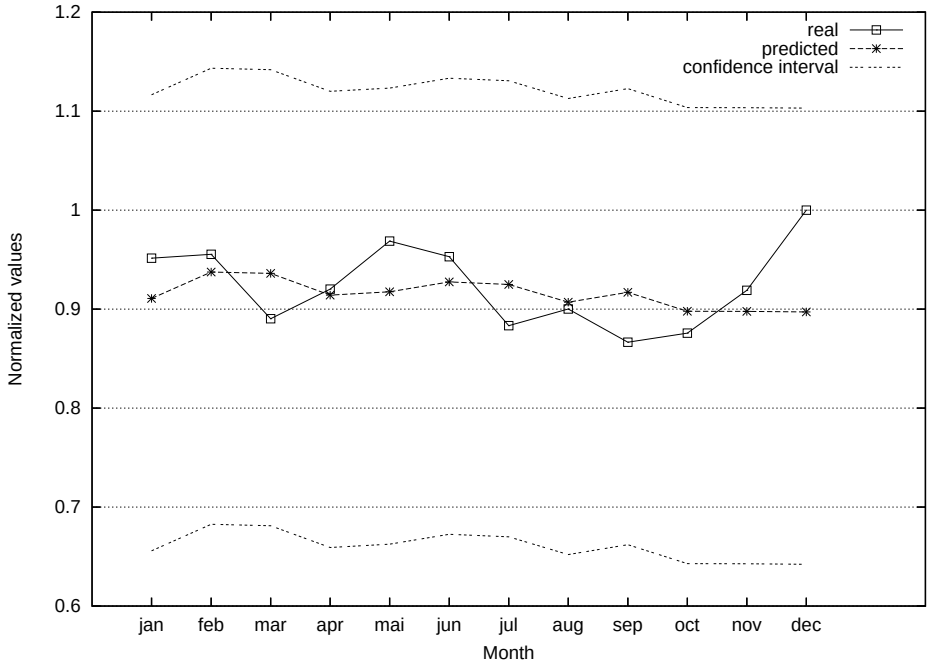


Figure 6.12. Robust confidence intervals for predictions in the test set of *earning 2* time series (year 2002)

of these series were seasonal with period 12. The test set of each time series did not contain any novelty. The results have shown that all real values of each series lay within the respective robust confidence interval. This means that the method can correctly discriminate between normal and novelty values.

Although this chapter presents experiments using only univariate short-time forecasting, there are methods for computing robust confidence for multivariate and/or long-term forecasting as well [Mas95]. This means that the novelty detection method based on robust confidence intervals introduced here may be easily extended for these cases.

In the previous chapter, classification-based methods were proposed as an alternative to forecasting-based ones. The former methods have the ability to detect a novelty in a time series window of size w. On the other hand, forecasting-based methods can detect novelty in a single point of a given time series. This can be an advantage for fraud detection applications such as accountancy and payroll auditing. An alternative to further improve novelty detection performance would consist in combining the methods of this chapter with the classification-based methods of chapter 5. The classification-based method would be applied firstly in order to detect novelty in a window of the time series. Subsequently, the forecasting-based method proposed here based on robust confidence intervals would be applied in order to identify the point of the time series in which the novelty took place.

CHAPTER 7

CONCLUSION

This thesis has investigated and presented novel methods for the detection of novelties – also referred as anomalies – in time series. A novelty in a time series is an unexpected behavior. Algorithms for detecting such novelties find practical application in a number of areas including machine fault detection, medical diagnosis and fraud detection in financial systems. The time series appearing in these domains have different characteristics and therefore different novelty detection techniques may be more appropriate for one or another. This work has focused on time series novelty detection methods for fraud detection in financial systems. These techniques are appropriate for fraud detection in accountancy or payroll systems, for example.

Novelty detection methods for time series can be classified into forecasting-based and classification-based. The former methods compare values predicted by a neural network (or other time series forecasting technique) with the values provided by the system under analysis and issue an alarm if the difference between these values is above a pre-defined threshold. These techniques have been criticized because it is difficult to define a threshold beforehand. This work has proposed a method based on *robust confidence intervals* as a suitable means to define the thresholds for novelty detection in time series.

Classification-based novelty detection methods detect novelties by classifying a given time series window of data points as normal or novelty. This work has reviewed a number of classification-based time series novelty detection methods. In contrast to forecasting-based methods, the classification-based methods available do not tackle directly time series of the kind encountered in accountancy and payroll fraud detection.

This thesis has introduced two novel methods for novelty detection in time series through classification. Both methods are based on the idea of using an envelope in order to define the regions or normality and novelty. One method is based on the use of negative samples in the training phase whereas the other does not depend on negative samples. A number of experiments with different time series and different classifiers were performed and their results reported and discussed. The results have shown that the second method gives better results because it does not depend on negative samples for training.

The performance of the classifier is, of course, very important for the overall performance of classification-based methods for novelty detection in time series. This has motivated the investigation of techniques for improving the performance of DDA, a constructive training algorithm for RBF networks. Four different methods were introduced and investigated in this thesis for this purpose. In order to assess the performance of the proposed methods, several experiments using six benchmark classification datasets from the UCI machine learning repository were conducted. The results obtained have shown that, indeed, the proposed methods considerably improve RBF-DDA classification performance in the datasets considered. The improved constructive RBF classifiers in-

troduced here were shown to outperform MLPs and AdaBoost in the datasets considered and to achieve similar performance to k-NN and support vector machines.

Two of the methods introduced in this thesis for improving RBF-DDA generalization performance were also applied to the problem of detecting novelties in time series via classification of windows from the time series. Experiments conducted using a number of real-world time series have shown that these methods are also valuable for improving performance of novelty detection in time series.

7.1 CONTRIBUTIONS

The main contributions of this thesis are summarized below:

- **First:** This thesis has introduced the idea of using absolute thresholds instead of relative thresholds and robust confidence intervals in order to improve forecasting-based time series novelty detection. The main problem with forecasting-based time series novelty detection is the selection of adequate values for the thresholds to be used to detect novelties. The use of robust confidence intervals introduced here is a suitable way to define these thresholds.

- **Second:** A novel method for time series novelty detection based on classification was proposed. The method classifies time series windows into normal or novelty. Training is carried out by using an augmented training set formed by artificially generating both normal and novelty random patterns and adding them to the original training set.

  The method is based on the idea of using an *envelope* to define normal and novelty regions. Normal random patterns have all data points inside an envelope around the time series windows whereas novelty patterns are outside this envelope. Novelty random patterns are also called *negative samples*. Normal random patterns are added in order to improve performance when the method is used for novelty detection in short time series.

  A number of different neural networks were used as classifiers, including MLPs, RBF-DDA, committee machines formed by these networks and SVMs. The proposed method was evaluated on some real-world time series. Performance of the different classifiers were compared.

  In contrast to previous classification-based time series novelty detection methods, the method introduced in this work is devoted to short time series and is designed to detect more subtle novelties. This is important for fraud detection applications. The main disadvantage of this method is that it needs negative samples and therefore its performance depends on the number of negative samples used in training.

- **Third:** A novel classification-based novelty detection method that does not need negative samples was also proposed in this thesis. The method is also based on the idea of the envelope and on the DDA algorithm for constructive training of RBF neural networks. The proposed method was evaluated using some real-world

time series. The results show that it outperforms the method based on negative samples proposed here. It is also shown that the method's performance is almost independent on test set size. Conversely, performance of the method based on negative samples degrades when the test set increases because it depends on negative samples for training.

- **Fourth:** The work reported here presents contributions also in the area of classification by constructive RBF neural networks. In this regard, four methods for boosting RBF-DDA performance were proposed. The first method introduced in this thesis improves generalization performance of RBF-DDA by selection of the value of parameter θ^-. This method is shown to considerably improve RBF-DDA performance in several classification problems, yet it builds much larger networks than default RBF-DDA. Three alternative methods were introduced in this thesis to tackle this problem.

  The second method for improving RBF-DDA adjusts the values of the network weights in order to improve performance. In both this and the previous method, a validation set is used to help avoiding overfitting. The third method introduced in this thesis combines a data reduction algorithm with θ^- selection for improving performance of constructive RBF networks without increasing the size of the networks. Finally, the fourth method, which we call RBF-DDA-SP, combines selective pruning of neurons considered redundant after the networks is built with θ^- selection [OMM05a].

  A number of simulations carried out using six benchmark classification datasets from the UCI machine learning repository were conducted and the results were reported in this thesis. We have compared the methods regarding classification performance, and space and time complexities. Comparisons with a number of other classifiers, including MLPs, k-NN, RBF networks with a more traditional training method, AdaBoost and SVM were also provided. The simulations of the proposed methods were carried out using both the *holdout* and the *cross-validation* methods. A number of Student paired t-test were also conducted in order to verify whether the difference in performance of the classifiers were statistically significant.

  The results of the experiments reported on this thesis have shown that the method based on θ^- considerably improved performance (in comparison with the DDA trained with default parameters) for all datasets considered. Moreover, the method has outperformed RBF trained with a more traditional method, MLPs and AdaBoost on these datasets. The proposed method achieved performance comparable to k-NN and SVM in the datasets considered. It offers the advantage of build classifiers with smaller space complexity than k-NN and much faster to train than SVMs.

  The results of the experiments have also shown that the three methods proposed for improving RBF-DDA performance without increasing the size of the networks also achieved their goals. The three methods were able to produce networks with generalization performance close to the first method (RBF-DDA with θ^- selection),

yet generating much smaller networks. The second method, which integrated data reduction with θ^- selection, was the best regarding both generalization performance and size of the networks generated. This method was able to produce networks with approximately the same size as those produce by the default RBF-DDA, but having the generalization performance of the method based solely on θ^- selection.

Two of the methods introduced in this thesis for improving RBF-DDA, namely RBF-DDA with θ^- selection and RBF-DDA-SP, were also applied in conjunction with classification-based method for novelty detection in time series using negative samples. The simulation results have shown that the proposed methods considerably improves novelty detection performance on the time series considered.

7.2 FUTURE WORK

The envelope - used as a basis for the methods for novelty detection in time series based on classification proposed here - are needed to tackle the uncertainty associated with the future behavior of the time series under analysis. In the experiments reported it was assumed that a normal pattern (time series windows) had all data points deviating at most 10% from the time series original data point (above or below). That is, envelopes of width 0.2 were used. The question is: what should be the width of the envelope in practical applications? Another question which arises is: the envelope width should be constant or variable?

Further research is in need in order to try to answer these questions. Initially, the influence of the envelope widths on the performance of the methods should be studied. A possible approach for the automatic determination of the envelope width is to use the width which optimizes performance on a validation set. In practice, the width should have an upper limit because fraud detection applications, for example, are interested in detecting small deviations from the true value.

The methods proposed here were designed for tackling short time series, which are common in important applications such as accountancy and payroll auditing. The forecasting methods and the classification method with negative samples have been tested with both seasonal time series and more general ones such as the chaotic Mackey-Glass time series. However, the classification-based method has been tested only on short seasonal time series with period 12. It is important to compare this method with the one based on negative samples also on non-seasonal time series, which can also appear in practice.

The ideas and methods proposed in this work can be further explored in various ways. Some possibilities related to classification-based novelty detection in time series are:

- The DDA algorithm has also been adapted for training probabilistic neural networks (PNNs) [BD98]. These networks directly provide a probabilistic interpretation of their outputs, which can be important for some applications. A possible investigation consists in using PNNs in conjunction with both classification-based novelty detection methods proposed in this work. In this case, the novelty detection method would provide the probability that a given time series window be novelty.

- The classifiers used in conjunction with the proposed methods are all focused TLFNs (time lagged feedforward networks) and therefore have limited temporal processing abilities. A further improvement in the proposed classification methods would be to use *distributed* TLFNs, such as the *time delay radial basis functions networks – TDRBFs* trained with the DDA [Ber94]. TDRBF was designed for temporal processing and has the advantage of requiring less parameters than TDNN. Furthermore, TDRBF is faster to train.

- The classification-based novelty detection method without negative samples proposed in this work has performed better that the method based on these samples. The experiments reported in this work regarding the former method have used window size $w = 12$. An investigation should be carried out in order to analyze the impact of the window size on the performance of the method.

- Finally, a possible alternative for improving novelty detection in time series would be to combine forecasting and classification-based methods. A classification-based method could be used to detect novelty in a time series windows and then the forecasting based method would be used to identity the particular data point(s) at which novelty happened. This could improve confidence in the output of the novelty detection system.

The methods proposed for improving performance of RBF-DDA can be further investigated in the following directions:

- The method based on θ^- selection has lead to a suboptimal network for one of the partitionings of the *soybean* dataset. The reason can be the use of simple validation. The use of $m - fold$ cross-validation in conjunction with this method can be investigated as a method to refine θ^- selection for small datasets.

- The weights adjustment of method two could be applied to the third method, which integrated a data reduction algorithm with θ^- selection. The idea is to adjust the weights of the network generated by the integrated method in order to attempt to further improve generalization performance.

- The weights adjustment of method two could also be applied to RBF-DDA-SP, with the same purpose outlined above.

- An interesting research direction consists in investigating if the methods proposed here can also improve performance in PNNs trained by the DDA.

- The DDA algorithm has also been adapted to train networks consisting of rectangular instead of radial basis functions [BH95]. The adaption of the methods proposed here for this kind of network can also be a research direction.

- Finally, another possible investigation consists in the adaptation of the optimization methods proposed here for training TDRBFs. This can lead to improvements in a number of applications of these networks, which include phoneme recognition and gesture recognition.

7.3 FINAL REMARKS

The work reported in this thesis has studied the problem of novelty detection in time series, with particular emphasis on methods for application in fraud detection in financial systems. The investigations carried out have contributed with both forecasting and classification-based methods for this problem. It is hoped that the methods proposed here can be of practical applications to real-world problems such as to build systems for automatic fraud detection in accountancy and payrolls.

In addition, this work has proposed methods that improved classification performance of constructive RBF neural networks in a number of image recognition tasks and in a soybean diseases dataset. Constructive neural networks have a good practical appeal because they overcome the, sometimes difficult and/or cumbersome, task of defining the network architecture. It is expected that the work proposed here lead to improved and, at the same time, easier to build classifiers for a number of different datasets.

APPENDIX A

METHODS FOR IMPROVING RBF-DDA: RESULTS OF 10FOLD CROSS-VALIDATION EXPERIMENTS

This appendix presents in more detail the cross-validation experiments of subsection 4.3.7, carried out in order to compare two of the methods introduced in this thesis for improving RBF-DDA performance. We have carried out 10-fold cross-validation simulations in order to compare the method based on θ^- (introduced in subsection 4.2.1) selection with the integrated method for constructive training of RBFNs (introduced in subsection 4.2.3). These methods are also compared to RBF-DDA trained with default parameters and to k-NN.

In order to compare two classifiers using the Student paired t-test, we first perform 10-fold cross-validation using the same training and test sets for each of the classifiers. Subsequently, we compute the collection of test errors, $\{x_i\}$ for the first classifier and $\{y_i\}$ for the second one. Then, we compute $d_i = x_i - y_i$, which is used to compute t as follows

$$t = \frac{\bar{d}}{\sqrt{s_d^2/k}} \tag{1.1}$$

where $\bar{d}$ is the mean of d_i, s_d is the standard deviation of d_i and k is the number of folds. In our experiments, we have performed 10-fold cross-validation, thus $k = 10$. Moreover, we employ 95% confidence level. For this confidence level, the t-student distribution table with $k - 1 = 9$ gives $z = 2.262$. Hence, for 95% confidence level, the results produced by two classifiers being compared will be considered statistically different only if $t > z$ or $t < -z$.

Tables A.1, A.3, A.5, A.7 and A.9 show the classification errors for each fold and the mean and standard deviation of the cross-validation error for each classifier (in the last line), for the datasets *optdigits*, *pendigits*, *letter*, *satimage* and *segment*, respectively. For all datasets, except *segment*, k-NN experiments were carried out with $k = 3$. For *segment*, these experiments used $k = 1$, $k = 2$ and $k = 3$. Table A.9 shows only k-NN results with $k = 1$, since this value achieved the best generalization performance. The results of these tables have been summarized and presented also in table 4.25 of subsection 4.3.7.

The results of these tables show that both methods introduced in chapter 4 considerably outperform RBF-DDA trained with default parameters. Moreover, they achieve performance similar to k-NN in all datasets.

Tables A.2, A.4, A.6, A.8 and A.10 show results of hypothesis tests (Student paired t-test) intended to compare the classifiers considered here in pairs for the datasets *optdigits*, *pendigits*, *letter*, *satimage* and *segment*, respectively.

k-NN	RBF-DDA (default)	RBF-DDA (θ^- selection)	Integrated method (subsection 4.2.3)
3.74%	9.96%	2.14%	2.31%
0.89%	7.65%	0.53%	0.36%
1.25%	9.43%	1.07%	1.42%
2.85%	9.61%	1.96%	1.25%
1.96%	8.36%	1.60%	1.25%
1.07%	9.96%	1.42%	1.42%
3.91%	7.83%	2.31%	1.96%
2.49%	10.32%	2.67%	1.78%
2.85%	9.96%	3.02%	2.14%
1.78%	8.01%	1.96%	1.42%
2.28% (1.07%)	9.11% (1.03%)	1.87% (0.74%)	1.53% (0.56%)

Table A.1. 10-fold cross validation errors for *optdigits*

k-NN × DDA (def.)	k-NN × DDA (θ^-) sel.	k-NN × int. met.	DDA (def.) × DDA (θ^-) sel.	DDA (def.) × int. met.	DDA (θ^-) sel. × int. met.
$t = -15.93$	$t = 1.79$	$t = 3.21$	$t = 23.81$	$t = 27.35$	$t = 2.49$
diff.	equal	**diff.**	**diff.**	**diff.**	**diff.**

Table A.2. Hypothesis tests for *optdigits*

The last line of the each table provides the final results of each comparison. In these lines, *equal* means that the difference in performance of the classifiers in the dataset is not statistically significant. Conversely, *diff.* means that the difference is statistically significant.

The results of these tables show that the difference between k-NN and RBF-DDA with default parameters is statistically significant for all datasets considered, yet the difference between k-NN and RBF-DDA with θ^- selection is not significant for none of the datasets. These tables also show that the differences in performance between each of the methods proposed for improving RBF-DDA and the default RBF-DDA are statistically significant for all datasets. The comparison of k-NN to the integrated method for improving RBF-DDA has shown that the difference in the results is statistically significant only for the datasets *optdigits*, *pendigits* and *satimage*. On the other hand, RBF-DDA with θ^- selection and the integrated method produced results with a difference statistically significant only for the datasets *optdigits* and *satimage*.

k-NN	RBF-DDA (default)	RBF-DDA (θ^- selection)	Integrated method (subsection 4.2.3)
0.82%	4.64%	1.00%	1.00%
0.45%	3.55%	0.55%	1.64%
0.55%	3.00%	0.36%	0.82%
0.64%	5.28%	0.82%	1.00%
0.91%	4.09%	1.18%	1.46%
0.73%	3.55%	0.73%	0.82%
0.27%	4.00%	0.73%	0.64%
1.46%	6.37%	1.09%	1.27%
0.73%	5.10%	0.73%	0.73%
1.09%	4.90%	1.18%	1.09%
0.77% (0.34%)	4.45% (1.01%)	0.84% (0.27%)	1.05% (0.32%)

Table A.3. 10-fold cross validation errors for *pendigits*

k-NN × DDA (def.)	k-NN × DDA (θ^-) sel.	k-NN × int. met.	DDA (def.) × DDA (θ^-) sel.	DDA (def.) × int. met.	DDA (θ^-) sel. × int. met.
$t = -14.52$	$t = -0.98$	$t = -2.31$	$t = 13.29$	$t = 10.42$	$t = -1.88$
diff.	equal	**diff.**	**diff.**	**diff.**	equal

Table A.4. Hypothesis tests for *pendigits*

k-NN	RBF-DDA (default)	RBF-DDA (θ^- selection)	Integrated method (subsection 4.2.3)
4.75%	15.05%	4.95%	4.95%
5.30%	16.65%	5.20%	5.15%
4.75%	13.25%	3.90%	4.85%
4.75%	14.00%	4.20%	4.15%
4.15%	13.75%	4.20%	4.05%
5.30%	14.80%	5.60%	5.50%
5.20%	14.00%	5.05%	5.20%
4.05%	13.40%	3.95%	4.00%
4.05%	15.20%	4.80%	4.35%
5.05%	14.90%	4.95%	4.95%
4.74% (0.50%)	14.50% (1.03%)	4.68% (0.58%)	4.72% (0.53%)

Table A.5. 10-fold cross validation errors for *letter*

k-NN × DDA (def.)	k-NN × DDA (θ^-) sel.	k-NN × int. met.	DDA (def.) × DDA (θ^-) sel.	DDA (def.) × int. met.	DDA (θ^-) sel. × int. met.
$t = -33.19$	$t = 0.40$	$t = 0.25$	$t = 43.15$	$t = 33.93$	$t = -0.31$
diff.	equal	equal	**diff.**	**diff.**	equal

Table A.6. Hypothesis tests for *letter*

k-NN	RBF-DDA (default)	RBF-DDA (θ^- selection)	Integrated method (subsection 4.2.3)
11.20%	11.51%	7.93%	7.31%
10.58%	9.64%	8.71%	8.86%
17.11%	22.08%	17.73%	17.26%
16.49%	20.53%	15.86%	15.55%
9.49%	13.69%	8.40%	7.78%
7.31%	15.09%	9.49%	8.86%
11.82%	18.35%	9.64%	9.95%
7.31%	8.24%	6.53%	6.22%
13.22%	17.42%	12.44%	12.29%
8.49%	19.14%	7.87%	6.94%
11.30% (3.47%)	15.57% (4.71%)	10.46% (3.70%)	10.10% (3.76%)

Table A.7. 10-fold cross validation errors for *satimage*

k-NN × DDA (def.)	k-NN × DDA (θ^-) sel.	k-NN × int. met.	DDA (def.) × DDA (θ^-) sel.	DDA (def.) × int. met.	DDA (θ^-) sel. × int. met.
$t = -3.81$	$t = 1.78$	$t = 2.69$	$t = 5.31$	$t = 5.44$	$t = 2.98$
diff.	equal	**diff.**	**diff.**	**diff.**	**diff.**

Table A.8. Hypothesis tests for *satimage*

k-NN	RBF-DDA (default)	RBF-DDA (θ^- selection)	Integrated method (subsection 4.2.3)
3.90%	16.45%	0.00%	0.43%
6.93%	42.42%	5.19%	6.93%
17.32%	43.72%	16.88%	13.42%
2.16%	25.97%	4.33%	6.06%
0.00%	2.16%	0.00%	6.06%
15.58%	55.84%	17.75%	19.05%
16.45%	25.54%	22.08%	23.38%
24.24%	16.02%	23.38%	22.51%
12.99%	59.31%	14.72%	15.15%
4.33%	37.23%	16.02%	10.82%
10.39% (8.01%)	32.47% (18.41%)	12.04% (8.85%)	12.38% (7.69%)

Table A.9. 10-fold cross validation errors for *segment*

k-NN × DDA (def.)	k-NN × DDA (θ^-) sel.	k-NN × int. met.	DDA (def.) × DDA (θ^-) sel.	DDA (def.) × int. met.	DDA (θ^-) sel. × int. met.
$t = -3.95$	$t = -1.19$	$t = -1.55$	$t = 3.78$	$t = 3.56$	$t = -0.36$
diff.	equal	equal	**diff.**	**diff.**	equal

Table A.10. Hypothesis tests for *segment*

APPENDIX B

CORRELOGRAMS OF TIME SERIES USED IN EXPERIMENTS

In many cases, pre-processing a time series before building a neural network model for forecasting can be very important for improving forecasting performance. For example, if an MLP is used for forecasting and the time series is non-stationary, it is important to differentiate the series before training the network, as discussed in chapter 2. The correlogram is a visual tool used to identify trends, seasonality and stationarity [Cha89].

The correlogram is a graphics that depicts the values of the autocorrelations of order 1 to $n-1$, where n is the size of the series. The autocorrelation of order k measures the influence of a value at time $t+k$ on the past value k slots before, that is, at time t. Given a time series $x_1, \cdots x_n$, the autocorrelation of order k (or lag k), r_k is given by:

$$r_k = \frac{\sum_{t=1}^{n-k}(x_t - \overline{x}) \times (x_{t+k} - \overline{x})}{\sum_{t=1}^{n}(x_t - \overline{x})^2} \tag{2.1}$$

where $\overline{x}$ is the mean over all values of the time series.

Figures B.1, B.2, B.3, and B.4 depict the correlograms of time series 1, 2, 3 and 4, respectively. These time series are shown in figures 6.2 and 5.5. These correlograms confirm that these series are seasonal and non-stationary, because they present periodic peaks and have significant values even for large values of the lag k. Notice that the periodic peaks appear at lags $k = 12$, $k = 24$, $k = 36$, and so on. This means that the series are seasonal with period 12, as expected from the visual examination of the respective time series graphics.

The correlograms of the *earning 2* and of the *empper* time series are shown in figures B.5, and B.6, respectively. These series are depicted in figures 6.4 and 6.3. Their correlograms show that these are non-stationary and non-seasonal time series because they exhibit many significant values (for high values of k) and the values do not alternate much between positive and negative values. In fact, for both cases the correlogram start with positive values up to a certain lag and then the autocorrelation values are all negative until $k = n - 1$. Finally, the correlogram of the chaotic Mackey-Glass time series (shown in figure 5.11) is depicted in figure B.7. The correlogram of this time series shows that it exhibits long range dependence, a characteristic of chaotic time series.

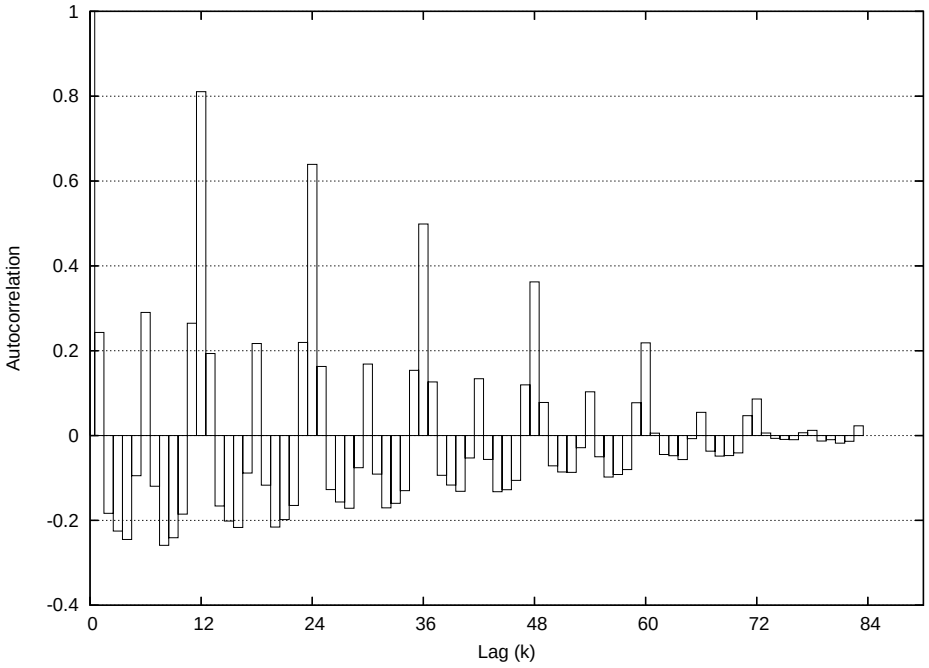


Figure B.1. Correlogram of time series 1

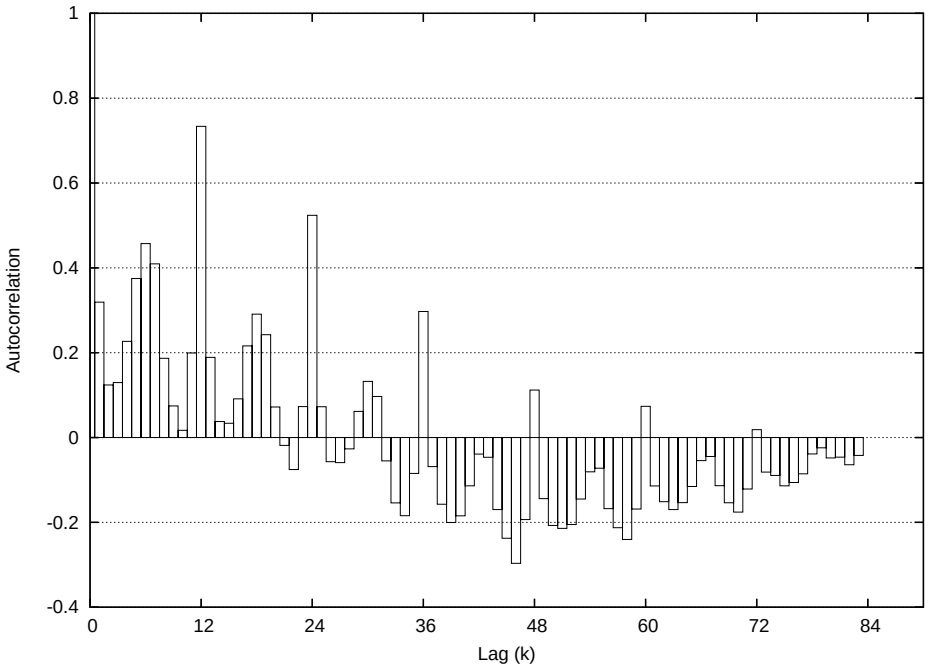


Figure B.2. Correlogram of time series 2

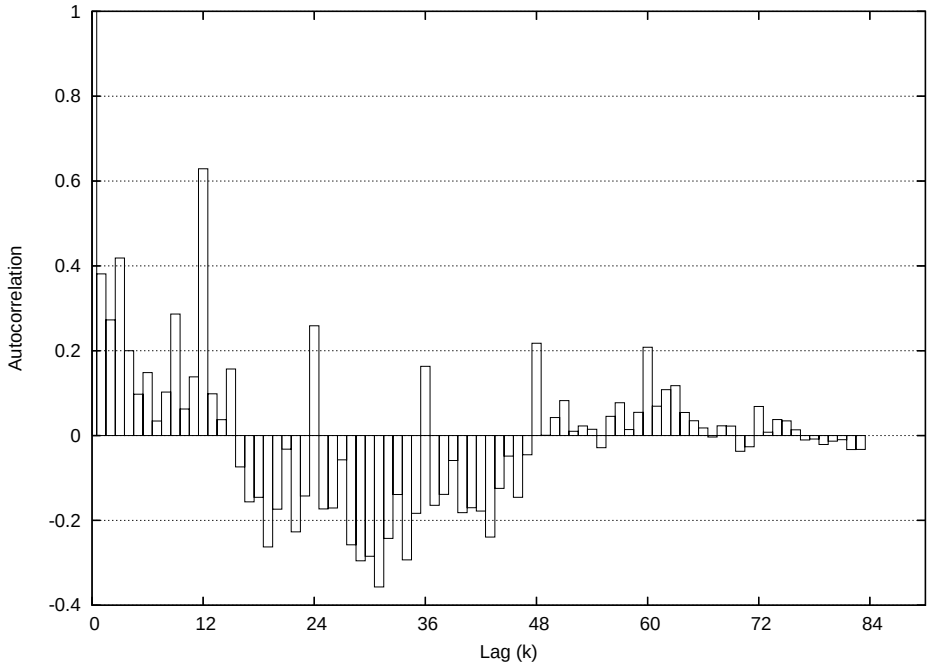


Figure B.3. Correlogram of time series 3

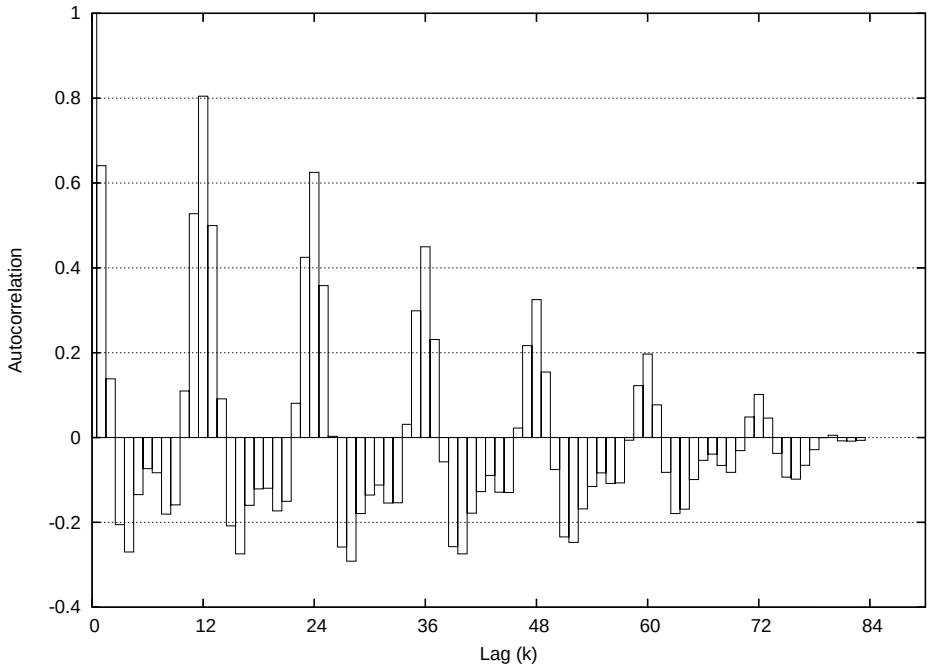


Figure B.4. Correlogram of time series 4

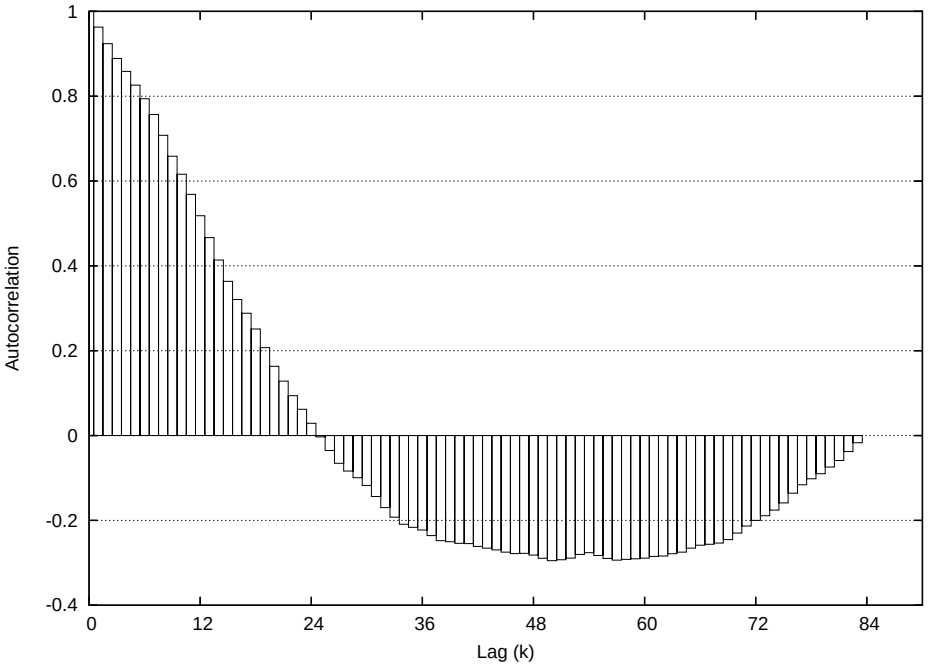


Figure B.5. Correlogram of the *earning 2* time series

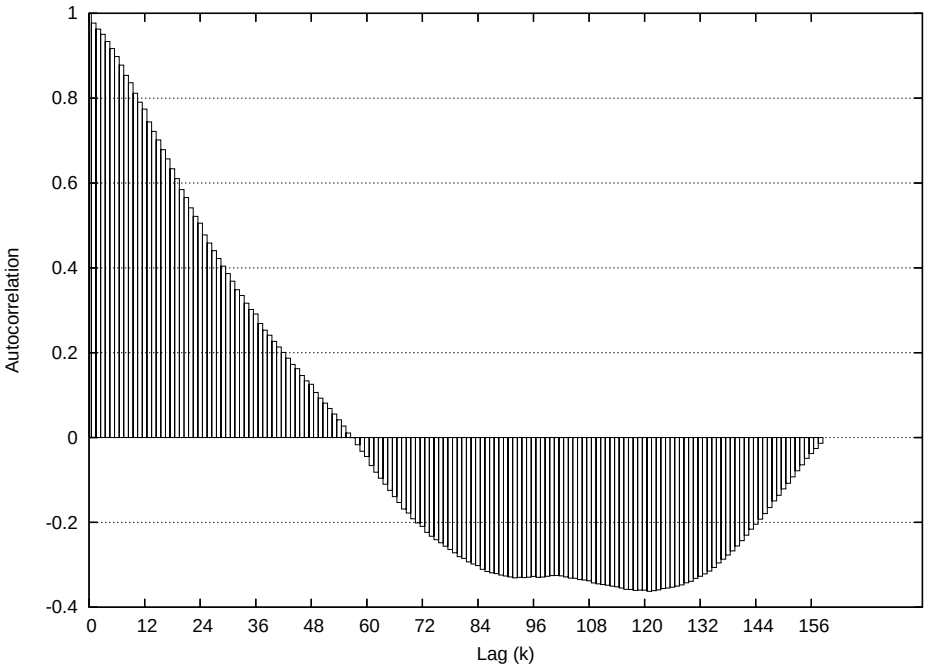


Figure B.6. Correlogram of the *empper* time series

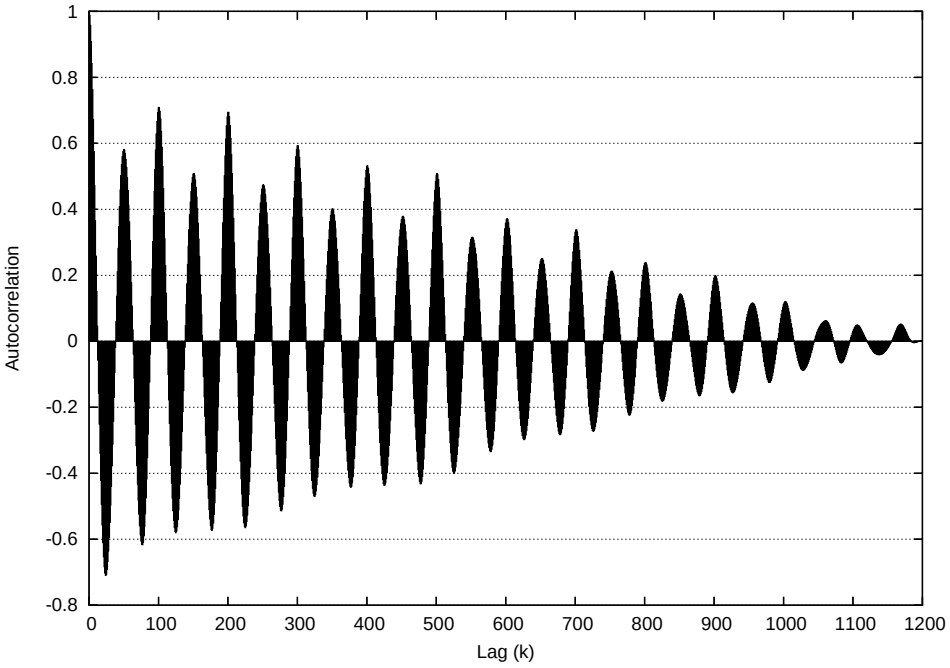


Figure B.7. Correlogram of the chaotic Mackey-Glass time series

BIBLIOGRAPHY

[A.03] V. David Sanchez A. Advanced support vector machines and kernel methods. *Neurocomputing*, 55:5–20, 2003.

[ABK+00] S. Albrecht, J. Busch, M. Kloppenburg, F. Metze, and P. Tavan. Generalised radial basis function networks for classification and novelty detection: Self-organisation of optimal bayesian decision. *Neural Networks*, 13:1075–1093, 2000.

[acl] ACL. http://www.acl.com.

[ASS00] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[A.W02] A.Webb. *Statistical Pattern Recognition*. Wiley, second edition, 2002.

[BBGC99] R. Bogacz, M. W. Brown, and C. Giraud-Carrier. High capacity neural networks for familiarity discrimination. In *Proc. of ICANN'99*, pages 773–778, Edinburgh, UK, 1999.

[BD95] Michael R. Berthold and Jay Diamond. Boosting the performance of RBF networks with dynamic decay adjustment. In G. Tesauro et al, editor, *Advances in Neural Information Processing*, volume 7, pages 521–528. MIT Press, 1995.

[BD98] M. Berthold and J. Diamond. Constructive training of probabilistic neural networks. *Neurocomputing*, 19:167–183, 1998.

[Ber94] M. R. Berthold. A time delay radial basis function network for phoneme recognition. In *Proc. of the IEEE International Conference on Neural Networks*, volume 7, pages 4470–4473, 1994.

[BGG97] C. S. Burrus, R. A. Gopinath, and H. Guo. *Introduction to Wavelets and Wavelet Transforms – A Primer*. Prentice-Hall, 1997.

[BH69] A. E. Bryson and Y. Ho. *Applied Optimal Control*. Blaisdell, 1969.

[BH95] M. Berthold and K-P Huber. From radial to rectangular basis functions: A new approach for rule learning from large datasets. Technical Report Internal Report 15-95, University of Karlsruhe, 1995.

[Bis94] C. Bishop. Novelty detection and neural network validation. *IEE Proceedings on Vision, Image and Signal Processing*, 141(4):217–222, 1994.

[BJ76] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA, USA, 1976.

[BLC00] A. P. Braga, T. B. Ludermir, and A. C. P. L. F. Carvalho. *Redes Neurais Artificiais – Teoria e Aplicações*. LTC, 2000.

[BM98] C. Blake and C. Merz. UCI repository of machine learning databases. Available from [http://www.ics.uci.edu/~mlearn/MLRepository.html], 1998.

[BM01] J. Bruce and R. Miikkulainen. Evolving populations of expert neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 251–257. Kaufmann, 2001.

[CH01] P. Crook and G. Hayes. A robot implementation of a biologically inspired method for novelty detection. In *Proc. Towards Intelligent Mobile Robots Conference*, Manchester, UK, 2001.

[Cha89] C. Chatfield. *The Analysis of Time Series – An Introduction*. Chapman & Hall, fourth edition, 1989.

[CHBK04] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, 2004.

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[CLC03] L. J. Cao, H. P. Lee, and W. K. Chong. Modified support vector novelty detector using training data with outliers. *Pattern Recognition Letters*, 24(Issue 14):2479–2487, October 2003.

[CST00] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[CV95] C. Cortes and V. Vapnik. Support-vector network. *Machine Learning*, pages 273–297, 1995.

[Cyb88] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, 1988.

[Cyb89] G. Cybenko. Approximation by superpositions of sigmoid functions. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[DAO97] D. Dasgupta and N. Attoh-Okine. Immunity-based systems: A survey. In *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 369–374, 1997.

[Das97] D. Dasgupta. Artificial neural networks and artificial immune systems: Similarities and differences. In *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 873–878, 1997.

[DF96] D. Dasgupta and S. Forrest. Novelty detection in time series data using ideas from immunology. In *Proc. of the 5th International Conference on Intelligent Systems*, 1996.

[DFH96] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: Algorithms, analysis, and implications. In *Prof. of IEEE Symposium on Reseach in Security and Privacy*, pages 110–119, 1996.

[DG01] D. Dasgupta and F. Gonzalez. Evolving complex fuzzy classifier rules using a linear genetic representation. In *Proc. of International Conference on Genetic and Evolutionary Computation (GECCO'2001)*, San Francisco, CA, USA, Jul 2001. Morgan Kaufmann.

[DG02] D. Dasgupta and F. Gonzalez. An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, 6(3):1081–1088, June 2002.

[DHS00] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley-Interscience, second edition, 2000.

[DI02] C.P. Diehl and J.B. Hampshire II. Real-time object classification and novelty detection for collaborative vide surveillance. In *Proc. of IEEE IJCNN Conference*, volume 3, pages 2620–2625, 2002.

[DS92] M. R. W. Dawson and D. P. Schopflocher. Modifying the generalized delta rule to train networks of non-monotonic processors for pattern classification. *Connection Science*, 4(1):19–31, 1992.

[EKT00] V. Emamian, M. Kaveh, and A. H. Tewfik. Robust clustering of acoustic emission signals using the Kohonen network. In *Proc. of IEEE ICASSP Conference*, volume 6, pages 3891–3894, Istanbul, 2000.

[Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[Fla93] G. W. Flake. *Nonmonotonic Activation Functions in Multilayer Perceptrons.* PhD thesis, Department of Computer Science, University of Maryland, 1993.

[Fle87] R. Fletcher. *Practical methods of optimization.* Wiley, 1987.

[FPAC94] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Prof. of IEEE Symposium on Reseach in Security and Privacy*, pages 202–212, May 1994.

[FRTT94] S. Fredrickson, S. Roberts, N. Townsend, and L. Tarassenko. Speaker identification using networks of radial basis functions. In *Proc. of the VII European Signal Processing Conference*, pages 812–815, Edinburgh, UK, 1994.

[FS91] P. W. Frey and D. J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, mar 1991.

[GD02] F. Gonzalez and D. Dasgupta. Neuro-immune and self-organizing map approaches to anomaly detection: A comparison. In *Proc. of the 1st International Conference on Artificial Immune Systems*, pages 203–211, 2002.

[GDK02] F. Gonzalez, D. Dasgupta, and R. Kozma. Combining negative selection and classification techniques for anomaly detection. In *Proc. of IEEE Congress on Evolutionary Computation*, pages 705–710, 2002.

[Hay98] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1998.

[HB98] A. Howell and H. Buxton. Learning identity with radial basis function networks. *Neurocomputing*, 20:15–34, 1998.

[HB02] A. Howell and H. Buxton. Time-delay rbf networks for attentional frames in visually mediated interaction. *Neural Processing Letters*, 15:197–211, 2002.

[HCL04] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. *A Practical Guide to Support Vector Classification*, 2004. Available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[HF00] S. A. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation*, 8(4):443–473, 2000.

[HM98] M. Habib and A. Mourad. Architecture and design methodology of the rbf-dda neural network. In *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'98)*, volume 3, pages 199–202, Monterey, CA, USA, may 1998.

[Hud92] M. J. Hudak. RCE classifiers: Theory and practice. *Cybernetics and Systems*, 23:483–515, 1992.

[ide] IDEA - Interactive Data Extraction and Analysis. http://www.audimation.com.

[KA00] C. Kaynak and E. Alpaydin. Multistage cascading of multiple classifiers: One man's noise in another man's data. In *Proc. of the 17th International Conference on Machine Learning*, pages 455–462, 2000.

[KGC00] S. Kumar, J. Ghosh, and M. Crawford. A bayesian pairwise classifier for character recognition. In Nabeel Mursheed, editor, *Cognitive and Neural Models for Word Recognition and Document Processing*. World Scientific Press, 2000.

[KLC02] E. Keogh, S. Lonardi, and W. Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proc. ACM Knowledge Discovery and Data Mining - SIGKDD'02*, pages 550–556, 2002.

[KLSK96] T. Koskela, M. Lehtokangas, J. Saarinen, and K. Kaski. Time series prediction with multi-layer perceptron, FIR and Elman neural networks. In *Proceedings of the World Congress on Neural Networks*, pages 491–496, San Diego, USA, 1996.

[Kos00] Eija Koskivaara. Artificial neural network models for predicting patterns in auditing monthly balances. *Journal of Operational Research Society*, 51(9):1060–1069, Sept 2000.

[Kos03] Eija Koskivaara. Artificial neural networks in auditing: State of the art. Technical Report TUCS-TR-509, February 2003.

[KS02] S. Kurjakovic and A. Svennberg. Implementing av rbf i vhdl. Diploma Thesis, Kungl Tekniska Hogskolan (Royal Institute of Technology), Stockholms Centrum for Fysik, Astronomi, Bioteknik (in Swedish), 2002. `http://www.particle.kth.se/~lindblad/DiplomaWorks.htm`.

[LCL03] E. Lacerda, A. C. P. L. F. Carvalho, and T. Ludermir. Model selection via genetic algorithms for rbf networks. *Journal of Intelligent and Fuzzy Systems*, 13(2-4):111–122, 2003.

[LeC85] Y. LeCun. A learning procedure for assymetric threshold network. In *Proceedings of Cognitiva 85*, pages 599–604, 1985.

[LK89] A. Linden and J. Kindermann. Inversion of multilayer nets. In *Proc. of International Joint Conference in Neural Networs (IJCNN'89)*, volume 2, pages 425–430, 1989.

[LSP+96] Th. Lindblad, G. Székely, M. L. Padgett, A. Eide, and C. S. Lindsey. Implementing the dynamic decay adjustment algorithm in a cnaps parallel computer system. *Nucl. Instr. Meth.*, A381:502–507, 1996.

[LV02] K. Labib and R. Vemuri. NSOM: A real-time network-based intrusion detection system using self-organizing maps. Technical report, Dept. of Applied Science, University of California, Davis, 2002.

[Mac92a] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[Mac92b] D. J. C. MacKay. A practical bayesian framework for backprop networks. *Neural Computation*, 4(3):448–472, 1992.

[Mar01] S. Marsland. *On Line Novelty Detection through Self-Organization, with Application to Inspection Robotics*. PhD thesis, Department of Computer Science, University of Manchester, 2001.

[Mar03] S. Marsland. Novelty detection in learning systems. *Neural Computing Surveys*, 3:157–195, 2003.

[Mas95] Timothy Masters. *Neural, Novel & Hybrid Algorithms for Time Series Prediction*. John Wiley and Sons, Inc., 1995.

[May79] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, 1979.

[MC80] R. S. Michalski and R. L. Chilausky. Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, 4(2):125–161, 1980.

[MLH03] D. Meyer, F. Leisch, and K. Hornik. The support vector machine under test. *Neurocomputing*, 55:169–186, 2003.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP03] J. Ma and S. Perkins. Time series novelty detection using one-class support vector machines. In *Proc. of the Int. Joint Conference on Neural Networks (IJCNN'2003)*, volume 3, pages 1741–1745, 2003.

[MS03a] M. Markou and S. Singh. Novelty detection: A review, part i: Statistical approaches. *Signal Processing*, 83:2481–2497, 2003.

[MS03b] M. Markou and S. Singh. Novelty detection: A review, part ii: Neural network based approaches. *Signal Processing*, 83:2499–2521, 2003.

[MSN02] S. Marsland, J. Shapiro, and U. Nehmzow. A self-organizing networks that grows when required. *Neural Networks*, 15(8-9):1041–1058, 2002.

[MSR+97] K.-R. Muller, A. Smola, G. Ratsch, B. Scholkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *Proc. of International Conference on Neural Networks (ICANN'97)*, Lecture Notes in Computer Science, Vol. 1327, pages 999–1004. Springer-Verlag, 1997.

[MST94] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[Nab01] I.T. Nabney. *Netlab: Algorithms for Pattern Recognition*. Advances in Pattern Recognition. Springer, 2001.

[OABS03] A. L. I. Oliveira, G. Azevedo, A. Barros, and A. L. M. Santos. A neural network based system for payroll audit support (in portuguese). In *Proceeding of the IV Brazilian National Artificial Intelligence Meeting*, pages 487–496, 2003.

[OCHO03] Y.-Y. Ou, C.-Y. Chen, S.-C. Hwang, and Y.-J. Oyang. Expediting model selection for support vector machines based on data reduction. In *Proc. of IEEE Conference on Systems, Man and Cybernetics*, volume 1, pages 786–791, 2003.

[OMM05a] A. L. I. Oliveira, B. J. M. Melo, and S. R. L. Meira. Improving constructive training of RBF networks through selective pruning and model selection. *Neurocomputing*, 64C:537–541, 2005. Available at http://www.sciencedirect.com.

[OMM05b] A. L. I. Oliveira, B. J. M. Melo, and S. R. L. Meira. Integrated method for constructive training of radial basis functions networks. *IEE Electronics Letters*, 2005. In Press. Available at http://ieeexplore.ieee.com.

[OMNM04] A. L. I. Oliveira, B. J. M. Melo, F. B. L. Neto, and S. R. L. Meira. Combining data reduction and parameter selection for improving rbf-dda performance. In *Proc. of IX Ibero American Conference on Artificial Intelligence (IBERAMIA'2004)*, volume 3315 of *Lecture Notes in Computer Science*, pages 778–787, Berlin Heidelberg, 2004. Springer-Verlag.

[ONM03] A. L. I. Oliveira, F. B. L. Neto, and S. R. L. Meira. Novelty detection for short time series with neural networks. In A. Abraham, M. Köppen, and K. Franke, editors, *Design and Application of Hybrid Intelligent Systems*, volume 104 of *Frontiers in Artificial Intelligence and Applications*, pages 66–76. IOS Press, 2003.

[ONM04a] A. L. I. Oliveira, F. B. L. Neto, and S. R. L. Meira. Combining MLP and RBF neural networks for novelty detection in short time series. In *Proc. of Mexican International Conference on Artificial Intelligence*, volume 2972 of *Lecture Notes in Computer Science*, pages 844–853, Berlin Heidelberg, 2004. Springer-Verlag.

[ONM04b] A. L. I. Oliveira, F. B. L. Neto, and S. R. L. Meira. Improving novelty detection in short time series through RBF-DDA parameter adjustment. In *Proc. of International Joint Conference on Neural Networks (IJCNN'2004)*, volume 3, pages 2123–2128, Budapest, Hungary, 2004. IEEE Press.

[ONM04c] A. L. I. Oliveira, F. B. L. Neto, and S. R. L. Meira. Improving RBF-DDA performance on optical character recognition through parameter selection. In *Proc. of the 17th International Conference on Pattern Recognition (ICPR'2004)*, volume 4, pages 625–628, Cambridge,UK, 2004. IEEE Computer Society Press.

[ONM04d] A. L. I. Oliveira, F. B. L. Neto, and S. R. L. Meira. A method based on rbf-dda neural networks for improving novelty detection in time series. In *Proc. of the International FLAIRS Conference*, pages 670–675. AAAI Press, 2004.

[ONM04e] A. L. I. Oliveira, F. B. L. Neto, and S. R. L. Meira. Novelty detection in time series by neural networks forecasting with robust confidence intervals. In *Proc. Brazilian Symposium on Neural Networks (SBRN'2004)*. IEEE Computer Society Press, 2004.

[Pae04] J. Paetz. Reducing the number of neurons in radial basis function networks with dynamic decay adjustment. *Neurocomputing*, 62:79–91, 2004.

[Pre94] L. Prechelt. Proben1 – a set of neural networks benchmark problems and benchmarking rules. Technical Report 21/94, Universität Karlsruhe, Germany, 1994.

[RB93] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN 93)*, volume 1, pages 586–591, 1993.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[RK04] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[RLM98] J. Ryan, M. J. Lin, and R. Miikkulainen. Intrusion detection with neural networks. In M. Jordan et al., editor, *Advances in Neural Information Processing Systems*, pages 943–949. Cambridge, MA: MIT Press, 1998.

[RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. The MIT Press, 1986.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.*, 65:386–408, 1958.

[Ros62] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan Books, New York, 1962.

[RZF94] A. N. Refenes, A. Zapranis, and G. Francis. Stock performance modeling using neural networks: A comparative study with regression models. *Neural Networks*, 5:961–970, 1994.

[Sim02] D. Simon. Training radial basis neural networks with the extended kalman filter. *Neurocomputing*, 48:455–475, 2002.

[SM92] F. J. Smieja and H. Muhlenbein. Reflective modular neural network systems. Technical report, German National Research Centre for Computer Science, Germany, 1992.

[SM02] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural networks topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*. Morgan Kaufmann, 2002.

[SM04] S. Singh and M. Markou. An approach to novelty detection applied to the classification of image regions. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):396–406, April 2004.

[STC04] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[STZ00] C. Shahabi, X. Tian, and W. Zhao. TSA-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. In *Proc. of 12th International Conference on Scientific and Statistical Database Management*, pages 55–68, 2000.

[SWS+00] B. Scholkopf, R. C. Williamson, A. J. Smola, J. Shawe-Tayler, and J. Platt. Support vector method for novelty detection. *Neural Information Processing Systems*, pages 582–588, 2000.

[TD01] D. M. J. Tax and R. P. W. Duin. Uniform object generation for optimizing one-class classifiers. *Journal of Machine Learning Research*, 2:155–173, 2001.

[TK03] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Elsevier, second edition, 2003.

[ts-] Time series library. [http://www-personal.buseco.monash.edu.au/ hyndman/TSDL].

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, 1995.

[Vas95] Germano C. Vasconcelos. *An Investigation of Feedforward Neural Networks with Respect to the Detection of Spurious Patterns*. PhD thesis, University of Kent at Canterbury, 1995.

[VFB93] G. C. Vasconcelos, M. C. Fairhurst, and D. L. Bisset. The guard unit approach for rejecting patterns from untrained classes. In *Proc. 1993 World Congress on Neural Networks (WCNN'93)*, volume IV, pages 256–259, Portland, OR, USA, 1993.

[VFB94] G.C. Vasconcelos, M.C. Fairhurst, and D.L. Bisset. Recognizing novelty in classification tasks. In *NIPS Workshop on Novelty Detection, Adaptive Systems Monitoring*, 1994.

[VFB95] G. C. Vasconcelos, M. C. Fairhurst, and D. L. Bisset. Investigating feedforward neural networks with respect to the rejection of spurious patterns. *Pattern Recognition Letters*, 16(2):207–212, 1995.

[VHB02] H. Vassilakis, A. J. Howell, and H. Buxton. Comparison of feedforward (tdrbf) and generative (tdrgbn) network for gesture based control. In *Gesture and Sign Languages in Human-Computer Interaction - International Gesture Workshop, GW 2001*, Lecture Notes in Computer Science 2298, pages 317–321, 2002.

[Wan94] E. Wan. Time series prediction by using a connectionist network with internal delay line. In *Time Series Prediction. Forecasting the Future and Understanding the Past*. Addison-Wesley, 1994.

[WB01] Greg Welch and Gary Bishop. An introduction to the kalman filter. SIGGRAPH 2001 - Course 8, 2001. Available at http://www.cs.unc.edu/ welch/kalman.

[Wer74] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[WG94] A. S. Weigend and N. A. Gershenfeld, editors. *Time-series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, 1994.

[WNC03] J. Wang, P. Neskovic, and L. N. Cooper. Partitioning a feature space using a locally defined confidence measure. In *Proc. of International Conference on Artificial Neural Networks (ICANN'2003)*, 2003.

[WTT+04] X. G. Wang, Z. Tang, H. Tamura, M Ishii, and W. D. Sun. An improved backpropagation algorithm to avoid the local minima problem. *Neurocomputing*, 56:455–460, 2004.

[Yao99] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[YdSL02] A. Yamazaki, M.C.P. de Souto, and T.B. Ludermir. Optimization of neural network weights and architectures for odor recognition using simulated annealing. In *Proc. of the 2002 International Joint Conference on Neural Networks (IJCNN'2002)*, pages 547–552, 2002.

[YL97] X. Yao and Y. Liu. EPNet for chaotic time-series prediction. In *Selected Papers from the First Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'96)*, volume 1285, pages 146–156. Springer-Verlag, 1997.

[YLdS02a] A. Yamazaki, T.B. Ludermir, and M.C.P. de Souto. Global optimization methods for designing and training neural networks. In *Proc. of the VII Brazilian Symposium on Neural Networks*, pages 136–141, 2002.

[YLdS02b] A. Yamazaki, T.B. Ludermir, and M.C.P. de Souto. Simulated annealing and tabu search for optimization of neural networks. In *Proc. of the 26th Annual Conference of the Gesellschaft fur Klassifikation (GfKl)*, pages 510–520, 2002.

[YT00] J. Yao and C. L. Tan. A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, (34):79–98, 2000.

[Zel98] A. Zell. *SNNS - Stuttgart Neural Network Simulator, User Manual, Version 4.2*. University of Stuttgart and University of Tubingen, 1998.

[ZPH98] G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14:35–62, 1998.