UNIVERSITAT POLITÈCNICA DE CATALUNYA
DEPARTAMENT DE LLENGUATGES I SISTEMES INFORMÀTICS
PROGRAMA DE DOCTORAT EN INTEL·LIGÈNCIA ARTIFICIAL

Ph.D. Thesis Dissertation

Learning with Feed-forward Neural Networks: Three Schemes to Deal with the Bias/Variance Trade-off

Enrique Romero Merino
October 2004

Advisors: René Alquézar Mancho and Josep Maria Sopena i Sisquella

A mi madre, a mi hermana, a Ana y a Rodrigo

Agradecimientos

Me gustaría expresar mi gratitud a todos los que, de una u otra forma, han contribuído a la realización de esta tesis.

A mis directores, René Alquézar y Josep Maria Sopena, por sus numerosos consejos, dedicación y paciencia durante todos estos años. Pero sobre todo por su generosidad, esa cosa tan rara en estos tiempos que corren.

A Lluís Belanche este trabajo le debe mucho, aunque seguramente no se pueda expresar de manera demasiado concreta. Las frecuentes (y muchas veces inacabadas) discusiones han sido, y son, una ayuda impagable.

A Angela Nebot, por el buen humor con el que dirige nuestro grupo de investigación. Y por recordarme multitud de cosas básicas que olvido constantemente.

A Victoria Arranz, por permitir que le robara demasiado tiempo con demasiadas preguntas sobre gramática inglesa (los errores que todavía quedan en el documento son responsabilidad exclusivamente mía). Y por ser de mi pueblo. "En La Sequera muy burros y en Hontangas muy guasones, en Adrada los cabritos, Fuentemolinos...".

A Xavi Carreras, Lluís Màrquez, Horacio Rodríguez y Jordi Turmo, porque el café en compañía sabe mejor.

A Rafel Cases, por algunos privilegios no escritos.

A Gerard Escudero y Jordi Marco, por la absurda situación laboral que vivimos (la ambigüedad del tiempo verbal es intencionada). "De segar de los secanos ya vienen los segadores...". Mención especial para Conrado Martínez, por su sensibilidad en esta y otras cuestiones.

Al personal del laboratorio de cálculo y secretaría, por todas esas pequeñas cosas.

A las personas que han compartido los momentos más duros. A mi madre, por ser una persona extraordinaria. A mi hermana, por tantas lecciones de bondad y tenacidad. A Ana, por iluminar el camino con su maravillosa y serena sonrisa. A Rodrigo, por su mirada. Al resto de familia y amigos, por estar siempre ahí.

Un recuerdo muy especial para Charo, María Luisa y Nacho. Sobran los motivos.

Acknowledgements

I would like to thank two anonymous reviewers for helpful comments on a previous summary of this document.

This work was supported by Consejo Interministerial de Ciencia y Tecnología (CICYT), under projects TAP1999-0747 and DPI2002-03225.

Abstract

In terms of the Bias/Variance decomposition, very flexible (i.e., complex) Supervised Machine Learning systems may lead to unbiased estimators but with high variance. A rigid model, in contrast, may lead to small variance but high bias. There is a trade-off between the bias and variance contributions to the error, where the optimal performance is achieved.

In this work we present three schemes related to the control of the Bias/Variance decomposition for Feed-forward Neural Networks (FNNs) with the (sometimes modified) quadratic loss function:

1. An algorithm for sequential approximation with FNNs, named *Sequential Approximation with Optimal Coefficients and Interacting Frequencies* (*SAOCIF*). Most of the sequential approximations proposed in the literature select the new frequencies (the non-linear weights) guided by the approximation of the residue of the partial approximation. We propose a sequential algorithm where the new frequency is selected taking into account its interactions with the previously selected ones. The interactions are discovered by means of their optimal coefficients (the linear weights). A number of heuristics can be used to select the new frequencies. The aim is that the same level of approximation may be achieved with less hidden units than if we only try to match the residue as best as possible. In terms of the Bias/Variance decomposition, it will be possible to obtain simpler models with the same bias. The idea behind *SAOCIF* can be extended to approximation in Hilbert spaces, maintaining orthogonal-like properties. In this case, the importance of the interacting frequencies lies in the expectation of increasing the rate of approximation. Experimental results show that the idea of interacting frequencies allows to construct better approximations than matching the residue.

2. A study and comparison of different criteria to perform Feature Selection (FS) with Multi-Layer Perceptrons (MLPs) and the Sequential Backward Selection (SBS) procedure within the wrapper approach. FS procedures control the Bias/Variance decomposition by means of the input dimension, establishing a clear connection with the curse of dimensionality. Several critical decision points are studied and compared. First, the stopping criterion. Second, the data set where the value of the loss function is measured. Finally, we also compare two ways of computing the saliency (i.e., the relative importance) of a feature: either first train a network and then remove temporarily every feature or train a different network with every feature temporarily removed. The experiments are performed for linear and non-linear models. Experimental results suggest that the increase in the computational cost associated with retraining a different network with every feature temporarily removed previous

to computing the saliency can be rewarded with a significant performance improvement, specially if non-linear models are used. Although this idea could be thought as very intuitive, it has been hardly used in practice. Regarding the data set where the value of the loss function is measured, it seems clear that the SBS procedure for MLPs takes profit from measuring the loss function in a validation set. A somewhat non-intuitive conclusion is drawn looking at the stopping criterion, where it can be seen that forcing overtraining may be as useful as early stopping.

3. A modification of the quadratic loss function for classification problems, inspired in Support Vector Machines (SVMs) and the AdaBoost algorithm, named *Weighted Quadratic Loss* (*WQL*) function. The modification consists in weighting the contribution of every example to the total error. In the linearly separable case, the solution of the hard margin SVM also minimizes the proposed loss function. The *hardness* of the resulting solution can be controlled, as in SVMs, so that this scheme may also be used for the non-linearly separable case. The error weighting proposed in *WQL* forces the training procedure to pay more attention to the points with a smaller margin. Therefore, variance tries to be controlled by not attempting to overfit the points that are already well classified. The model shares several properties with the SVMs framework, with some additional advantages. On the one hand, the final solution is neither restricted to have an architecture with so many hidden units as points (or support vectors) in the data set nor to use kernel functions. The frequencies are not restricted to be a subset of the data set. On the other hand, it allows to deal with multiclass and multilabel problems in a natural way. Experimental results are shown confirming these claims.

A wide experimental work has been done with the proposed schemes, including artificial data sets, well-known benchmark data sets and two real-world problems from the Natural Language Processing domain. In addition to widely used activation functions, such as the hyperbolic tangent or the Gaussian function, other activation functions have been tested. In particular, sinusoidal MLPs showed a very good behavior. The experimental results can be considered as very satisfactory. The schemes presented in this work have been found to be very competitive when compared to other existing schemes described in the literature. In addition, they can be combined among them, since they deal with complementary aspects of the whole learning process.

Contents

Chapter 1

Introduction

The main objective of this thesis is to study three different components of the learning process with Feed-forward Neural Networks (FNNs): the network architecture, the input dimension and the loss function. These components are related to the Bias/Variance trade-off.

1.1 Motivation

The main objective of a Supervised Machine Learning (SML) system is that of obtaining good generalization performance from a finite set of examples. The goal is not to learn an exact representation of the data set, but rather to build a model of the underlying process which generates the data. In this case, it will be possible to make good predictions on points that do not belong to the data set. Unfortunately, this is a very difficult task. In terms of the Bias/Variance decomposition of the error [Geman et al. 1992; Domingos 2000a], very flexible (i.e., complex) systems may lead to unbiased estimators but probably with high variance. A rigid model, in contrast, may lead to small variance but high bias. The bias reflects the adjustment of the output function to the target values whereas the variance reflects the sensitivity to the training data set. Either high bias or variance can contribute to poor performance. There is a trade-off between the bias and variance contributions to the error, where the optimal performance is achieved. This is related to the *Occam's razor* principle, which states that simple models (whenever capable of representing the data) should be preferred to complex ones.

Several schemes can be found in the literature related to the Bias/Variance trade-off. Some of them try to reduce the variance term while keeping a low bias. Other ones try to reduce the bias term with a limited flexibility. The rest try to control both terms at the same time. Although many of them were developed in an independent way, they can be seen in terms of the Bias/Variance decomposition. Some of them

are general and others are specific for FNNs (the following list does not aim to be an exhaustive one):

1. Feature Selection [Kittler 1978; Liu and Motoda 1998].
2. Models related to Regularization Theory [Tikhonov and Arsenin 1977], such as Regularization Networks [Poggio and Girosi 1990; Girosi et al. 1995].
3. Statistical Learning Theory [Vapnik and Chervonenkis 1971; Vapnik 1982, 1995, 1998a] and Support Vector Machines [Boser et al. 1992; Cortes and Vapnik 1995].
4. Sparse representations [Daubechies 1992; Chen et al. 1998; Graepel et al. 2000].
5. Bayesian Neural Networks [MacKay 1992].
6. Sequential approximations, also named constructive, growing or incremental [Kwok and Yeung 1997a].
7. Pruning algorithms for FNNs [Reed 1993].
8. Specific heuristics for FNNs, such as
   (a) Weight decay [Hinton 1987; Krogh and Hertz 1992].
   (b) Early stopping [Morgan and Bourlard 1990; Prechelt 1998].
   (c) Training with noise [Sietsma and Dow 1991; Grandvalet et al. 1997].
   (d) Soft weight sharing [Nowlan and Hinton 1992].
9. Ensembles of estimators and voting methods, such as
   (a) Committees [Hansen and Salamon 1990; Perrone and Cooper 1993].
   (b) Mixtures of experts [Jacobs et al. 1991; Jordan and Jacobs 1994].
   (c) Boosting [Schapire 1990; Schapire and Singer 1999].
   (d) Bagging [Breiman 1996a].

Some of these schemes have been proved to be related among them (see, for example, [Bishop 1995b; Breiman 1998; Poggio and Girosi 1998; Girosi 1998; Smola et al. 1998; Evgeniou et al. 2000; Xu et al. 2001; Rätsch et al. 2002; András 2002]).

This work is placed in three of these frameworks: Sequential Approximations with Feed-forward Neural Networks, Feature Selection with Multi-Layer Perceptrons and Margin Maximization, which are briefly described in the next subsections.

1.1.1 Sequential Approximations with FNNs

Sequential approximations (also named constructive, growing or incremental) are iteratively constructed by adding terms to the previously obtained partial approximations. We will focus on sequential approximations with FNNs.

In terms of the Bias/Variance decomposition, as far as the number of hidden units (i.e., the number of terms of the approximation) of an FNN grows, bias decreases and variance increases. This happens because the flexibility of the model grows with the number of hidden units [Geman et al. 1992; Barron 1994]. Therefore, choosing an adequate architecture is a very important point in order to obtain good generalization performance.

Sequential approximations with FNNs allow to dynamically construct the network, starting from scratch, without setting *a priori* the architecture [Kwok and Yeung 1997a]. These methods start with a small network (usually with no hidden units), and sequentially add hidden units until a satisfactory solution is found. They can help to find a proper trade-off between bias and variance by controlling, among other things, the number of hidden units.

1.1.2 Feature Selection with Multi-Layer Perceptrons

The problem of Feature Selection (FS) can be defined as follows [Liu and Motoda 1998]: given a set of I candidate features, select a subset that performs the best under some evaluation criterion. We will focus on FS from an SML point of view. That is, our major concern is to improve the predictive performance of the system. In addition to reducing the input dimension, the storage requirements and increasing the computational speed, FS may lead to improve the performance of an SML system, as it has been known for a long time [Kittler 1978, 1986; Siedlecki and Sklansky 1988].

From an SML point of view, FS procedures control the Bias/Variance decomposition by means of the input dimension, establishing a clear connection with the curse of dimensionality [Liu and Motoda 1998]. When too many variables are considered, the system can (surely) fit very well the data set but it may also be too complex, increasing the variance term (whenever the complexity of the system is not controlled by other ways or there is not enough data to filter irrelevant variables, for example). As far as the number of variables decreases, the complexity is reduced, together with the capacity of fitting the data set.

We will deal with specific FS methods for Multi-Layer Perceptrons (MLPs). In general, there is no reason to think that, during the learning process of an MLP, irrelevant variables will not be used by the system in order to fit the training set. For new data, the performance of a system that takes into account this kind of variables can be far to be optimal. This problem can be worsened if some important

features are missing or the number of available examples for the problem is small. Unfortunately, it is not possible to know *a priori* whether we are in these situations or not. This problem is shared by all SML approaches (both linear and non-linear), including MLPs, so that FS procedures become necessary.

FS procedures for MLPs found in the literature usually search the space of feature subsets with the Sequential Backward Selection (SBS) algorithm. It starts by training a network with the whole set of features. Then, the *saliencies* (i.e., the relative importances) of every feature are computed. The input with the lowest saliency is removed. The weights of the network are adjusted, and the loop starts again until a certain criterion is satisfied. SBS may help to detect irrelevant variables in the first steps, since (in theory) generalization should improve when the system does not use this kind of variables. We will also use the SBS algorithm.

1.1.3 Margin Maximization, Support Vector Machines and AdaBoost

1.1.3.1 Support Vector Machines

The idea behind Support Vector Machines (SVMs) can be described as follows [Vapnik 1995; Cristianini and Shawe-Taylor 2000]: the input vectors are mapped into a (usually high-dimensional) inner product space through some non-linear mapping ϕ, chosen *a priori*. In this space (the *feature space*), an optimal hyperplane is constructed. By using a kernel function $K(u, v)$ the mapping can be implicit, since the inner product defining the hyperplane can be evaluated as $\langle \phi(u), \phi(v) \rangle = K(u, v)$. In the linearly separable case, an optimal hyperplane means a separating hyperplane with maximal distance with respect to the data set (hard margin). That is equivalent to a hyperplane with maximum normalized (or geometrical) margin with respect to the data set. When the data are not linearly separable (either in the input space or in the feature space), some tolerance to noise is introduced in the model (soft margin case). The most usual kernel functions $K(u, v)$ are polynomial, Gaussian-like or some particular sigmoids.

Using Lagrangian and Kuhn-Tucker theory, the maximal margin hyperplane for SVMs is obtained by solving a constrained optimization problem. In the soft margin case, the Bias/Variance trade-off is controlled by a parameter C which expresses the importance given to the violation of the margin constraints and limits the complexity of the model. When $C = 0$ the variance is also 0, but the bias may be very high. When $C = \infty$, the hard margin hyperplane (if exists) is a solution with low bias (it allows to classify correctly every point in the data set), but probably with high variance. In some sense, the parameter C acts as a regularization parameter.

In their original definition, SVMs are designed for 2-class problems. In addition, as a consequence of the constrained optimization problem posed, the output function

of an SVM is restricted to be a linear combination of kernel functions dependent on a subset of the data set (the *support vectors*).

1.1.3.2 AdaBoost

The AdaBoost algorithm is a particular boosting algorithm introduced in [Freund and Schapire 1996, 1997] and later improved in [Schapire and Singer 1999]. In AdaBoost, the weak hypotheses are learned sequentially, one at a time. Conceptually, at each iteration the weak hypothesis is biased to classify the examples which were most difficult to classify by the preceding weak hypotheses. The functional margin of every example is considered in order to construct every weak hypothesis, which are linearly combined into a single output rule named the *combined hypothesis*.

The learning bias of AdaBoost is proven to be very aggressive on maximizing the functional margin of the training examples, since it concentrates on the examples with the smallest margins [Schapire et al. 1998]. This makes a clear connection to the SVMs learning paradigm.

1.2 Contributions

In this thesis we present three schemes related to the control of the Bias/Variance decomposition for FNNs with the (sometimes modified) quadratic loss function:

1. An algorithm for sequential approximation with FNNs.

2. A study and comparison of different criteria to perform Feature Selection with MLPs and the SBS procedure within the *wrapper* approach.

3. A modification of the quadratic loss function for classification problems, inspired in Support Vector Machines and the AdaBoost algorithm.

A wide experimental work has been done with the proposed schemes, showing a very good performance and allowing to extract interesting conclusions and future lines of research.

In the next subsections we briefly describe these contributions.

1.2.1 *SAOCIF*: A Sequential Algorithm for FNNs

The problem of finding good weights in sequential approximations with FNNs is a very hard non-linear problem. Most of the sequential approximations proposed in the literature select the new frequency (the non-linear weights) guided by the approximation of the residue of the partial approximation. Although this strategy

leads to approximations convergent towards the target function, it may be far from being the best one.

An algorithm for sequential approximations with FNNs is described, named *Sequential Approximation with Optimal Coefficients and Interacting Frequencies* (*SAOCIF*). The new frequency is selected taking into account the interactions of this frequency with the previously selected ones. The interactions are discovered by means of their optimal coefficients (the linear weights). The aim is that it may be able to obtain better approximations, with the same number of hidden units, than just matching the residue as best as possible. Likewise, the same level of approximation may be achieved with less hidden units. In terms of the Bias/Variance decomposition, it will be possible to obtain simpler models with the same bias. In the proposed algorithm, a number of heuristics can be used to select the new frequencies. The idea behind *SAOCIF* can be extended to approximation in Hilbert spaces, maintaining orthogonal-like properties. In this case, the importance of the interacting frequencies lies in the expectation of increasing the rate of approximation. Experimental results show that the idea of interacting frequencies allows to construct better approximations than that of matching the residue.

Some author's publications related to these ideas are [Romero and Alquézar 2002a, 2004].

1.2.2 Performing Feature Selection with MLPs

Although from an SML point of view trying to minimize the value of the loss function is surely the optimal saliency for FS (and that will be the saliency used in this work), there is no commonly accepted criterion about neither when to stop the training phase nor which is the best data set to measure the saliency. In addition, the saliency of a variable is almost always computed when the network has been trained with that feature, in contrast to the alternative of computing the saliency in a network trained without it. In order to compute the value of the loss function, as in our case, it may be very different first train the network and then remove temporarily a feature than first remove temporarily the feature and then train the network. There exists a lack of comparative results among these issues in the literature.

A study and comparison of different criteria to perform FS with MLPs and the SBS procedure is described. Several critical points are studied and compared:

1. The stopping criterion of the network training. We compared the point where the minimum loss function in a validation set is achieved with the (probably overtrained) point where a minimum of the training set is obtained.

2. The data set where the value of the loss function is measured. In our experiments, the loss function was computed either in the training set or a validation set.

3. The network retraining with respect to the computation of the saliency: first train the network and then remove temporarily the feature in order to compute the saliency or first remove temporarily the feature and then train the network previous to computing the saliency.

The experiments were performed for linear and non-linear models.

This scheme has generated several author's publications [Sopena et al. 1999b; Mijares et al. 2001; Selva et al. 2002, 2003; Armadans et al. 2003; Romero et al. 2003; Sopena and Romero 2004; Romero and Sopena 2004].

1.2.3 Maximizing the Margin with FNNs

A modification of the quadratic loss function for classification problems is proposed. We will refer to this scheme as *Weighted Quadratic Loss* (*WQL*) function. The modification consists in weighting the contribution of every example to the total error. This weighting depends on the margin and is inspired in the AdaBoost algorithm. In the linearly separable case, the solution of the hard margin SVM also minimizes the proposed loss function. The *hardness* of the resulting solution can be controlled, as in SVMs, so that this scheme can also be used for the non-linearly separable case. The error weighting proposed in *WQL* forces the training procedure to pay more attention to the points with smaller margin: during the training procedure, the contribution of every point decreases with its margin. Therefore, variance tries to be controlled by not attempting to overfit the points that are already well classified. The model shares several properties with the SVMs framework, with some additional advantages. On the one hand, the final solution is neither restricted to have an architecture with so many hidden units as points (or support vectors) in the data set nor to use kernel functions. The frequencies are not restricted to be a subset of the data set. On the other hand, it allows to deal with multiclass and multilabel problems in a natural way. Experimental results are shown confirming these claims.

Some author's publications related to this approach can be found in [Romero and Alquézar 2002b; Romero et al. 2004a,b].

1.2.4 Experiments

The previously described schemes have been empirically tested in a variety of situations:

1. In artificial data sets.

2. In well-known benchmark data sets.

3. In two real-world problems from the Natural Language Processing domain.

The experimental results can be considered as very satisfactory. The schemes presented in this thesis have been found to be very competitive when compared to other existing methods described in the literature.

In addition to widely used activation functions, such as the hyperbolic tangent or the Gaussian function, we worked with other activation functions through the experiments. In particular, the behavior of sinusoidal MLPs (i.e., MLPs where the activation function of the hidden units was either the sine or the cosine) was very satisfactory.

1.2.4.1 Experiments with *SAOCIF*

Experimental results for *SAOCIF* show that the idea of interacting frequencies allows to obtain better results than matching the residue for both approximation and generalization purposes. Surprisingly, selecting the frequencies from the points in the data set allows to obtain similar (and sometimes superior) results than other more sophisticated strategies, with a much smaller computational cost. In addition, selecting the frequencies from the points in the data set seems well suited not only for Radial Basis Function Networks, as commonly used, but also for Multi-Layer Perceptrons. In this case, the resulting model shares with SVMs the property that their frequencies are a subset of the data set.

1.2.4.2 Experiments with the SBS Procedure for MLPs

Experimental FS results using the SBS procedure for MLPs suggest that the high computational cost associated with retraining the network with every feature temporarily removed previous to computing the saliency can be rewarded with a significant performance improvement, specially if non-linear models are used. Although this idea could be thought as very intuitive, it has been hardly used in practice. Regarding the data set where the value of the loss function is measured, it seems clear that the SBS procedure for MLPs takes profit from measuring the loss function in the validation set. Again, this is a quite intuitive idea, although many models in the literature do not take this approach. A somewhat non-intuitive conclusion is drawn looking at the stopping criterion, where it is suggested that forcing overtraining may be as useful as using early stopping within the SBS procedure for MLPs. There exist an important improvement in the overall results with respect to learning with the whole set of variables and compared with other existing FS wrappers in the literature. Although the model can be further improved, the good results obtained are mainly due, in our opinion, to a proper detection of irrelevant variables.

1.2.4.3 Experiments with *WQL*

Experiments on artificial data sets show that models equivalent to hard margin SVMs can be obtained by training FNNs with *WQL* in linearly separable cases, both for two-class and multiclass problems. In addition, models similar to non-linear hard SVMs can be obtained without an "SVM architecture" (i.e., with so many hidden units as points in the data set or support vectors) and without kernel functions, whenever the *WQL* is minimized. In the real-world problems tested, a consistent correlation was observed between FNN models obtained minimizing *WQL* and SVM models. Both linear models and non-linear ones (with and without kernel functions) had this behavior, in two-class and multiclass problems.

1.2.5 Combination of the Proposed Ideas

The schemes presented in this work can be combined among them, since they are related to complementary aspects of the entire learning process:

1. The SBS procedure may be performed with *SAOCIF*.
2. *SAOCIF* can be modified so as to deal with the *WQL* function.
3. The SBS procedure can also be performed minimizing the *WQL* function.

1.3 Thesis Overview

Chapter 2 includes some background material and a description of the previous work found in the literature related to the contributions of this thesis. The algorithm for sequential approximation with FNNs is described in chapter 3, together with its experimental evaluation. In chapter 4, the experimental study of different criteria to perform FS with MLPs for classification problems is carried out. Chapter 5 is devoted to motivate, describe and test the proposed modification of the quadratic loss function. The document ends with some concluding remarks and a prospect for future work. The appendices contain the description of the data sets used in the experiments and several author's publications related to the contents of this thesis.

Chapter 2

Background Material

We will focus our work on Feed-forward Neural Networks (FNNs) and their most usual models, namely Multi-Layer Perceptrons (MLPs) and Radial Basis Function Networks (RBFNs). In this chapter, an overview of some concepts is given in order to make the whole document more self-contained. First, general concepts are visited, namely Supervised Machine Learning, the quadratic loss function, Feed-forward Neural Networks and the Bias/Variance decomposition. Most of this material will probably be already known to the reader. In the last three sections of this chapter we briefly describe the three frameworks where this thesis is placed: Sequential Approximations with FNNs, Feature Selection with MLPs and Margin Maximization, with special attention to their relationships with the Bias/Variance decomposition.

2.1 Notation

The following acronyms are used in this work, listed in alphabetical order within three categories:

- General acronyms:
  - BGA: Breeder Genetic Algorithm.
  - BP: Back-Propagation.
  - CV: Cross-Validation.
  - FNNs: Feed-forward Neural Networks.
  - FS: Feature Selection.
  - MLPs: Multi-Layer Perceptrons.
  - RBF: Radial Basis Function.
  - RBFNs: Radial Basis Function Networks.

  - SBS: Sequential Backward Selection.
  - SFS: Sequential Forward Selection.
  - SML: Supervised Machine Learning.
  - SVMs: Support Vector Machines.
  - TC: Text Categorization.
  - VC-dimension: Vapnik-Chervonenkis dimension.
  - WSD: Word Sense Disambiguation.

- Acronyms for other authors' work:

  - CC: Cascade-Correlation, described in [Fahlman and Lebiere 1990].
  - DNC: Dynamic Node Creation, described in [Ash 1989].
  - GMDH: Group Method of Data Handling, described in [Ivakhnenko 1971].
  - HEA data sets: Data sets used in [Hwang et al. 1994].
  - ILQP: Incremental Linear Quasi-Parallel, described in [Kůrková and Beliczyński 1995a,b].
  - KMP: Kernel Matching Pursuit, described in [Vincent and Bengio 2002].
  - MN: Meiosis Networks, described in [Hanson 1990].
  - MP: Matching Pursuit, described in [Mallat and Zhang 1993].
  - OLSL: Orthogonal Least Squares Learning, described in [Chen et al. 1991a].
  - OMP: Orthogonal Matching Pursuit, described in [Pati et al. 1993].
  - PPLN: Projection Pursuit Learning Network, described in [Hwang et al. 1994].
  - PPR: Projection Pursuit Regression, described in [Friedman and Stuetzle 1981].
  - RAN: Resource-Allocating Network, described in [Platt 1991].
  - ZM98: The sequential learning algorithm for MLPs described in [Zhang and Morris 1998].

- Specific acronyms in this work:

  - *BPW*: Back-Propagation minimizing the *WQL* function.
  - *MFT*: Maximum Fourier Transform.
  - *OCMFT*: Optimal Coefficients after Maximum Fourier Transform.

- *SAOCIF*: Sequential Approximation with Optimal Coefficients and Interacting Frequencies.
- *WQL*: Weighted Quadratic Loss.

2.2 Supervised Machine Learning

The main problem in Supervised Machine Learning (SML) is that of finding a function which predicts the target in the best possible way from a finite set of examples. More generally, a model of the conditional probability distribution of the target with respect to the input is desired [Bishop 1995a; Vapnik 1998a; Anthony and Bartlett 1999]. In this context, several basic issues need to be defined:

1. The *problem model*, which can be usually described with two components:

   (a) A *generator of input vectors* $x \in X$, drawn independently from a fixed but unknown distribution $P(x)$.

   (b) A *supervisor* that returns a target vector $y \in Y$ for every input vector x, according to a conditional distribution $P(y|x)$, also fixed but unknown.

   Several important problems can be enclosed within this model:

   (a) *Regression*, where y represents a (maybe noisy) measurement of some real-valued quantity.

   (b) *Classification*, where y belongs to a finite set of classes. The classification problem can also be considered as a particular case of the regression problem.

   In this work we will concentrate on these problems. We will suppose that $Y \subseteq \mathbb{R}$. The extension to $Y \subseteq \mathbb{R}^O$ is straightforward. We will also assume that $X \subseteq \mathbb{R}^I$.

   The distributions $P(x)$ and $P(y|x)$ in the problem model are usually unknown. Instead, a data set of examples

   $$D = \{(x_1, y_1), \cdots, (x_L, y_L)\} \subset X \times Y \tag{2.1}$$

   is given. The observations[1] are assumed to be independent and identically distributed, drawn according to $P(x, y) = P(y|x) \cdot P(x)$.

[1] Without distinction, we will refer to the elements of D as examples, observations, points or vectors.

2. The *loss (or error) function* $\mathcal{L} : Y \times Y \to \mathbb{R}^+$ is the function that measures the discrepancy between two values in Y. Loss functions use to be distances. Usual choices for $\mathcal{L}$ are the quadratic loss function ($\mathcal{L}^2(u,v) = (u-v)^2$, see section 2.3) for regression problems, and the 0/1 loss function ($\mathcal{L}^{0/1}(u,v) = 0$ if $u = v$, and 1 otherwise) only for classification problems.

3. The *hypothesis class* $\mathcal{H}$ is a subset of functions from $X \times \Sigma$ to Y where the solution will be selected from. Σ is the space of parameters of the hypothesis class $\mathcal{H}$. Therefore, once $\sigma \in \Sigma$ is fixed, we have a function from X to Y.

The loss function and the hypothesis class can be chosen by the user. In SML, the goodness of a function $f : X \to Y$ is measured by the expected value of the loss function $\mathcal{L}$, the *expected risk* functional:

$$R(\mathcal{L}, f) = E_{X \times Y}\left[\mathcal{L}(y, f(x))\right] = \int_{X \times Y} \mathcal{L}(y, f(x))\ P(x,y)\ dx\ dy. \tag{2.2}$$

The objective of an SML system is to find a function $f : X \to Y$ so as to minimize the expected risk functional (2.2). The *learning algorithm* will attempt to model the behavior of the target by constructing an *output function* $f^o \in \mathcal{H}$. Therefore, the objective is that of finding $\sigma^* \in \Sigma$ so that the discrepancy $R(\mathcal{L}, f^o(\cdot, \sigma^*))$ between the target and the output function is minimum. The set of parameters $\sigma^* \in \Sigma$ is usually adjusted during the learning process. In practice, the choice of the output function (or equivalently, σ^*) is based on the data set of examples D in (2.1) and maybe some prior information on the problem[2]. The *generalization* ability derived from the minimization of (2.2) implies that it will be possible to predict (with minimum expected error according to $\mathcal{L}$) the function behavior on points that do not belong to the data set D.

As an approximation to the expected risk, the *empirical risk* functional is defined as the risk evaluated in the data set D:

$$R_e(\mathcal{L}, f) = \frac{1}{L} \sum_{i=1}^{L} \mathcal{L}(y_i, f(x_i)). \tag{2.3}$$

2.3 The Quadratic Loss Function

We will mainly work with the quadratic loss function:

$$\mathcal{L}^2(u,v) = (u-v)^2. \tag{2.4}$$

[2]From now on, we will omit the parameter σ in $f^o(x, \sigma)$

The functional $R(\mathcal{L}^2, f)$ is the squared norm of $(y - f(x))$ in the Hilbert space L^2 of squared integrable functions, where the integral is defined with regard to the probability measure of the model[3]. We will suppose that the functions involved in our work belong to L^2 when necessary.

The quadratic error function has some desirable properties for both regression and classification problems, as explained next.

As it is well known, the *regression function* $E_Y[y|x] = \int_Y y\ P(y|x)\ dy$ minimizes the expected risk

$$R(\mathcal{L}^2, f) = \int_{X \times Y} (y - f(x))^2\ P(x, y)\ dx\ dy \tag{2.5}$$

for the quadratic loss function (2.4). This is so because (see, for example, [Richard and Lippmann 1991; Geman et al. 1992; Niyogi and Girosi 1999])

$$\begin{aligned} R(\mathcal{L}^2, f) &= E_{X \times Y}\left[(y - E_Y[y|x])^2\right] + E_{X \times Y}\left[(f(x) - E_Y[y|x])^2\right] \\ &= E_{X \times Y}\left[\mathcal{L}^2(y, E_Y[y|x])\right] + E_X\left[(f(x) - E_Y[y|x])^2\right]. \end{aligned} \tag{2.6}$$

for any $f : X \to Y$. This result is similar to the Pythagoras' Theorem, and it is a consequence of the orthogonality of $(y - E_Y[y|x])$ and $(f(x) - E_Y[y|x])$ for any $f : X \to Y$.

The first term in the right hand of (2.6) does not depend on f. Therefore,

1. Regardless of the hypothesis class, the expected risk may not be zero, since it will be at least $E_{X \times Y}[\mathcal{L}^2(y, E_Y[y|x])]$. When the target is a function of the input values $y = g(x)$, this term vanishes, since $E_Y[y|x] = g(x)$.

2. The minimum expected risk is achieved when $f(x) = E_Y[y|x]$, as previously said. Therefore, we can consider that $E_Y[y|x]$ is the target function.

For classification problems, when a 1-of-C target coding scheme is used (the correct class unity, all others zero), $E_Y[y|x]$ is the Bayesian posterior probabilities that the input belong to the class (see, for example, [Richard and Lippmann 1991]).

[3] An *inner product space* (or pre-Hilbert space) is a vector space with an *inner product* $\langle \cdot, \cdot \rangle : H \times H \to \mathbb{C}$ such that $\forall \alpha_1, \alpha_2 \in \mathbb{C}\ \ \forall x_1, x_2, x, y \in H$

1. $\langle x, y \rangle = \overline{\langle y, x \rangle}$.
2. $\langle \alpha_1 \cdot x_1 + \alpha_2 \cdot x_2, y \rangle = \alpha_1 \langle x_1, y \rangle + \alpha_2 \langle x_2, y \rangle$.
3. $\langle x, x \rangle \geqslant 0$ (in particular $\langle x, x \rangle \in \mathbb{R}$).
4. $\langle x, x \rangle = 0$ if and only if $x = 0$.

A *Hilbert space* is a complete inner product space [Yosida 1965]. Sometimes, a Hilbert space is defined as an inner product space, and complete Hilbert spaces are used [Achieser 1956].

Linear transformations of the target values (from $\{0,1\}$ to $\{-1,+1\}$, for example) maintain, in essence, this property. Other nice properties of the quadratic error function for classification problems can be found in [Kearns and Schapire 1990, 1994].

For the quadratic loss function, the empirical risk takes the form of a sum-of-squares:

$$R_e(\mathcal{L}^2, f) = \frac{1}{L}\sum_{i=1}^{L} \mathcal{L}^2(y_i, f(x_i)) = \frac{1}{L}\sum_{i=1}^{L} (y_i - f(x_i))^2. \tag{2.7}$$

For classification problems, the minimization of the sum-of-squares function leads, asymptotically, to the approximation of the Bayesian posterior probabilities [Ruck et al. 1990b].

2.4 Feed-forward Neural Networks

The hypothesis class we will work with in this thesis is the class of FNNs with one hidden layer of units. An extensive description of FNNs can be found in [Haykin 1994; Bishop 1995a], for example.

The well-known architecture (number of hidden units, activation functions and connectivity) of an FNN is structured by layers of units, with connections between units from different layers in forward direction. A fully connected FNN with one output unit and one hidden layer of units computes the function $f^o_{FNN} : \mathbb{R}^I \to \mathbb{R}$:

$$f^o_{FNN}(x) = \varphi_0\left(b_0 + \sum_{k=1}^{N} \lambda_k\ \varphi_k\left(\omega_k, x, b_k\right)\right) \qquad \omega_k \in \mathbb{R}^I \quad \lambda_k, b_0, b_k \in \mathbb{R}, \tag{2.8}$$

where N is the number of units in the hidden layer. Whereas output activation functions $\varphi_0 : \mathbb{R} \to \mathbb{R}$ use to be sigmoidal or linear, different models of FNNs can be obtained by changing the activation functions of the hidden units φ_k. FNNs with several output units can also be defined, in a natural way. For convenience, we will divide the weights into *frequencies* $\{\omega_k\}_{k=1}^N$, *coefficients* $\{\lambda_k\}_{k=1}^N$ and *biases* $\{b_k\}_{k=0}^N$.

Note that the appearance of the output function given by (2.8) is determined by the architecture of the FNN.

The objective of the training process in an FNN is to choose adequate parameters to minimize a predetermined cost function. The previously defined sum-of-squares loss function (2.7) is the most usual:

$$E(D) = \frac{1}{L}\sum_{i=1}^{L} (y_i - f^o_{FNN}(x_i))^2, \tag{2.9}$$

where we have omitted the factor L for convenience. For C-class problems, architectures with C output units are used [Bishop 1995a], and the objective is recast to minimize

$$E^C(D) = \frac{1}{L}\sum_{i=1}^{L}\sum_{c=1}^{C}(y_i^c - f_{FNN}^{o,c}(x_i))^2. \tag{2.10}$$

Although the global minimum of (2.9) or (2.10) exists, it cannot be found with a low computational cost in the general case [Scarselli and Tsoi 1998]. Most of the times, a local minimum is found.

According to their definition, FNNs can deal with regression and classification problems in a natural way. The most common models of FNNs are Multi-Layer Perceptrons (MLPs) and Radial Basis Function Networks (RBFNs).

2.4.1 Multi-Layer Perceptrons

An MLP with one hidden layer of units and a linear output unit computes a function $f_{MLP}^o : \mathbb{R}^I \to \mathbb{R}$ as follows:

$$f_{MLP}^o(x) = b_0 + \sum_{k=1}^{N}\lambda_k\,\varphi\,(\omega_k \cdot x + b_k) \quad \omega_k \in \mathbb{R}^I \quad \lambda_k, b_0, b_k \in \mathbb{R}, \tag{2.11}$$

where $\omega_k \cdot x$ represents the inner product in $\mathbb{R}^I$, ω_k is the weight vector associated with the connections between the input layer units and the unit k of the hidden layer. The biases b_k are external parameters for each unit, and they can be considered as weights with a simple transformation. The activation function $\varphi : \mathbb{R} \to \mathbb{R}$ is usually the same for all the hidden units. The most usual activation functions are sigmoidal-like (continuous, non-constant, increasing and bounded), such as the logistic function $\varphi(x) = \frac{1}{1+e^{-x}}$ or the hyperbolic tangent function $\varphi(x) = \frac{1-e^{-x}}{1+e^{-x}}$, although many other functions may be used [Leshno et al. 1993]. MLPs with several layers of hidden units can also be defined in the same way.

The architecture is usually fixed *a priori*, whereas the weights are learned during the training process. The most celebrated training algorithms with fixed architecture are gradient descent with Back-Propagation (BP) [Werbos 1974; Rumelhart et al. 1986] and its many variations. There exist, however, other schemes that sequentially construct the architecture (see section 2.6).

2.4.2 Radial Basis Function Networks

An RBFN with a linear output unit computes a function $f^o_{RBFN} : \mathbb{R}^I \to \mathbb{R}$ as follows:

$$f^o_{RBFN}(x) = b_0 + \sum_{k=1}^{N} \lambda_k \, \varphi\left(\frac{x - \omega_k}{b_k}\right) \quad \omega_k \in \mathbb{R}^I \quad \lambda_k, b_0, b_k \in \mathbb{R}, \tag{2.12}$$

where $\varphi : \mathbb{R}^I \to \mathbb{R}$ is the Radial Basis Function (RBF), which is usually the same for all the hidden units. The most frequent RBFs are radially symmetric, such as $\varphi(x) = e^{-\|x\|^2}$. In this case, ω_k is the center of the function, whereas b_k is the RBF-width. As for MLPs, many other functions may be used [Liao et al. 2003], but the architecture only contains one layer of hidden units, and the output units use to be linear.

The learning process of an RBFN is usually divided in two consecutive steps:

1. The selection of the frequencies (the centers), usually in an unsupervised way. The k-means algorithm [McQueen 1967] or the expectation maximization algorithm [Duda and Hart 1973] are commonly used to this end.

2. The computation of the coefficients, which are usually found through gradient descent on the quadratic loss function.

The whole set of weights of the RBFN can also be optimized with a global gradient descent procedure. As for MLPs, there exist other techniques that sequentially construct the architecture (see section 2.6).

2.4.3 Universal Approximation of FNNs

The universal approximation capability (see section 2.5.1) is a desirable property for a hypothesis class, since it allows to have the possibility of choosing the output function that minimizes the expected risk.

FNNs have been shown to be universal approximators for several "well-behaved" classes of functions (including, for example, continuous, integrable or square integrable functions) with many families of activation functions and several metrics [Leshno et al. 1993; Liao et al. 2003]. In particular, MLPs and RBFNs are universal approximators with the quadratic loss function, provided that no constraints are imposed on the number of hidden units and the size of the weights.

In practice, however, when a data set is used to compute the output function, the universal approximation capability may not be very useful, even if the global minimum of the empirical risk can be found. Other elements, such as the number of available examples, the input dimension or the particular hypothesis class used[4],

[4] In practice, the number of hidden units and the weights must be constrained. In addition, the same data set may be approximated with many different functions.

are involved in the whole process. Section 2.5 is devoted to the discuss some of these issues.

2.5 Expected Risk and the Bias/Variance Decomposition

2.5.1 Approximation Error and Estimation Error

Let us return to the main problem of SML. For any output function f^o, there are two components that contribute to the expected risk $R(\mathcal{L}, f^o)$ (see, for example, [Barron 1994; Niyogi and Girosi 1999]):

1. The *approximation error* is the distance, in terms of the expected risk, between the target and its closest (i.e., optimal) function f^* in the hypothesis class $\mathcal{H}$. In other words, the approximation error indicates how well can the target be approximated in $\mathcal{H}$. A hypothesis class is said to have the *universal approximation property* when it is broad enough to contain the target function or an arbitrarily close approximation to it. When the target is not a function of the input values, the distance is measured with respect to the function which minimizes the expected risk ($E_Y[y|x]$ for the quadratic loss function, for example). The *rate of the approximation error* indicates how changes the approximation error when $\mathcal{H}$ varies. For example, if the hypothesis class contains finite linear combinations of simple functions, the rate of approximation may depend, among other things, on the number of terms of the approximation. An approximation scheme is said to have the *convergence property* if it is able to produce a sequence of output functions convergent to the target function. The approximation error does not depend on the data set, and it is studied within the Approximation Theory.

2. Since the output function is constructed from a particular set of observations, it may happen that the information contained in the data set does not allow to find a good enough approximation of f^*. The *estimation error* is related to the difficulty of estimating the optimal parameters with finite data, and it is a typical problem of the Statistics framework. There are several definitions of the estimation error. In [Barron 1994], for example, it is defined as the distance (in terms of the expected risk) between the output function and the optimal approximation f^* in $\mathcal{H}$. Within the Statistical Learning Theory (see [Vapnik 1998a], for example), the estimation error is usually defined as the difference between the expected and the empirical risk of the output function. Most of the times, the output function is such that minimizes the empirical risk, so that in Statistical Learning Theory the estimation error is related to the

suitability of minimizing the empirical risk in order to minimize the expected risk. The *sample complexity* is related to the number of examples needed to guarantee a near-optimal estimation error.

Roughly speaking, the expected risk is bounded by the sum of the approximation and the estimation error. Therefore, minimizing both terms is a sufficient condition to achieve the desired objective. Ideally, we would like to have a hypothesis class flexible enough to approximate well large classes of targets (i.e., to have small approximation error). However, for a fixed number of examples, the estimation error is expected to increase with the flexibility of the hypothesis class. There are many results that support this statement, both for FNNs and other approaches. The next section discusses some of these results.

2.5.2 Theoretical Bounds on the Expected Risk

For the quadratic loss function, in [Barron 1994] it is shown that the expectation (over data sets of fixed size L) of the expected risk of a sigmoidal MLP (2.11) with bounded weights $\left(|\omega_k|_1 \leqslant \tau_N,\ |b_k| \leqslant \tau_N,\ \sum_{k=1}^{N} |\lambda_k| \leqslant C_f\right)$ which minimizes the empirical risk is, under several conditions, bounded by:

$$O\left(\frac{C_f^2}{N}\right) + O\left(\frac{N \cdot I \cdot \log L}{L}\right), \tag{2.13}$$

where N is the number of hidden units, τ_N satisfies certain technical conditions related to sigmoidal functions, I is the input dimension, L is the number of observations, and C_f is the first absolute moment of the Fourier magnitude distribution of the target function f (which must be finite). The term C_f may grow with I, although there are a number of interesting functions for which C_f exhibits only a moderate growth $O(I)$, as shown in [Barron 1993]. The important point is that the first term in (2.13) is a bound on the approximation error, whereas the second one is related to the estimation error. An increase in the number of hidden units (number of terms in the approximation) leads to more flexible models that allow to approximate more target functions. However, this fact has the effect of making more difficult the estimation of f^*. As it was expected, the number of examples and the input dimension also influence the estimation error. For RBFNs, a slightly different bound can be found in [Niyogi and Girosi 1994, 1999], also for the quadratic loss function but with a different technical approach. In particular, it is proved that with probability at least $1 - \eta$ the expected risk for the RBFNs (2.12) with Gaussian activation function and bounded coefficients $\left(\sum_{k=1}^{N} |\lambda_k| \leqslant C\right)$ minimizing the

empirical risk is bounded by:

$$O\left(\frac{1}{N}\right) + O\left(\sqrt{\frac{N \cdot I \cdot \log(N \cdot L) - \log \eta}{L}}\right), \tag{2.14}$$

Starting from a different point of view, the Statistical Learning Theory provides probabilistic bounds on the distance between the empirical and the expected risk. In particular, it can be proved that with probability at least $1 - \eta$ the following inequality holds for all functions $f^o \in \mathcal{H}$ such that $0 \leqslant \mathcal{L}(y, f^o(x)) \leqslant B$ [Vapnik 1995, 1998a]:

$$R(\mathcal{L}, f^o) \leqslant R_e(\mathcal{L}, f^o) + \frac{B \cdot E}{2}\left(1 + \sqrt{1 + \frac{4R_e(\mathcal{L}, f^o)}{B \cdot E}}\right), \tag{2.15}$$

where

$$E = 4\frac{h\left(\log \frac{2L}{h} + 1\right) - \log \frac{\eta}{4}}{L},$$

L is the number of observations and h is the *Vapnik-Chervonenkis dimension* (VC-dimension) of $\mathcal{H}$. In the particular case of the 0/1 loss function, a simpler expression can be proved:

$$R(\mathcal{L}, f^o) \leqslant R_e(\mathcal{L}, f^o) + \frac{\sqrt{E}}{2}. \tag{2.16}$$

The VC-dimension of a set of indicator functions (functions which take only the values zero and one) is the maximum number h of vectors which can be separated in all 2^h possible ways with functions of this set [Vapnik and Chervonenkis 1971; Vapnik 1982]. Roughly speaking, the VC-dimension measures the capacity of the hypothesis class to separate an arbitrary set of vectors (i.e., its flexibility). The first terms in the right hand of (2.15) and (2.16) are related to the approximation error, whereas the second are related to the estimation error, similar to (2.13) and (2.14). An increase in the VC-dimension leads to more flexible models that allow to fit better the data set, reducing the empirical risk. However, large VC-dimensions must be compensated with a large number of examples in order to reduce the second term in (2.15) or (2.16). Although the quadratic loss function is not bounded in the general case, the previous result can be applied to FNNs minimizing the quadratic loss function with bounded weights and a limited number of hidden units.

A different result can be found in [Bartlett 1998], where some bounds of the expected risk for classification problems are shown. In particular, it is proved that with probability at least $1 - \eta$ every $f^o \in \mathcal{H}$ satisfies that its expected risk (with the

0/1 loss function) is bounded by

$$\frac{|\{i \in D \ \mid \ y_i f^o(x_i) < \gamma\}|}{L} + \sqrt{\frac{2\left(d \ln\left(\frac{34Le}{d}\right) \log_2(578L) + \ln\left(\frac{4}{\eta}\right)\right)}{L}}, \tag{2.17}$$

where $\gamma > 0$ and d is the *fat-shattering dimension* [Kearns and Schapire 1990] of H with margin γ. The fat-shattering dimension is a flexibility measure that generalizes the concept of VC-dimension to classes of real-valued functions: in addition to shattering the vectors, a margin greater or equal than γ is required. In the limit, when $\gamma \to 0$, it is referred to as *combinatorial dimension* [Haussler 1989] or *pseudo-dimension* [Haussler 1992]. The fat-shattering dimension decreases as γ increases [Anthony and Bartlett 1999].

As a consequence, in [Bartlett 1998] it is proved that with probability at least $1 - \eta$ every MLP f^o in (2.11) with

1. Activation function $\varphi : \mathbb{R} \to [-1, +1]$ non-decreasing,
2. Bounded coefficients $\sum_{k=1}^{N} |\lambda_k| \leqslant A$, with $A \geqslant 1$.

satisfies that its expected risk (with the 0/1 loss function) is bounded by

$$\frac{|\{i \in D \ \mid \ y_i f^o(x_i) < \gamma\}|}{L} + \sqrt{\frac{B\left(\frac{A^2 I}{\gamma^2} \log\left(\frac{A}{\gamma}\right) \log^2 L + \log\left(\frac{1}{\eta}\right)\right)}{L}}, \tag{2.18}$$

where $0 < \gamma \leqslant 1$, L is the number of observations and B is a universal constant. For the quadratic loss function, a similar bound can be obtained, since

$$\frac{\sum_{i=1}^{L} (y_i - f^o(x_i))^2}{L} < \varepsilon \ \text{ implies } \ \frac{|\{i \in D \ \mid \ y_i f^o(x_i) < \gamma\}|}{L} < \frac{\varepsilon}{(1-\gamma)^2},$$

whenever $0 < \gamma < 1$ and $y_i \in \{-1, +1\}$. Similar to (2.13), (2.14), (2.15) and (2.16), the first terms in the right hand of (2.17) and (2.18) are related to the approximation error and the second one to the estimation error. The margin γ controls the trade-off between both terms. However, the difference is specially important with respect to (2.15), since the MLPs in (2.18) have infinite VC-dimension but finite fat-shattering dimension. The bound on the 1-norm of the coefficients plays a crucial role in this result.

Results like (2.13), (2.14), (2.15), (2.16), (2.17) or (2.18) suggest the Structural Risk Minimization principle [Vapnik 1982, 1992, 1998a, 1999], based on the simultaneous minimization of the two terms in these expressions. In practice, a nested sequence of hypothesis classes

$$\mathcal{H}_1 \subset \mathcal{H}_2 \subset \cdots \subset \mathcal{H}_M$$

is defined, where each $\mathcal{H}_k$ has finite (and increasing with k) complexity measure. Then, given a set of observations, the Structural Risk Minimization principle chooses the particular element f_k^o from every $\mathcal{H}_k$ such that the corresponding upper bound on the expected risk is minimized. Finally, the best f_k^o is selected. When the upper bound is similar to (2.16), then it is reasonable to choose f_k^o minimizing the empirical risk in $\mathcal{H}_k$. These ideas lead to the development of Support Vector Machines (SVMs) [Boser et al. 1992; Cortes and Vapnik 1995], based on the VC-dimension of the set of hyperplanes and its relationship with the maximal geometric margin (see section 2.8).

There are many other results similar (in a somewhat wide sense) to (2.13), (2.14), (2.15), (2.16), (2.17) or (2.18) (see [Pollard 1984; Dudley 1984; Haussler 1989; Baum and Haussler 1989; Barron 1990; Moody 1991, 1992; Haussler 1992; Faragó and Lugosi 1993; Lugosi and Pintér 1996; Lee et al. 1996; Schapire et al. 1998; Cheang and Barron 1999; Graepel et al. 2000; Herbrich et al. 2000], for example).

In summary, some common features of these results are:

1. The flexibility of the hypothesis class is controlled (and limited) by a number of parameters, such as the number of hidden units and the size of the weights in (2.13) and (2.14), the VC-dimension in (2.15) and (2.16), the fat-shattering dimension in (2.17) or the 1-norm of the coefficients in (2.18). The flexibility of the model may also be controlled by means of the activation function, for example.

2. As a general rule, very flexible models lead to small bounds on the approximation error and large bounds on the estimation error. Little flexibility implies large bounds on the approximation error and small ones on the estimation error.

3. The bounds on the estimation error may be diminished by increasing the number of examples L in the data set.

4. Increasing the input dimension I affects negatively to the expected risk, as a sign of the *curse of dimensionality* [Bellman 1961].

Lower bounds on the expected risk and the sample complexity can also be found in the literature (see [Baum and Haussler 1989; Kearns and Schapire 1990, 1994; Devroye and Lugosi 1995; Bartlett 1998], for example). These bounds also depend on some measure of the flexibility of the model. In this case, increasing the flexibility of the model also increases the lower bounds on the expected risk or the sample complexity.

Regarding the flexibility of FNNs, the universal approximation capability implies that their VC-dimension and fat-shattering dimension are infinite in the general case.

However, they are finite if some of the parameters of the model are constrained, such as the number of hidden units, the magnitude of the weights or the type of the activation function (see [Maass 1995; Bartlett and Williamson 1996; Sontag 1998; Bartlett 1998; Bartlett and Maass 2003], for example).

Note that upper bounds take the form of inequalities. Lower bounds are the result of the worst-case analysis. Therefore, these theoretical results do not imply that an algorithm that chooses the output function in a very flexible hypothesis class will have bad generalization performance.

2.5.3 The Bias/Variance Decomposition

Despite certain obvious differences, results in the previous section are in analogy to the Bias/Variance decomposition [Geman et al. 1992; Domingos 2000a]. Knowing that the minimum expected risk of the quadratic error is achieved for $E_Y[y|x]$ (see (2.6)), and taking into account that it is independent on the data, it is possible to derive

$$\begin{aligned} E_D\left[(f_D^o(x) - E_Y[y|x])^2\right] = \\ &= (E_D[f_D^o(x)] - E_Y[y|x])^2 + E_D\left[(f_D^o(x) - E_D[f_D^o(x)])^2\right] \\ &= \mathcal{L}^2\left(E_D[f_D^o(x)], E_Y[y|x]\right) + E_D\left[\mathcal{L}^2\left(f_D^o(x), E_D[f_D^o(x)]\right)\right], \end{aligned} \tag{2.19}$$

where $E_D[\cdot]$ represents the expectation with respect to all the possible data sets D of fixed sample size L, and f_D^o is the output function obtained by the learning algorithm with the training set D. The first term in the right hand of (2.19) is the (squared) *bias* of $E_D[f_D^o(x)]$ as an estimator of $E_Y[y|x]$, whereas the second is the *variance* of $f_D^o(x)$ with respect to the ensemble of possible data sets.

Putting (2.6) and (2.19) together we have

$$\begin{aligned} E_D\left[R(\mathcal{L}^2, f_D^o)\right] &= E_{X\times Y}\left[(y - E_Y[y|x])^2\right] + E_X\left[E_D\left[(f_D^o(x) - E_Y[y|x])^2\right]\right] \\ &= E_X\left[E_Y\left[\mathcal{L}^2(y, E_Y[y|x])\right]\right] + \\ &\quad E_X\left[\mathcal{L}^2(E_D[f_D^o(x)], E_Y[y|x])\right] + \\ &\quad E_X\left[E_D\left[\mathcal{L}^2\left(f_D^o(x), E_D[f_D^o(x)]\right)\right]\right]. \end{aligned} \tag{2.20}$$

This result is a particular case of the unified Bias/Variance decomposition for the quadratic and the 0/1 loss functions [Domingos 2000a,b], where it is proved that for certain loss functions we have

$$E_D[R(\mathcal{L}, f_D^o)] = E_X[C_1(x)N(x)] + E_X[B(x)] + E_X[C_2(x)V(x)], \tag{2.21}$$

where

1. f_D^o is the output function obtained by the learning algorithm with the training set D.

2. The *Noise* $N(x) = E_Y[\mathcal{L}(y, f^*)]$, where $f^*(x) = \text{argmin}_f E_Y[\mathcal{L}(y, f)]$ is the *optimal prediction*, the closest function to the target in the hypothesis class. The noise is the unavoidable component of the error, regardless of the learning algorithm. For the quadratic loss function, $f^*(x) = E_Y[y|x]$. For two-class problems with the 0/1 loss function, $f^*(x)$ is given by the Bayesian posterior class probabilities.

3. The *Bias* $B(x) = \mathcal{L}(f^m, f^*)$, where $f^m(x) = \text{argmin}_f E_D[\mathcal{L}(f, f_D^o)]$ is the *main prediction*, the function whose average loss relative to the output functions $f_D^o(x)$ is minimum. The bias is the loss of the main prediction as an estimator of the optimal prediction. For the quadratic loss function, $f^m(x) = E_D[f_D^o(x)]$. For two-class problems with the 0/1 loss function, $f^m(x)$ is the mode (the most frequent prediction) of the output functions.

4. The *Variance* $V(x) = E_D[\mathcal{L}(f^m, f_D^o)]$ is the average loss of the output functions $f_D^o(x)$ relative to the main prediction.

5. $C_1(x)$ and $C_2(x)$ have different expressions for different loss functions:

   (a) For the quadratic loss function, (2.20) is obtained by taking $C_1(x) = C_2(x) = 1$.

   (b) For two-class problems with the 0/1 loss function, (2.21) holds with $C_1(x) = 2P_D(f_D^o(x) = f^*(x))$ and $C_2(x) = (1 - 2B(x))$. P_D is the probability over data sets D of fixed sample size L.

Similar results are proved in [Domingos 2000a,b] for multiclass problems with the 0/1 loss functions, and for two-class problems with other loss functions satisfying several weak conditions. Other decompositions of the 0/1 loss functions can be found in [Kong and Dietterich 1995; Kohavi and Wolpert 1996; Tibshirani 1996; Breiman 1996b; Friedman 1997]. Finding a decomposition for the multiclass case with functions different from the 0/1 loss function is an open problem. Similarly, finding a decomposition for regression problems with loss functions such as $\mathcal{L}(u, v) = |u - v|$ is another open problem.

For the 0/1 loss functions, the main consequence of a decomposition as in (2.21) is the non-strict additive behavior of bias and variance. For example, when $f^*(x) = f^m(x)$, we have $B(x) = 0$, and therefore $C_2(x) = 1$. In this case bias and variance have an additive behavior as for the quadratic loss function. In contrast, when $f^*(x) \neq f^m(x)$, it holds that $C_2(x) = -1$. In this case generalization performance increases by increasing the variance. Hence, the increase in average loss caused by

variance on correctly classified examples may be partly compensated by its decrease on incorrectly classified ones. It may help to explain why very simple methods for classification are often competitive and sometimes superior to more sophisticated ones [Friedman 1997; Sargent 2001].

Either high bias or variance can contribute to poor performance. Typically, very flexible models can nearly fit every point in every data set, leading to unbiased estimators but probably with high variance. A rigid class of hypothesis, in contrast, may lead to small variance but high bias (a constant function is the extreme case). By limiting the flexibility, the fitting capability can be lost. There is a trade-off between the bias and variance contributions to the error, where the optimal performance is achieved. In this sense, bias is related to the approximation error and variance to the estimation error. As an example, the number of hidden units and the size of the weights, the VC-dimension, the fat-shattering dimension and the 1-norm of the coefficients control the bias and variance contribution in (2.13), (2.14), (2.15), (2.16), (2.17) and (2.18) respectively. Note that the Bias/Variance decomposition takes the form of equality, different from the aforementioned bounds on the expected risk, which take the form of inequalities. In contrast, it gives no information about how to control both quantities. Anyway, the number of examples in the data set and the input dimension, related to the curse of dimensionality, play a crucial role for a proper estimation of the parameters.

Several schemes can be found in the literature related to the Bias/Variance trade-off. Some of them try to reduce the variance term while keeping a low bias. Other ones try to reduce the bias term with a limited flexibility. The rest try to control both terms at the same time. Although many of them were developed in an independent way, they can be seen in terms of the Bias/Variance decomposition. Some of them are general and others are specific for FNNs (the following list does not aim to be an exhaustive one):

1. Feature Selection [Kittler 1978; Liu and Motoda 1998].

2. Models related to Regularization Theory [Tikhonov and Arsenin 1977], such as Regularization Networks [Poggio and Girosi 1990; Girosi et al. 1995].

3. Statistical Learning Theory [Vapnik and Chervonenkis 1971; Vapnik 1982, 1995, 1998a] and Support Vector Machines [Boser et al. 1992; Cortes and Vapnik 1995].

4. Sparse representations [Daubechies 1992; Chen et al. 1998; Graepel et al. 2000].

5. Bayesian Neural Networks [MacKay 1992].

6. Sequential approximations, also named constructive, growing or incremental [Kwok and Yeung 1997a].

7. Pruning algorithms for FNNs [Reed 1993].

8. Specific heuristics for FNNs, such as

   (a) Weight decay [Hinton 1987; Krogh and Hertz 1992].

   (b) Early stopping [Morgan and Bourlard 1990; Prechelt 1998].

   (c) Training with noise [Sietsma and Dow 1991; Grandvalet et al. 1997].

   (d) Soft weight sharing [Nowlan and Hinton 1992].

9. Ensembles of estimators and voting methods, such as

   (a) Committees [Hansen and Salamon 1990; Perrone and Cooper 1993].

   (b) Mixtures of experts [Jacobs et al. 1991; Jordan and Jacobs 1994].

   (c) Boosting [Schapire 1990; Schapire and Singer 1999].

   (d) Bagging [Breiman 1996a].

Some of these schemes have been proved to be related among them (see, for example, [Bishop 1995b; Breiman 1998; Poggio and Girosi 1998; Girosi 1998; Smola et al. 1998; Evgeniou et al. 2000; Xu et al. 2001; Rätsch et al. 2002; András 2002]). In addition, they can be combined [Raviv and Intrator 1996; Avnimelech and Intrator 1999; Intrator 1999].

The next sections are devoted to describe the three frameworks where this thesis is placed: Sequential Approximations with FNNs, Feature Selection with MLPs and Margin Maximization, with special attention to their relationships with the Bias/Variance decomposition.

2.6 Sequential Approximations with FNNs

Sequential approximations (also named constructive, growing or incremental) are iteratively constructed by adding terms (usually one at a time) to the previously obtained partial approximations. Regarding the terminology, there exists some ambiguity on the adjective "sequential", since it is used in several frameworks to express different concepts. The same happens with the adjectives "constructive", "growing" and "incremental". We will use the word "sequential" for approximations that sequentially add terms to the previously obtained partial approximations. We will focus on sequential approximations with FNNs.

The selection of the optimal number of hidden units for FNNs has been widely discussed through the literature. In terms of the Bias/Variance decomposition, as far as the number of hidden units of an FNN grows, bias decreases and variance increases. As pointed out in [Geman et al. 1992], this happens because the flexibility

of the model grows with the number of hidden units. Theoretical results like (2.13) and (2.14) also support this claim (see section 2.5). Although the number of hidden units is not the only factor involved in the resulting generalization performance[5], it remains desirable to find solutions with a number of parameters as small as possible, as pointed out in [Lawrence et al. 1996, 1997]. Therefore, choosing an adequate architecture is a very important point in order to obtain good generalization performance.

Sequential approximations with FNNs allow to dynamically construct the network, starting from scratch, without setting *a priori* the architecture [Kwok and Yeung 1997a]. These methods start with a small network (usually with no hidden units), and sequentially add hidden units until a satisfactory solution is found. They can help to find a proper trade-off between bias and variance by controlling the number of parameters.

Pruning methods are an alternative to sequential ones [Reed 1993]. Pruning algorithms work roughly as follows. A larger than needed network is trained until an acceptable solution is found. Subsequently, some hidden units or weights are removed (pruned) if they are considered useless. Then, the network may be retrained, and the process starts again.

The sequential approach, however, presents a number of advantages over the pruning approach [Kwok and Yeung 1997a]. First, it is straightforward to specify an initial network for sequential algorithms, whereas for pruning algorithms it is not clear how large the initial network should be. Second, pruning algorithms spend most of the time training networks larger than necessary, whereas sequential algorithms always search for small solutions first. Therefore, sequential methods are more likely to obtain smaller networks with less computational cost than pruning methods.

Of course, sequential algorithms also have shortcomings. For example, it is not clear when to stop the addition of hidden units. Obtaining the weights of the added hidden units is also a difficult non-linear optimization problem [Horst and Tuy 1993]. In addition, it is well known that greedy approaches may be suboptimal in many cases. If the first hidden units are not properly chosen, the rest of the process will be negatively affected.

In spite of these shortcomings, sequential approximations can be very interesting in several ways:

1. The non-linear optimization problem posed at every step in sequential approximations is, in theory, easier to solve than for a non-sequential one [Huber

[5] The activation functions, the complexity of the target function, the noise level in the data or the number of training patterns also play an important role (see [Martin and Pittman 1991; Prechelt 1994; Lawrence et al. 1996, 1997; Caruana et al. 2001], for example).

1985; Jones 1992; Kürková 1998], since they take place in a lower dimensional space.

2. Although it cannot be guaranteed that the solution obtained is minimal, it is expected to have few terms. Approximations with a low number of terms are cheaper to store and compute.

3. Different activation functions can be chosen at every step, so that the network adapts its architecture to the specific target.

4. The system can be controlled by monitoring the training process after every new term is added: error decreasing on the training set, performance on an independent validation set, etc.

5. It is possible to save the parameters of the intermediate steps of the training and recover them if desired (to test, for example, a different training strategy).

We will mainly concentrate on sequential methods for regression. A number of other constructive methods that can *only* be applied to classification problems will not be discussed here. Some surveys on these methods can be found in [Parekh et al. 1997, 2000]. Indeed, classification problems can be considered as particular cases of regression problems.

Sequential approximations with FNNs cannot be described without referring to other sequential methods for regression found in the literature. In Statistics and Signal Processing literature, for example, several sequential schemes related to FNNs have been described. The following sections are devoted to a summary of sequential methods for regression with (or related to) FNNs. These methods are related to *SAOCIF*, the sequential algorithm that we propose in chapter 3. Some of them were originally designed for pure approximation purposes and others for generalization, but we will not make any distinction among them. Our aim has been to carry out a review as complete as possible. The original notations of the described works have been maintained as much as possible. A summary and discussion of several important aspects of the described schemes are included in section 2.6.8.

2.6.1 Dynamic Node Creation

The **Dynamic Node Creation** (DNC) method [Ash 1989] is a simple sequential method where, during the training, a new hidden node is added when the rate of decrease of the average squared error is less than a certain value. More precisely, a new node is added if both of the following conditions are satisfied:

$$t - w \geqslant t_0 \qquad \frac{e_t - e_{t-w}}{e_{t_0}} < \Delta$$

where e_t is the average squared error at time t. The width of the window w and Δ are defined *a priori*. The value t_0 is the time where the last node was added to the hidden layer (with an initial value of 0). The new node receives complete connections from the inputs, it is connected to all outputs, and its weights are initialized to small random values. The rest of the weights are not initialized, so that the obtained weights prior to the addition of the new hidden unit are the initial guess to train the next network. After a new node is added, the whole network is trained with standard BP until the solution is satisfactory or another node is needed.

Several variants of the original DNC procedure can be found in the literature. The main differences lie on:

1. The existence of a subsequent pruning procedure of hidden units [Hirose et al. 1991; Bartlett 1994; Hernández and Fernández 2002]. Hidden units or layers may also be pruned during the construction of the network [Nabhan and Zomaya 1994] (in this case, new hidden layers can also be added between the last hidden layer and the output one).

2. The algorithms used to train the network [Bello 1992; Azimi-Sadjadi et al. 1993; Setiono and Hui 1995].

3. The type of hidden units [Bartlett 1994; Shin and Ghosh 1995; Leerink et al. 1995].

4. The number of hidden units of the starting network [Bastian 1994].

5. The criteria to add new hidden units [Wang et al. 1994].

6. The addition of patterns to the training set during the training process [Zhang 1994; Chung and Lee 1995].

7. The different ways of connecting the new hidden unit to the previously existing ones [Vinod and Ghose 1996].

Convergence to the target function of these methods follows directly from the universal approximation property of the underlying architecture [Kwok and Yeung 1997a].

We have not found, in the revised literature, either a comparative study among these modifications of the DNC procedure or other sequential schemes. When compared with fixed-architecture MLPs trained with gradient descent, they usually obtain similar or better performance with faster convergence, specially when the resulting networks are small.

2.6.2 Meiosis Networks

The main idea behind **Meiosis Networks** (MN) [Hanson 1990] is that the changes in the standard deviation of the weight distribution can be taken as an uncertainty measure of the weight strength. Hidden units with significant standard deviation are not desired (in [Hanson 1990], a weight is a random variable with a finite mean and variance, adjusted with a "stochastic delta rule"). When this situation occurs during the learning process, the unit splits into two units each copying half the variance to each of the new units. Different strategies to split the hidden units are applied in several subsequent works:

1. In [Wynne-Jones 1992], Principal Component Analysis is used to compute the direction and size of the weights of the new hidden units.

2. In [Khorasani and Weng 1994; Weng and Khorasani 1996] the unit with the highest fluctuation rate is splitted. The fluctuation rate is the product of the differences between weights and output values. Several units can be splitted at a time.

A similar procedure is described in [Sanger 1991a,b], where the learning procedure constructs a tree using unidimensional functions of fixed frequencies, based on the hypothesis that successful approximations will not always require all the dimensions of the input data. Additional input dimensions are incorporated, in principle, only when needed. In order to work, the activation functions used must be separable (i.e., they can be written as a product of unidimensional functions). The large variance of a weight is decreased by inserting a subnetwork at that point. The insertion of every subnetwork can, in principle, create a new hidden layer.

Similar to the DNC scheme, convergence to the target function of these methods follows directly from the universal approximation property of the underlying architecture.

The experiments performed with these models, although they are not very exhaustive, allow to reduce the training time with respect to gradient descent in fixed-architecture MLPs.

2.6.3 Resource-Allocating Network

In [Platt 1991], the **Resource-Allocating Network** (RAN) is described. When the network performs well on a presented pattern, the whole network is trained. Otherwise, a new localized RBF unit, centered on that pattern, is allocated. Therefore, a specialization process results, in some sense, when the network does not perform well on a presented pattern. Due to this learning strategy, RAN is well suited for on-line learning. The adjustment of the parameters (coefficients, output bias and

centers) between the addition of two hidden nodes is performed by gradient descent (for the coefficients and the output bias, the Least Mean Squares algorithm is used). The width of the RBF unit is fixed *a priori*.

A very similar algorithm can be found in [Lee and Kil 1991]. The use of Gaussian RBFs and the adjustment of all the parameters of the network (including the widths of the RBF units) with BP are the main differences with respect to the RAN algorithm.

Independently, a similar method named **Growing Cell Structures** was developed in [Fritzke 1994a,b], combining an unsupervised self-organizing neural network model with the RBFN approach.

Several modifications of the original RAN algorithm have been proposed (all of them are specific for RBFNs):

1. Replacing the Least Mean Squares algorithm by another one based on the Extended Kalman Filter algorithm [Kadirkamanathan and Niranjan 1993; Jankowski and Kadirkamanathan 1997], Lyapunov techniques [Liu et al. 1996, 1999] or the QR decomposition [Rosipal et al. 1998; Salmerón et al. 2001].

2. Replacing the multi-dimensional Gaussian basis functions by a layer of one-dimensional Gaussian functions and a layer of Pi-neurons [Deco and Ebmeyer 1993].

3. Modifying the growth criterion of the network, replacing it by a statistical criterion under Gaussian assumptions [Kadirkamanathan 1994; Jankowski and Kadirkamanathan 1997].

4. Including a pruning procedure [Jankowski and Kadirkamanathan 1997; Yingwei et al. 1997a,b; Rosipal et al. 1998; Liu et al. 1999; Salmerón et al. 2001; Alexandridis et al. 2003].

5. Adapting the algorithm to time-series prediction, by modifying dynamically the number of inputs considered [Salmerón et al. 2001].

6. Allowing direct connections between the input and output layer [Lee and Street 2003].

7. Adjusting the learning parameters to the partially constructed network [Lee and Street 2003].

8. Selecting the frequencies among the points of a variable grid [Liu et al. 1996, 1999] or the centers of a fuzzy partition of the input space into a number of subspaces [Alexandridis et al. 2003]. In [Lee and Street 2003], some hidden units are initialized to predefined values, specific to the problem at hand.

As pointed out in [Kwok and Yeung 1997a], the convergence properties of these algorithms are unknown.

Several comparisons of some of these variants of the RAN procedure can be found in [Kadirkamanathan and Niranjan 1993; Kadirkamanathan 1994; Yingwei et al. 1997a; Rosipal et al. 1998; Salmerón et al. 2001]. In general, the proposed modifications allow to obtain simpler models (in terms of the number of hidden units) with better generalization performance than the original RAN procedure and the previously described modifications. In [Platt 1991; Alexandridis et al. 2003], the RAN algorithm compares favorably with standard RBFN models. There exist, to our knowledge, a lack of comparative results with standard MLPs and other sequential schemes.

An important property of these models is that they can be usually trained with little computational effort.

2.6.4 Cascade Correlation

The most celebrated sequential method for Neural Networks is, probably, **Cascade-Correlation** (CC) [Fahlman and Lebiere 1990]. CC combines two key ideas. The first is the cascade architecture, in which the newly added hidden unit receives inputs from both the input layer and the previously added hidden units. The second is the learning algorithm. For each new hidden unit, the algorithm tries to maximize the correlation between the output of the new unit and the residual error of the network.

The input weights of the hidden unit are frozen at the time the new unit is added to the network. Only the output connections are trained repeatedly (output units may have non-linear activation functions). The learning algorithm works as follows. It begins with a network without hidden units, and the direct input-output connections are trained over the training set. To create a new hidden unit, it begins with a candidate unit that receives trainable input connections from all of the inputs of the network and from all the pre-existing hidden units. The output of this unit is not yet connected to the active network. The input weights of the candidate unit are adjusted to maximize the correlation (or, more precisely, the covariance)

$$S = \sum_o |Cov\,(E_o, H)| = \sum_o \left| \sum_p (E_{p,o} - \overline{E_o})(H_p - \overline{H}) \right|$$

where o varies over all the output units, p varies over the training patterns, $E_{p,o}$ is the residual error observed at unit o (without the candidate unit, which is not yet connected), and H_p is the activation of the candidate. The quantities $\overline{H}$ and $\overline{E_o}$ are the values of H_p and $E_{p,o}$ averaged over all patterns. In order to maximize S, a gradient ascent is performed. Once again, only a single layer of weights is trained. When S is considered to be maximized, the candidate is installed as a hidden unit,

its input weights are frozen, and all the output layer connections are trained over the training set. The cycle continues until the performance of the network is satisfactory. Instead of a single candidate unit, it is possible to use a pool of candidate units, each with a different set of random initial weights. Alternatively, the candidates might have different non-linear activation functions, and let them compete to be chosen for addition to the active network.

In [Prechelt 1997], two main drawbacks of CC are pointed out:

1. The covariance may be an ill-suited objective function for training the candidates. Maximizing covariance trains candidates to have a large activation (large deviation from average activation) whenever the error at their output is not equal to the average error.

2. Cascading the hidden units results in a network that can represent very strong non-linearities. Although this power is in principle useful, it can be a disadvantage if such strong non-linearity is not required to solve the problem.

There exist many variants in the literature of the original CC procedure. Some of them are related to the two drawbacks exposed in [Prechelt 1997]:

1. The optimization of a function different from the covariance, usually the sum-of-squares error [Littmann and Ritter 1992a,b; Lehtokangas 1999, 2000; Islam and Murase 2001] or a related function [Courrieu 1993; Lahnajärvi et al. 1999].

2. Modifications of the learning architecture (with a covariance-based optimization function):

   (a) Not allowing cascaded connections, either maintaining the direct connections between the input and output layers [Yeung 1991; Sjøgaard 1992; Yeung 1993] or not [Ma and Khorasani 2000, 2003, 2004]. New hidden layers also can be added in [Ma and Khorasani 2003].

   (b) Modifying the type of the hidden units [Littmann and Ritter 1992a; Leerink et al. 1995].

   (c) Allowing the addition of several units in the same layer [Littmann and Ritter 1992b; Fang and Lacher 1994; Mohraz and Protzel 1996].

   (d) Restricting the depth and the connectivity of the architecture [Phatak and Koren 1994; Mohraz and Protzel 1996; Treadgold and Gedeon 1998; Islam and Murase 2001].

   (e) Adapting the previously added hidden units, or even removing some of them depending on the evolution of the learning process [Treadgold and Gedeon 1998; Lehtokangas 1999, 2000].

(f) Adapting the architecture specifically to regression problems [Lehtokangas 2000].

3. Representing the hidden units in the transformed domain of the coefficients which constitute an orthogonal predefined basis [Vyšniauskas et al. 1995].

4. Scaling the network error to the range of sigmoidal units [Ma and Khorasani 2000, 2004].

5. Including a pruning procedure [Smotroff et al. 1991; Islam and Murase 2001; Ma and Khorasani 2000, 2004; Liang and Ma 2004].

6. Incorporating regularization [Kwok and Yeung 1996a; Treadgold and Gedeon 1999a].

7. Retraining the whole network after the addition of a hidden unit [Treadgold and Gedeon 1997a,b, 1999a; Islam and Murase 2001].

8. Using a genetic algorithm to find the initial weights [Liang and Dai 1998].

9. Adding the new hidden units by splitting existing units [Islam and Murase 2001].

In [Kwok and Yeung 1997b] the convergence property of CC is proved, under certain assumptions, with similar ideas to the theoretical results of convergence for Projection Pursuit Regression (see section 2.6.5).

Some experimental studies have shown that the idea of maximizing the correlation tends to produce saturate units [Hwang et al. 1996]. Moreover, the decision boundary may be very zigzag and unsmooth. This makes the CC algorithm more suitable, in principle, for classification problems than for regression ones. Similar to DNC, these models usually obtain smaller networks with similar or better performance and faster convergence than fixed-architecture MLPs.

A comparison of some of these variants for CC can be found in [Prechelt 1997; Treadgold and Gedeon 1999b; Lahnajärvi et al. 2002], for example. Results in [Prechelt 1997] suggest that error minimization is usually superior to correlation maximization. In addition, not cascading hidden units often allow to obtain better results than cascading them. When these two properties hold (error minimization without cascading hidden units), the resulting method is similar to Projection Pursuit Regression (see section 2.6.5). In [Treadgold and Gedeon 1999b] it was observed that, for cascade architectures, the combination of early stopping and regularization resulted in better generalization performance and smaller networks than the use of early stopping alone. Results in [Lahnajärvi et al. 2002] show similar performance for the models tested, although it was pointed out again that error minimization is usually superior to correlation maximization.

2.6.5 Projection Pursuit Regression: Matching the Residue and Convex Approximations

Projection Pursuit Regression is a method for sequential approximation based in the approximation of the previous residue. It was originally described in the Statistics literature, although several works with the same underlying ideas can also be found in the areas of Signal Processing and FNNs. Previous to describe FNN models inspired in Projection Pursuit Regression (section 2.6.5.3), the most relevant works in Statistics and Signal Processing are described (sections 2.6.5.1 and 2.6.5.2). Finally, section 2.6.5.4 describes several objective functions that can be used with the aim of matching the residue.

2.6.5.1 Projection Pursuit Regression in Statistics

Projection Pursuit is a family of optimization methods described in the Statistics literature. Its name is derived from the fact that the data are linearly projected onto several interesting directions, which are selected to maximize a certain objective function. The notion of interesting projections is motivated by the observation that for most high-dimensional data clouds, most low-dimensional projections are approximately normal [Diaconis and Freedman 1984]. Hence, a projection is less interesting the more nearly normal it is, and the discovery of interesting projections may lead to obtain useful information. An early version of Projection Pursuit for clustering was implemented in [Friedman and Tukey 1974]. These ideas were extended to **Projection Pursuit Regression** (PPR) [Friedman and Stuetzle 1981], Projection Pursuit Density Estimation [Friedman and Stuetzle 1984] and Projection Pursuit Classification [Friedman 1985].

A detailed description of Projection Pursuit can be found in [Huber 1985]. In general, given a random variable X with values in $\mathbb{R}^d$, the methods based in Projection Pursuit search for a linear projection A optimizing an objective function $Q(F_A)$, where F_A is the distribution of the random variable $A \cdot X$. A linear projection $A : \mathbb{R}^d \to \mathbb{R}^k$ is any linear map (or equivalently a $k \times d$ matrix) of rank k. By changing the objective function $Q(F_A)$, the particular Projection Pursuit methods are obtained. Projection Pursuit generalizes classical methods in multivariate analysis (Principal Components and Discriminant Analysis) and in factor analysis (the Quartimax and Oblimax methods, for example). Finally, it allows to create new families of penalty terms by combining, for example, unsupervised and supervised training, as in [Intrator 1993]. As a drawback, they use to be computationally high-demanding. Due to this computational cost, and to the interest in getting an ordered set of projections, stepwise methods like PPR become very attractive.

As a particular case of function approximation, PPR [Friedman and Stuetzle 1981] estimates the conditional expectation of a random variable $Y \in \mathbb{R}$ given $X \in \mathbb{R}^I$

by means of a sum of ridge functions

$$E[Y \mid X = x] = f(x) \cong \sum_{j=1}^{N} g_j(a_j^t \cdot x)$$

with the quadratic loss function $Q(F_A) = E\left[\left(f(x) - \sum_{j=1}^{N} g_j(a_j^t \cdot x)\right)^2\right]$ as the objective function as follows. Let $f_0(x) = 0$ and suppose that the first $n-1$ terms of the approximation (the frequencies a_j and the functions g_j) have already been determined. Let

$$f_{n-1}(x) = \sum_{j=1}^{n-1} g_j(a_j^t \cdot x)$$

be the approximation at step $n-1$, and define $g_{j,a_j}(x) = g_j(a_j^t \cdot x)$. Find a_n and g_n such that $E\left[(r_{n-1} - g_{n,a_n})^2\right]$ is the minimum, where $r_{n-1}(x) = f(x) - f_{n-1}(x)$ is the residue at step $n-1$. Then, $f_n(x) = f_{n-1}(x) + g_n(a_n^t \cdot x)$. This process is repeated until the residue is smaller than a user-defined threshold. That is, at every step the residue is approximated as best as possible with a single term, which is added to the solution obtained at the previous step.

Some properties of the approximations with PPR are discussed in [Diaconis and Shahshahani 1984]. They studied necessary and sufficient conditions for functions to be exactly represented as linear combinations of ridge functions and the uniqueness of that representations.

In the original definition of PPR [Friedman and Stuetzle 1981], the approximation is defined from a set of observations (*discrete version*). For a given a_n, the function g_n is non-parametrically constructed from the scatterplot of r_{n-1} against $a_n^t \cdot X$, so that g_n is smooth and fits the scatterplot. This process may be improved by *back-fitting*: omit some of the earlier summands g_j, determine better replacements, and then iterate. Usually, the directions a_j during the *back-fitting* process are kept fixed.

In the *abstract version*, the function itself is available instead of just a set of observations [Huber 1985]. In order to be well-defined, $f \in L^2$ and the integral is defined with regard to a probability measure, so that $E[h^2] = \|h\|^2$. In this case it can be proved that, for a fixed a_n, the function g_n minimizing $E\left[(r_{n-1} - g_{n,a_n})^2\right]$ is

$$g_n(z) = E\left[r_{n-1}(X) \mid a_n^t \cdot X = z\right], \tag{2.22}$$

and

$$E\left[(r_{n-1} - g_{n,a_n})^2\right] = E\left[r_{n-1}^2\right] - E\left[g_{n,a_n}^2\right]. \tag{2.23}$$

Thus, the problem at every step is to find the minimizing direction a_n (or, equivalently, a maximizing direction for $E\left[g_{n,a_n}^2\right]$). In [Huber 1985] it is conjectured that,

under mild smoothness conditions, the minimizing direction a_n exists for every n, and $\lim_{n\to\infty} E[r_n^2] = 0$ (i.e., PPR has the convergence property). In [Jones 1987] those conditions are generalized: given $0 < \rho < 1$, let a_n be any direction such that

$$E\left[g_n^2(a_n^t \cdot X)\right] > \rho \sup_{b^t \cdot b = 1} E\left[g_n^2(b^t \cdot X)\right],$$

with g_n as defined in (2.22). Then, PPR converges to f. The convergence property of Matching Pursuit ([Mallat and Zhang 1993], see below) is based on this result.

Later, in [Jones 1992] it was proved that the convergence may be accelerated if the approximation is made with a convex combination. Given a function set P_n and a sequence of functions f_n, we can define

$$s_n = \inf_{0 \leqslant \alpha \leqslant 1,\ \phi \in P_n} \|f - ((1-\alpha)f_n + \alpha\phi)\|.$$

A sequence of functions f_n is said to be *relaxed* with respect to f if $\|f - f_{n+1}\| \leqslant s_n$, and *asymptotically relaxed* if $\limsup(\|f - f_{n+1}\| - s_n) \leqslant 0$. Clearly, a relaxed sequence is asymptotically relaxed. In [Jones 1992] it is proved that any asymptotically relaxed sequence converges to f, under reasonable convexity conditions. In addition, if f lies on the closure of the convex hull of some bounded function set G ($\|g\| \leqslant M$ for every $g \in G$), and $G \subseteq P_n$ for every n, the approximation error of a relaxed sequence f_n is $O(1/\sqrt{n})$. The later derived upper approximations bounds for FNNs in [Barron 1993] or [Kúrková and Beliczyński 1995b] are based on this result. With these ideas, a *relaxed* variant of PPR is defined

$$f_n(x) = (1-\alpha_n)f_{n-1}(x) + \alpha_n g_n(a_n^t \cdot x),$$

where P_n is, for every n, the set of all ridge functions, $f_0(x) = 0$ and α_n, a_n and g_n minimize $\|f - ((1-\alpha)f_n + \alpha g_{n,a}\|$ at every step.

In practice, once g_n has been determined, the previous expression is minimized over α and a (usually in a non-linear manner). Moreover, in [Jones 1992] it is suggested that the coefficients of the previously selected terms can be optimized after g_n and α_n have been selected. Several variants to construct the function g_n can be found in the literature, both parametric and non-parametric, in different areas [Flick et al. 1990; Zhao and Atkeson 1991; Saha et al. 1993; Verkooijen and Daniels 1994; Roosen and Hastie 1994; Zhao and Atkeson 1996]. For FNNs, g_n is fixed *a priori*.

2.6.5.2 Projection Pursuit Regression in Signal Processing

Some methods with the same underlying ideas as PPR can be found in the area of Signal Processing. The probably most celebrated one is the **Matching Pursuit** (MP) algorithm, described in [Mallat and Zhang 1993]. MP decomposes any signal

into a linear expansion of waveforms that are selected from a (possibly redundant) dictionary of functions. Given a set of parameters (frequencies) , , a dictionary is defined as a family $\mathcal{D} = (g_\gamma)_{\gamma\in\Gamma}$ of vectors in a certain Hilbert space H, such that $\|g_\gamma\| = 1$ for all $\gamma\in$, . The theoretical results proved in [Mallat and Zhang 1993] are general in the sense that they can be applied to any vector f in H. The MP algorithm works roughly as follows. Let $R^0 f = f$, suppose that f_n is the approximation at step n ($f_0 = 0$) and the nth order residue $R^n f$ is already computed, for $n \geqslant 0$ ($R^n f = f - f_n$). Choose an element $g_{\gamma_n} \in \mathcal{D}$ which matches the residue $R^n f$ almost as much as the best, that is[6]

$$|\langle R^n f, g_{\gamma_n}\rangle| \geqslant \alpha \sup_{\gamma\in\Gamma} |\langle R^n f, g_\gamma\rangle|\,, \tag{2.24}$$

where α is an optimality factor satisfying $0 < \alpha \leqslant 1$. The residue $R^n f$ is decomposed into

$$R^n f = \langle R^n f, g_{\gamma_n}\rangle\, g_{\gamma_n} + R^{n+1} f,$$

so that

$$\|R^n f\|^2 = |\langle R^n f, g_{\gamma_n}\rangle|^2 + \left\|R^{n+1} f\right\|^2 \tag{2.25}$$

and

$$f = \sum_{i=0}^{n} \left\langle R^i f, g_{\gamma_i}\right\rangle g_{\gamma_i} + R^{n+1} f.$$

Therefore, we can define the approximation at step $n+1$ as

$$f_{n+1} = \sum_{i=0}^{n} \left\langle R^i f, g_{\gamma_i}\right\rangle g_{\gamma_i} = f_n + \langle R^n f, g_{\gamma_n}\rangle\, g_{\gamma_n}\,.$$

That is, at every step the residue is approximated as much as possible (and a certain tolerance, defined in (2.24)) with a single term, which is added to the solution obtained at the previous step. Although there exist some differences with PPR (for example, the fact that a previously defined dictionary is needed in MP), the links between both methods are quite clear: r_n and g_{n,a_n} in PPR are $R^n f$ and $\langle R^n f, g_{\gamma_n}\rangle\, g_{\gamma_n}$ in MP, respectively. Therefore, (2.23) is equivalent to (2.25).

In [Mallat and Zhang 1993] it is proved that, if the dictionary $\mathcal{D}$ spans H, then $f = \sum_{i=0}^{\infty} \langle R^i f, g_{\gamma_i}\rangle\, g_{\gamma_i}$. If $\mathcal{D}$ is an orthogonal basis, the MP decomposition is equivalent to an orthogonal expansion of f in the basis $\mathcal{D}$. In practice, a recalculation of the coefficients can be made after n steps, named *back-projection* (a particular case of *back-fitting* in PPR), to approximate f at best with the selected vectors.

[6] The element which best matches the residue is $\sup_{\gamma\in\Gamma} |\langle R^n f, g_\gamma\rangle|$ (see Lemma 2 in chapter 3).

In practice, when we only have a data set, we can consider that the dimension of H is finite. A vector is an element of $\mathbb{R}^L$, where L is the number of patterns. In the same way, the dictionary is a set of vectors in $\mathbb{R}^L$, obtained by applying every g_γ to every point in the data set. If the dictionary is finite, several optimization strategies can be used. First, it is possible to find exactly the element g_{γ_n} that matches the residue as best as possible simply with a pass through the dictionary. In addition, this element may be optionally tuned searching for a better frequency γ'_n in its neighborhood. A Newton method, for example, can be used to that end. Finally, since

$$\left\langle R^{n+1} f, g_\gamma \right\rangle = \left\langle R^n f, g_\gamma \right\rangle - \left\langle R^n f, g_{\gamma_n} \right\rangle \left\langle g_{\gamma_n}, g_\gamma \right\rangle,$$

it is possible, at every step, to store $\left\langle R^n f, g_\gamma \right\rangle$ for every $g_\gamma \in \mathcal{D}$ so that the computations at next iterations can be made more efficiently.

An application of MP with a dictionary of Gabor time-frequency atoms for a signal processing task is described in [Mallat and Zhang 1993], showing the usefulness of the method.

Similar results are obtained in [Qian and Chen 1992, 1994], but with a particular set of functions: the normalized Gaussian functions with adjustable width and time-frequency center.

The approximations of a Matching Pursuit procedure can be improved by orthogonalizing the directions of projection [Pati et al. 1993]. The idea of the **Orthogonal Matching Pursuit** (OMP) procedure works as follows. The vector g_{γ_n} selected by MP is not necessarily orthogonal to the previously selected vectors, and therefore the algorithm reintroduces new components in the directions of $\{g_{\gamma_i}\}_{0 \leqslant i < n}$. To avoid this fact, the residue $R^n f$ is projected, after g_{γ_n} has already been selected as in MP, on a vector u_n (substituting but obtained from g_{γ_n}) orthogonal to the previously selected vectors. It can be done by solving a linear equations system, and it is equivalent to the Gram-Schmidt orthogonalization procedure. As a consequence, the coefficients of the selected vectors are optimal at every step, and the whole procedure is equivalent to perform MP with *back-projection* at every iteration. Bordered systems are used to solve efficiently the linear equations system. To expand f over the original dictionary vectors, a change of basis must be performed, inverting the orthogonalization procedure.

2.6.5.3 Projection Pursuit Regression with FNNs

PPR was first introduced in the context of FNNs by [Barron and Barron 1988], showing that the two layer architecture of a neural network is well suited to construct a PPR in a natural way.

A "direct implementation" of PPR with MLPs is described in [Moody 1994]. The new frequencies are trained, from small random values, to a local minimum while keeping the weights of the previous units fixed (i.e., the previous residue is

tried to be approximated as best as possible). After a new unit is added, the whole network is trained (*back-fitting*). In [Jutten and Chentouf 1995], a similar procedure is described.

The rest of the section is devoted to a description of two different learning schemes with FNNs based on PPR: the Projection Pursuit Learning Network and the Incremental Linear Quasi-Parallel algorithm.

The **Projection Pursuit Learning Network** (PPLN) described in [Hwang et al. 1994], following their previous work in [Maechler et al. 1990; Hwang et al. 1991, 1992a,b], is modeled as a two-layer (one hidden layer) fully connected MLP

$$\hat{y}_i = \overline{y}_i + \sum_{k=1}^{m} \beta_{ik} g_k \left(\sum_{j=1}^{p} \alpha_{kj} x_j \right),$$

with $\overline{y}_i = E_L[y_i]$, $E_L[g_k] = 0$, $E_L[g_k^2] = 1$ and $\sum_{j=1}^{p} \alpha_{kj}^2 = 1$. The parameters α_{kj} and β_{ik} are, as usual, the frequencies and the coefficients respectively, and g_k is the unknown (trainable) "smooth" activation function of the kth hidden unit. Output units have linear activation functions. The training of all the parameters is based on the criterion of minimizing the error function

$$L_2^W = \sum_{i=1}^{q} W_i \cdot E\left[(y_i - \hat{y}_i)^2\right],$$

where q is the number of output units. The weightings W_i allow to specify the relative contribution of each output to the total error.

A PPLN learns unit by unit, and layer by layer cyclically after all the training patterns are presented. All the parameters to be estimated are hierarchically divided into m groups (each associated with one hidden unit), and each group, say the kth group, is further divided into three subgroups: the coefficients $\{\beta_{ik} : i = 1, \cdots, q\}$, the smooth non-linear function g_k of the kth hidden unit, and the frequencies $\{\alpha_{kj} : j = 1, \cdots, p\}$. The PPLN starts by setting the parameters associated with the first hidden unit (i.e., the first group). It updates each subgroup, $\{\alpha_{1j}\}$, g_1 and $\{\beta_{i1}\}$ consecutively (layer by layer) to minimize the error function L_2^W. It then estimates the parameters associated with the second hidden unit by consecutively updating $\{\alpha_{2j}\}$, g_2 and $\{\beta_{i2}\}$. A complete updating pass ends when the parameters associated with the mth (the last) hidden unit are updated. Repeated updating passes are made over all the groups (i.e., *back-fitting* for every unit) until convergence. As it can be seen, PPLN is not, strictly speaking, a sequential method, since it needs to define a maximum number of hidden units *a priori*. In contrast, the weights are learned in a sequential manner and taking into account the residue at the previous step, as in PPR. The process stops when

$$\frac{\left| L_2^W(new) - L_2^W(old) \right|}{L_2^W(old)}$$

is smaller than a prespecified small constant in two consecutive cycles. The kth group parameters are estimated as follows. Least Squares is applied to estimate $\{\beta_{ik}\}$ (given g_k and $\{\alpha_{kj} : j = 1, \cdots, p\}$, L_2^W is quadratic in the $\{\beta_{ik}\}$), g_k is estimated by a one-dimensional data smoother, and a non-linear optimization algorithm (Gauss-Newton) estimates $\{\alpha_{kj}\}$. Each layer weights are learned while the other layer weights remain fixed, and no global optimization of the coefficients is performed.

With the aim of optimizing the non-linearity degree and relaxing the necessity of predefining it, the Pooling Projection Pursuit Networks [Lay et al. 1994] use a pool of Hermite functions of several degrees during the training of a new candidate hidden unit. Some hybrid models can be found that attempt to take profit from the advantages of PPR and CC. In [You et al. 1994] the Cascaded Projection Pursuit Network is defined, which implements a PPLN with cascaded connections among the hidden units to allow high-order non-linearities.

A comparative study between PPLNs and standard FNNs was performed in [Hwang et al. 1994] with artificial problems. PPLNs were trained using as activation functions a non-parametric smoother named supersmoother [Friedman 1985] and parametric Hermite functions, based on the Hermite polynomials. Standard FNNs were trained with a Gauss-Newton optimization procedure. Both methods had quite comparable training speed, but PPLNs always outperformed standard FNNs on independent test data, specially with Hermite functions, when both use the same number of hidden units. When allowed to use different number of hidden units, both achieve comparable accuracy, but PPLNs are considerably more parsimonious in that fewer units are required to approximate the desired function. In addition, the frequencies may be very different between different simulations of standard FNNs with different number of hidden units, whereas PPLNs are more consistent in that sense. A comparison between PPLN and CC can be found in [Hwang et al. 1996], showing that the strong non-linearities generated by the cascade architecture may also be constructed with PPLNs trained with adequate parameters.

In [Kwok and Yeung 1995a,b, 1996b] it is proved that PPLNs with Hermite functions, as defined in [Hwang et al. 1994], do not have the universal approximation property. This is a consequence of the particular parametric approach. However, universal approximation capability can be achieved, in the same conditions as PPR, simply by including a bias term into each linear combination of the predictors, that is

$$\hat{y}_i = \overline{y}_i + \sum_{k=1}^{m} \beta_{ik} g_k \left(\sum_{j=1}^{p} \alpha_{kj} x_j + \theta_j \right).$$

Experiments in [Kwok and Yeung 1996b], with the same artificial problems as in [Hwang et al. 1994], confirmed the suitability of using biases. The proposed model also compared favorably with CC.

The **Incremental Linear Quasi-Parallel** (ILQP) algorithm, a sequential al-

gorithm for neural networks based on the ideas of PPR and MP was described in [Kůrková and Beliczyński 1995b]. A tolerance term is introduced in the definition of the approximation, giving an asymptotically relaxed approximation as defined in [Jones 1992]. Given a Hilbert space H, an *asymptotically optimal convex quasi-parallel incremental approximation of f with respect to $G \subseteq H$* is a sequence

$$f_n = (1 - \alpha_n) f_{n-1} + \alpha_n g_n$$

where $g_n \in G$ so that

$$\langle f - f_{n-1}, g_n \rangle + \varepsilon_n \geqslant \sup_{g \in G} \langle f - f_{n-1}, g \rangle$$

and

$$\|f - f_n\| \leqslant \inf_{\alpha} \|f - ((1 - \alpha) f_{n-1} + \alpha g_n)\| + \varepsilon_n$$

with $\lim_{n \to \infty} \varepsilon_n = 0$. An *asymptotically optimal linear quasi-parallel incremental approximation* is defined analogously, but allowing (with the same definition of g_n) complete flexibility of the coefficients at every step. Output units have linear activation functions. The first condition on $\langle f - f_{n-1}, g_n \rangle$ is similar to the condition (2.24) for MP in the sense of approximately matching the residue, and it is equivalent to say that g_n is as parallel as possible to $f - f_{n-1}$ with a certain tolerance.

The convergence property is satisfied by both definitions, if f lies on the closure of the convex hull of G. For the convex version, following [Jones 1992] and [Barron 1993], the approximation error is $O(1/\sqrt{n})$ if the vectors in the approximation are bounded ($\|g\| \leqslant B$ for every $g \in G$). For the linear version, it is pointed out that the same result holds. In the ILQP algorithm, following the linear definition, every iteration consists of two steps. In the first one, the frequency of a new hidden unit is determined. In the second one, all output weights are recalculated. The frequencies are calculated trying to find $g_n \in G$ so that nearly maximizes $|\langle f - f_{n-1}, g \rangle|$ over $g \in G$. This optimization step depends on the set of functions G and it can be done by any optimization method (gradient descent, for example). In the second step, output weights are optimized so that the error is minimized with the selected frequencies. Therefore, the resulting method is similar to OMP, where there exists a *back-projection* procedure at every iteration. A version for RBFNs with regularization can be found in [Kůrková and Beliczyński 1995a].

A refinement of the ILQP algorithm can be found in [Beliczyński 1996a,b], where an almost analytical incremental algorithm is described. The algorithm does not involve any searching method, but the hidden units must have an MLP activation functions invertible and bipolar (with range in $(-1, +1)$), such as the classical hyperbolic tangent function. Although the possibility of using different activation functions in the same network is suggested, it is not tested. Results in [Beliczyński 1996a,b] suggest that a *precise* maximization of $|\langle f - f_{n-1}, \phi\ (\omega \cdot x) \rangle|$ is not strictly

necessary. This idea can also be found in [Beliczyński and Kůrková 1997], where the frequencies are selected heuristically, with elliptic RBF Gaussian activation functions in the hidden units. The coefficients are optimal at every step.

Several variations of the ILQP algorithm can be found in the literature. All of them construct convex approximations, and obtain the new frequency trying to approximate the previous residue with a single term and a certain tolerance:

1. In [Draelos and Hush 1996], the frequencies are searched heuristically partitioning the surface of the target function (driven by the training set) into hyperplanar regions. Piecewise linear functions are used as activation functions in order to obtain the final convex combination.

2. A convex sequential procedure is described in [Dunkin et al. 1997], based on a theoretical result described in [Koiran 1994] and similar to that in [Kůrková and Beliczyński 1995b]. The new frequencies are obtained with BP. The main difference lies in the restriction that the coefficient of the new added term is bounded in advance.

3. A sequential algorithm for approximations of functions in the closure of finite subsets of a Hilbert space is defined in [Hlaváčková and Sanguinetti 1998]. The theoretical result is based on the concept of variation of a function with respect to a subset [Kůrková 1998], and is similar, in essence, to that of [Kůrková and Beliczyński 1995b]. The particular algorithm constructs a convex approximation, with a finite dictionary, whose approximation error is bounded by $O(1/\sqrt{n})$. A weight decay term is added to the cost function in [Hlaváčková and Fischer 2000].

Slightly different theoretical approaches can also be found, obtaining similar approximation error bounds. In [Dingankar and Sandberg 1996], α_n is fixed at every step to $1/n$, and the convex approximation is obtained by choosing g_n such that

$$\|f - f_n\| \leqslant \inf_{g \in G} \|f - ((1-\alpha_n) f_{n-1} + \alpha_n g)\| + \varepsilon_n.$$

In [Zhang 2002], the minimization is performed over g_n and α_n at the same time. No tolerance is allowed.

The experiments in these works are not very exhaustive. Many of them are more focused on theoretical results than in experimental ones. Most of the times, the experiments are performed without comparing the results with other existing methods. As an exception, the procedure described in [Dunkin et al. 1997] shows better generalization performance than CC, specially on noisy data sets.

2.6.5.4 Objective Functions with the Aim of Matching the Residue

In [Kwok and Yeung 1997b], a number of objective functions to obtain the new frequency are explored with the aim of matching the residue. In particular, it is proved that the convergence property is satisfied if the function g_n computed by the nth hidden unit maximizes one of the following objective functions:

- $S_1 = \frac{|\langle f - f_{n-1}, g_n \rangle|}{\|g_n\|}$, the function maximized by PPR, MP, OMP[7] and ILQP (in some particular cases);
- $S_2 = |\langle f - f_{n-1}, g_n \rangle|$, the function maximized by ILQP (in the general case);
- $S_3 = \frac{|Cov(f - f_{n-1}, g_n)|}{\|g_n\|}$;
- $S_4 = |Cov\,(f - f_{n-1}, g_n)|$, the function maximized by CC;

or their respective squared functions. Output units must have linear activation functions, the coefficients must be optimized after the selection of every term and $0 < \|g_n\| < B$ for every $\|g_n\|$. In contrast to ILQP, convexity is not required. Note that, in practice, these functions can be computed with computational cost $O(L)$, where L is the number of points in the data set. In order to maximize these functions there is no need, in principle, to back-propagate any error, although the non-linear optimization problem is still very difficult to solve.

Experiments in [Kwok and Yeung 1997b], with the same artificial problems as in [Hwang et al. 1994], show that although theoretical results apply to many different objective functions, their performance can be very different in practice. In general, S_1 and S_3 show the best performance. For noiseless data, S_3 slightly ouperforms S_1, but shows a higher sensitivity to the particular training set used. S_1 is better than S_4, confirming other comparative results that stated that error minimization is usually superior to correlation maximization (see [Prechelt 1997; Kwok and Yeung 1996b; Dunkin et al. 1997; Lahnajärvi et al. 2002]). The normalization performed in S_1 and S_3 (with respect to S_2 and S_4) allows to obtain better solutions. It is worth noting that the comparison is performed with the lowest mean test error among 15 networks ranging from 1 to 15 hidden units in 100 trial, but the authors do not give the number of hidden units where the results are achieved.

2.6.6 Orthogonal Least Squares Learning

In this section, several sequential methods that do not select the new frequency with the aim of matching the residue are described. The previously selected frequencies

[7] It can also be considered that MP and OMP maximize S_2, since the vectors in the dictionary have norm 1.

are kept fixed, and an (implicit or explicit) orthogonalization is performed in order to select the new frequency. *SAOCIF*, the method that we propose in chapter 3, is also based on this idea.

2.6.6.1 The Orthogonal Least Squares Learning Algorithm

In [Chen et al. 1991a] the **Orthogonal Least Squares Learning** (OLSL) algorithm is proposed, a learning procedure for RBFNs with the same underlying ideas that orthogonal least squares methods for the selection of the significant monomials in non-linear control systems modelling [Billings et al. 1988; Chen et al. 1989]. Every RBFN unit is considered as a regressor in a linear regression model, and the selection of the centers can be regarded as a problem of subset model selection. The procedure starts with a single Gaussian RBF hidden unit and it sequentially increases the number of hidden units, one at a time, until the model error is sufficiently small. The frequency of the new hidden unit (the center) is selected among the points in the data set. The classical Gram-Schmidt orthogonalization method is used at each step to form an orthogonal basis for the space spanned by the output vectors of the previously selected hidden units. In this context, an output vector is an element of $\mathbb{R}^L$, where L is the number of patterns, obtained by applying the Gaussian RBF to every point in the data set. For every point in the data set, the orthogonal component of its output vector to that space is obtained. After computing its optimal coefficient (with the sum-of-squares loss function), the point in the data set maximizing the error reduction ratio is selected. The procedure is terminated when a predetermined percentage of the total error is reduced.

Several extensions, optimizations and variations of the original OLSL can be found in the literature, although there is no theoretical result of convergence in any of them:

1. Extension to multi-output RBFNs [Chen et al. 1991b, 1992], computing the optimal coefficients for every output.

2. Reduction of the computational cost in some special cases [Chng et al. 1994, 1995], premultiplying the linear regression model by an orthonormal matrix.

3. Reduction of the computational cost in the general case [Chen and Wigger 1995], directly updating scalar inner products instead of updating column vectors (it is an internal optimization of the implementation).

4. Introduction of a regularization term [Chen 1995; Orr 1995; Chen et al. 1996].

5. Use of Genetic Algorithms to adjust the two key learning parameters, the regularization parameter and the hidden unit width [Chen et al. 1995, 1999].

Most of the optimizations are possible due to the fact that the set of frequencies is finite and known *a priori*. Although the algorithm does not always obtains the smallest possible network [Shertinsky and Picard 1996], its results are usually quite good (see, for example, [Lin et al. 2001]). In particular, it compares favorably with standard RBFN models.

2.6.6.2 ZM98

With similar underlying ideas to [Chen et al. 1991a], a sequential orthogonal scheme to the building and training of single hidden layer neural networks is described in [Zhang and Morris 1998] (we will refer to this method as **ZM98**). The main difference with respect to the OLSL algorithm lies in the selection of the frequencies.

The Gram-Schmidt orthogonalization is used at each step to form a set of orthogonal vectors $G_1(\omega_1), \cdots, G_n(\omega_n)$ for the space spanned by the output vectors of the hidden units. Hidden layer weights ω_n are found through gradient descent over $\|R_{n-1} - \lambda_n G_n(\omega_n)\|^2$ with random initialization, where R_{n-1} is the network error with the previously added hidden units and $G_n(\omega_n)$ is orthogonal to $G_1(\omega_1), \cdots, G_{n-1}(\omega_{n-1})$. As in [Chen et al. 1991a], an output vector is an element of $\mathbb{R}^L$, where L is the number of patterns, obtained by applying the activation function to every point in the data set. Output units have linear activation functions. Output layer weight λ_n is the optimal coefficient if R_{n-1} was approximated by $G_n(\omega_n)$ with minimum squared error (i.e., $\lambda_n = \frac{\langle R_{n-1}, G_n(\omega_n)\rangle}{\|G_n(\omega_n)\|^2}$, see Lemma 2 in chapter 3), and it is computed at every iteration of the gradient descent procedure for ω_n. When the training procedure is terminated, output layer weights must be recalculated in order to accommodate the effects of the non-orthogonal part of the output vectors of the hidden units, using the Gram-Schmidt orthogonalization results obtained at each step. Small $\|G_n(\omega_n)\|$ may lead to numerical problems. Therefore, the cost function is modified to penalize small $\|G_n(\omega_n)\|$. This is done by including the addition of a regularization term $\frac{\gamma}{\|G_n(\omega_n)\|^2}$ in the cost function to be minimized. An extended version of the algorithm with mixed types of hidden units (linear, simoidal and Gaussian) is also described.

As in OLSL, there is no theoretical result of convergence. In [Zhang and Morris 1998] the method is tested with very promising results when compared with standard MLPs. A comparison with OLSL is also described, showing similar performance with less hidden units. It is not compared with other sequential schemes.

2.6.6.3 Kernel Matching Pursuit

Recently, [Vincent and Bengio 2002] described the **Kernel Matching Pursuit** (KMP) algorithm, an extension of MP that can be used to build kernel-based solutions to SML problems. The emphasis of the KMP approach is put on the building

of an alternative to SVMs that controls the sparsity of the solution (i.e., the number of support vectors). As in SVMs, the frequencies of the resulting network are a subset of the points in the data set. Whereas good generalization abilities of SVMs are related to margin maximization, KMP is designed to build sparse kernel-based solutions minimizing the sum-of-squares error function. The idea behind KMP is simply the application of the MP family of algorithms to problems in SML using a kernel-based dictionary. The model compexity is directly controlled by the sparsity of the solution.

Three versions of KMP are defined:

1. Basic KMP, similar to classical MP.

2. KMP with *back-fitting* at every step, similar to OMP.

3. KMP with *pre-fitting*, similar to OLSL.

All three versions share the dictionary definition (needed in MP): Given a data set D, the dictionary $\mathcal{D}$ is defined as the set of functions $\mathcal{D} = \{K(x, x_i) : x_i \in D\}$, where K may be, for example, a symmetric positive definite kernel function. Whereas basic KMP and KMP with *back-fitting* work as the original versions, in KMP with *pre-fitting* the following function is optimized at the nth step:

$$\min_{g \in \mathcal{D}, \alpha_1, \cdots, \alpha_n \in \mathbb{R}^n} \left\| y - \left(\sum_{k=1}^{n-1} \alpha_k g_k + \alpha_n g \right) \right\|^2,$$

where y is the target vector and $g_1, \cdots, g_{n-1}$ are the previously selected vectors. Similar to OLSL and ZM98, a vector is an element of $\mathbb{R}^L$, where L is the number of patterns. This optimization problem can be solved exactly because a finite dictionary is used: A pass through the data set allows to find the required vector g, since the optimal coefficients (for the sum-of-squares error function) can be computed for every frequency. In addition, several optimization strategies can be used, as in OLSL.

In the experiments described in [Vincent and Bengio 2002], Gaussian kernels are tested. However, the model is not restricted to Gaussian RBF units. In fact, it is suggested that additional flexibility can be obtained by using other activation functions, whenever a finite dictionary is used:

1. There is no restriction on the shape of the kernel (no positive-definiteness constraint, symmetry, etc).

2. The dictionary could include more than a single fixed kernel shape.

3. The dictionary can even incorporate non-kernel based functions.

4. For huge data sets, a reduced subset can be used as the dictionary to speed up the training.

Although it is possible to extend the same ideas to any error function different from the sum-of-squares, the solution of the optimization problem at every step may lead to an important computational effort, since the optimal coefficients may not be exactly computable.

Besides the similarities with SVMs by using kernels associated with the training examples, some links with other methods are pointed out. As it can be easily seen, squared-error KMP with *pre-fitting* and Gaussian kernels is identical to OLSL. KMP in its basic form but generalized to non-quadratic error functions is also quite similar to boosting algorithms [Schapire 1990]. In this sense, Leveraged Vector Machines [Singer 2000] are very closely related to KMP with a different loss function. A very similar method to KMP, but particular for Gaussian processes, can be found in [Smola and Bartlett 2001]. The main difference also lies in the loss function to be optimized. Due to the hypothesis of the Gaussian processes, specific optimizations can be performed in order to reduce the computational cost of the procedure. As in OLSL and ZM98, there is no theoretical result of convergence.

In an artificial separable binary classification problem, the basic version of KMP was unable to separate the data points with the same number of frequencies as the number of support vectors in the SVM model. In the same conditions, the *back-fitting* and *pre-fitting* versions were able to find good solutions, although they chose different points as frequencies. The solution obtained by the *pre-fitting* version of KMP was very similar to that of SVMs. The same behavior was observed for the US postal service database with Gaussian activation functions. In addition, only a slight loss of performance is observed when using half of the number of support vectors of the SVM model, similar to [Smola and Schölkopf 2000; Smola and Bartlett 2001]. Classical RBFN models obtained worse results than SVMs and KMP with *pre-fitting*. When a validation set was used to select the final number of frequencies (for KMP) or the parameter C (for SVMs), the experimental comparisons between KMP with *pre-fitting* and SVMs for several benchmark classification problems showed comparable results with typically much sparser models for KMP with *pre-fitting*.

2.6.7 Sequential Polynomial Approximations

Sequential polynomial approximations are also related to FNNs, although they do not share most of the features of the previously described ones. The use of polynomials allows to avoid non-linear optimization problems, because of the absence of non-linear frequencies. In contrast, the number of monomials (i.e., the number of terms) grows very fast with the input dimension and the degree.

The **Group Method of Data Handling** (GMDH) [Ivakhnenko 1971], for example, constructs polynomial approximations that can be implemented with FNNs. The construction of the network is made layer by layer as follows. For every two input variables, a second-order polynomial is constructed. The coefficients are computed by solving the normal equations in a minimum mean-square error sense with respect to the true target values on a training set. After the coefficients are computed, the performance index is determined by computing the sum-of-squares error on a test set. Only those elements (hidden units) whose performance index exceeds a certain threshold are allowed to be used in the second layer. These elements are used as the basis to construct, in the same way, the (more complex) elements of the third layer, an so on. The maximum number of layers is fixed *a priori*. For a review of GMDH see, for example, [Ivakhnenko and Ivakhnenko 1995]. Other models and applications inspired by GMDH can be found, for example, in [Tenorio and Lee 1990; Parker and Tummala 1992; Nikolaev and Iba 2003].

The works where the OLSL algorithm is inspired can be seen, in fact, as sequential polynomial approximations [Billings et al. 1988; Chen et al. 1989]. The selection of the monomials is performed trying to maximize the increment to the explained variance of the desired output, based on orthogonal least squares methods. In [Liu et al. 1998], after a fixed *a priori* number of monomials is selected, new monomials are added to the network as new observations are received, similar to the RAN approach. Lyapunov techniques are used to modify the coefficients.

In [Rivals and Personnaz 2003a], an orthogonal least squares procedure is used, as in OLSL, to iteratively select the monomials such that, together with the previously selected ones, decrease the residual error at most. The procedure is stopped based on Fisher tests or leave-one-out errors. In addition, Feature Selection is performed by selecting only the input variables present in those monomials. These variables are used in subsequent learning steps with MLPs.

2.6.8 Summary and Discussion

In this section we will use neural network terminology.

The construction of a sequential approximation can be formulated as a **state space search problem** [Kwok and Yeung 1997a].

The **state space** corresponds to the collection of functions that can be obtained by the model. In this sense, several points have been adressed in the literature:

1. The connectivity of the resulting network, which shows a large variety: from the classical FNN with one-hidden layer of units (with or without input-to-output connections) to cascade architectures with any number of units per layer.

2. The type of hidden units used. In addition to MLP or RBF units, Pi-Sigma networks or Product units are (exceptionally) used. The activation functions of the hidden units are more heterogeneous, depending not only on the method but also on the field of application: smoothers, sigmoidal functions, Gaussian functions, Hermite polynomials, fixed dictionaries, wavelets or kernel functions, among others.

3. The activation function of the output layer, which usually is sigmoidal or linear.

The **initial state** usually is a network with no hidden units. When input-to-output connections are used, they are trained (or computed) in the initial step of the procedure.

The **evaluation criterion** is, most of the times, an estimation of the network performance, although it may also be the training error.

The **termination of the search** happens either when the performance of the model is satisfactory or it begins to deteriorate. As an alternative, the search can be terminated when the addition of several hidden units does not reduce substantially the training error.

The **search strategy** determines how the model evolves during the construction. This is the most interesting part of the algorithm, and includes:

1. The number of hidden units added at every step and their connectivity with the other units. This point is strongly related to the state space defined. The most usual strategy consists in adding one hidden unit at every step, with the same kind of connectivity between two consecutive steps.

2. The activation function of the new hidden units, which is usually predefined *a priori*.

3. The training steps, which can be performed with any non-linear optimization algorithm.

4. How the new hidden units are obtained, together with the modification of the weights (frequencies and coefficients) of the whole network:

   (a) Obtaining the new frequencies without keeping the previously selected ones fixed:

   - Adding a new hidden unit when the error does not decrease significantly during the training process, following the Dynamic Node Creation scheme [Ash 1989]. The rest of the weights are not initialized, so that the obtained weights prior to the addition of the new hidden unit are the initial guess to train the next network. The

whole network is trained after the addition of every new hidden unit [Hirose et al. 1991; Bello 1992; Azimi-Sadjadi et al. 1993; Bastian 1994; Bartlett 1994; Nabhan and Zomaya 1994; Wang et al. 1994; Zhang 1994; Chung and Lee 1995; Leerink et al. 1995; Setiono and Hui 1995; Shin and Ghosh 1995; Vinod and Ghose 1996; Hernández and Fernández 2002].

- Splitting hidden units that have high variance during the training process, as in the Meiosis Networks procedure [Hanson 1990; Sanger 1991a,b; Wynne-Jones 1992; Khorasani and Weng 1994; Weng and Khorasani 1996].
- Combining a standard training process with a specialization process when the network does not perform well on a presented pattern, following the ideas of Resource-Allocating Network [Platt 1991], as in [Lee and Kil 1991; Fritzke 1994a,b; Kadirkamanathan and Niranjan 1993; Kadirkamanathan 1994; Liu et al. 1996; Jankowski and Kadirkamanathan 1997; Yingwei et al. 1997a,b; Rosipal et al. 1998; Liu et al. 1999; Salmerón et al. 2001; Lee and Street 2003; Alexandridis et al. 2003]. All of them are specific for RBFNs.

(b) Keeping the previously selected frequencies fixed, and training only the new frequency and the coefficients. The disavantage of weight freezing is that each optimization step is not optimal, and this can result in larger networks than those in which all the weights are optimized [Kwok and Yeung 1993]. In contrast, the difficulty and the computational cost of solving the underlying optimization problem is reduced. The coefficients are usually optimized (if needed) after the selection of the frequencies (*back-projection*):

i. The new frequencies are selected to optimize a certain objective function, which usually depends on the previous residue [Kwok and Yeung 1997b]. Most of the existing methods choose the new frequencies either to maximize the correlation between the output of the new unit and the residual error or to match the previous residue as best as possible (equivalently, the selected frequency maximizes the Fourier transform of the residue). Relevant works with these underlying ideas are:

   - Cascade-Correlation [Fahlman and Lebiere 1990] and its many variations, (usually) constructing cascade architectures with the new frequencies (usually) maximizing the correlation between the new unit and the residual error [Smotroff et al. 1991; Yeung 1991; Littmann and Ritter 1992a,b; Sjøgaard 1992; Courrieu 1993; Yeung 1993; Fang and Lacher 1994; Phatak and Koren 1994; Leerink

et al. 1995; Vyšniauskas et al. 1995; Kwok and Yeung 1996a; Mohraz and Protzel 1996; Prechelt 1997; Treadgold and Gedeon 1997a,b; Liang and Dai 1998; Treadgold and Gedeon 1998; Lahnajärvi et al. 1999; Treadgold and Gedeon 1999a; Lehtokangas 1999, 2000; Ma and Khorasani 2000; Islam and Murase 2001; Ma and Khorasani 2003, 2004].

- Projection Pursuit Regression [Friedman and Stuetzle 1981; Diaconis and Shahshahani 1984; Huber 1985; Jones 1987, 1992], originally described in the Statistics framework, with a two-layer architecture and matching the residue at every step, with or without convex approximations [Flick et al. 1990; Zhao and Atkeson 1991; Intrator 1993; Saha et al. 1993; Moody 1994; Verkooijen and Daniels 1994; Roosen and Hastie 1994; Jutten and Chentouf 1995; Zhao and Atkeson 1996].
- Matching Pursuit [Mallat and Zhang 1993] and Orthogonal Matching Pursuit [Pati et al. 1993], in the context of Signal Processing, similar to Projection Pursuit Regression with a previously fixed dictionary.
- Projection Pursuit Learning Network, a specific Projection Pursuit Regression algorithm for MLPs [Maechler et al. 1990; Hwang et al. 1991, 1992a,b; Lay et al. 1994; Hwang et al. 1994; Kwok and Yeung 1995a,b, 1996b].
- The Incremental Linear Quasi-Parallel algorithm, similar to Projection Pursuit Regression but constructing convex approximations with classical FNNs [Kůrková and Beliczyński 1995a,b; Beliczyński 1996a,b; Dingankar and Sandberg 1996; Draelos and Hush 1996; Beliczyński and Kůrková 1997; Dunkin et al. 1997; Hlaváčková and Sanguinetti 1998; Hlaváčková and Fischer 2000; Zhang 2002].

ii. Important exceptions to the idea of matching the residue, where an (implicit or explicit) orthogonalization is performed in order to select the new frequency, are:

- The Orthogonal Least Squares Learning algorithm [Chen et al. 1991a] and its extensions, optimizations and variations, specific for RBF units and the quadratic loss function. The frequencies are selected from the points in the data set [Chen et al. 1991b, 1992; Chng et al. 1994; Chen 1995; Chen and Wigger 1995; Chng et al. 1995; Orr 1995; Chen et al. 1996, 1999].
- ZM98 [Zhang and Morris 1998]. The selection of the frequencies are found through gradient descent with random initialization.

A regularization term is added to the quadratic loss function to avoid numerical problems derived from small norms.

- Kernel Matching Pursuit with *pre-fitting* [Vincent and Bengio 2002], similar to the Orthogonal Least Squares Learning algorithm, but with the aim of constructing kernel-based solutions for the quadratic loss function and not restricted to RBF units. Similar schemes to Kernel Matching Pursuit can be found in [Singer 2000; Smola and Bartlett 2001]

iii. Sequential polynomial approximations [Ivakhnenko 1971; Ivakhnenko and Ivakhnenko 1995; Billings et al. 1988; Chen et al. 1989; Tenorio and Lee 1990; Parker and Tummala 1992; Liu et al. 1998; Nikolaev and Iba 2003; Rivals and Personnaz 2003a]. In this case, the use of polynomials allows to avoid non-linear optimization problems, because of the absence of non-linear frequencies. In contrast, the number of monomials (i.e., the number of terms) grows very fast with the input dimension and the degree.

In practice, sequential approximations have been found to be very competitive in different tasks, such as detection and classification of buried dielectric anomalies [Azimi-Sadjadi et al. 1993], speech recognition [Mallat and Zhang 1993], electroencephalogram automatic epileptic seizure detection [Weng and Khorasani 1996], voltage and current fault detection [Lin et al. 2001], image compression [Ma and Khorasani 2002], handwritten digit recognition [Vincent and Bengio 2002] or detection and classification of breast cancer nuclei [Lee and Street 2003]. Experiments on well-known benchmark problems for classification and regression have also shown the good properties of sequential methods.

Most of the aforementioned methods use families of functions that have the universal approximation property. In the same way, the convergence property in L^2 is satisfied by many of them. When the convergence property is satisfied, the approximation error is upper bounded by $B/\sqrt{N}$, where N is the number of terms of the approximation. The constant B depends on the target function f and the particular hypotheses of the theoretical developments. This rate for sequential approximations has been several times rediscovered with different constants and hypotheses [Jones 1992; Barron 1993; Darken et al. 1993; Koiran 1994; Kůrková and Beliczyński 1995b; DeVore and Temlyakov 1996; Lee et al. 1996; Dingankar and Sandberg 1996; Kwok and Yeung 1997b; Hlaváčková and Sanguinetti 1998; Zhang 2002]. Many of them are based on the results in [Jones 1992]. In principle, this rate of approximation suggests that the curse of dimensionality can be avoided, since the dimension is not explicitly involved. However, it seems to be implicitly present in at least two ways [Kainen 1998]. First, the class of functions for which the approximation rate holds is smaller with increasing dimension. Second, the constant B also depends

implicitly on the dimension, because depends on f. One of the common features of these results is that they have always been proved (to our knowledge) either when the new term is selected so as to approximate the previous residue or for sequential convex approximations following the scheme

$$\|f - f_N\| \leqslant \inf_{g,\alpha_N} \|f - ((1-\alpha_N)f_{N-1} + \alpha_N g)\| + \varepsilon_N,$$

where the approximation of the residue is also present.

As it can be seen, most of the existing sequential methods that keep the previously selected frequencies fixed choose the new frequency looking at its approximation capability of the residue. Although this strategy leads to approximations convergent towards the target function, it may be far from being the best strategy, as it will be seen in chapter 3, even if the coefficients are optimized after the selection of the frequencies. Regardless of the coefficients optimization, trying to approximate the residue does not take into account the interactions with the previously selected terms.

2.7 Feature Selection with MLPs

2.7.1 Feature Selection

There exist many situations where one does not have *a priori* neither a model that could describe the phenomenon nor the knowledge of which variables are adequate to describe it. This is a very common situation in Medicine or Psychology, for example. The expert may have several intuitions about the variables related to a certain problem, but by no means has neither the security that those are all the variables needed to explain the phenomenon nor the confidence that all of them are useful. When important variables are missing, the problem cannot be solved. If some variables are irrelevant, models that consider them as important will probably have performance problems. In addition, the number of available examples is usually small and they may be noisy or incomplete.

Therefore, features[8] in an SML system are not equally informative. Some of them may be noisy, redundant or meaningless for the task at hand. The problem of Feature Selection (FS) can be defined as follows [Liu and Motoda 1998]: given a set of I candidate features, select a subset that performs the best under a certain evaluation criterion. In addition to reducing the storage requirements and increasing the computational speed, FS may lead to improve the performance of an SML system, as it has been known for a long time [Kittler 1978, 1986; Siedlecki and

[8]Sometimes, the term "variable" is used for the raw input, and "feature" for any input constructed from the original variables. We will not distinguish between variables and features.

Sklansky 1988]. Nowadays, FS is still an active line of research [Guyon and Elisseeff 2003].

We will focus on FS from the SML point of view. That is, our major concern is to improve the predictive behavior of the system. From an SML point of view, FS procedures control the Bias/Variance decomposition by means of the input dimension, establishing a clear connection with the curse of dimensionality [Liu and Motoda 1998]. When too many variables are considered the system may be too complex, increasing the variance term (whenever the complexity of the system is not controlled by other ways or there is not enough data to filter irrelevant variables, for example). As far as the variables are eliminated, the complexity is reduced, but the bias term may increase. Likewise, when too many variables are considered, there may be many different solutions capable of fitting the same data set. But only a few number of these solutions will lead to good generalization. If the system gives some importance to irrelevant variables in order to fit the data set, it will use this information for new data, probably leading to poor generalization even if we try to control the overfitting. This problem is shared by all SML approaches, and it is one of the most important motivations for FS from an SML point of view.

As pointed out in [Blum and Langley 1997; Liu and Motoda 1998; Molina et al. 2002], there is no commonly accepted definition of the relevance of a variable. Given a data set, we will consider that a variable is irrelevant for an SML system when its optimal performance is not affected negatively by the absence of that variable, following [Liu and Motoda 1998] (page 29). Note that this is a dynamic definition, since the relevance of a variable may be affected by the presence or absence of other ones. For example, redundant features may be considered as irrelevant. In addition, observe that this definition of relevance is dependent of the particular SML system used.

From a computational point of view, the previous definition of FS leads to a **search problem** in a space of 2^I elements. Therefore, two components must be specified: the **feature subset evaluation criterion** and the **search procedure** through the space of feature subsets.

2.7.1.1 Feature Subset Evaluation Criteria

Many different evaluation criteria to select the subset of features can be found in the literature, based on different measures, such as distance, information, consistency, dependence or accuracy, among others (see [Liu and Motoda 1998; Molina et al. 2002] for details). These measures are extremely related to the motivation of the FS procedure. For pure SML objectives, the accuracy of the induced model is surely the most appealing measure. From other points of view (data reduction, noise removal, for example), different measures can be adequate [Liu and Motoda 1998]. These two FS perspectives are in turn related to the *wrapper* and the *filter* approaches

respectively, as explained next.

In the wrapper approach [John et al. 1994], the feature subset selection is done using an SML algorithm as a black box (i.e., no knowledge of the algorithm is needed, just the interface). The evaluation criterion of a feature subset is the accuracy of the induced SML system with those features. The filter approach, in contrast, selects the features as in a preprocessing step, without specifically taking into account the effects of the selected (or discarded) features in the future performance of a particular SML system.

In the comparison performed in [Kohavi and John 1997], a significant improvement in accuracy is achieved by the wrapper approach with respect to the filter one. The independence of the SML system is precisely the main disadvantage of filter models for SML objectives. Wrappers, in contrast, are very sensitive to the SML algorithm, so that sometimes it is difficult to determine the "real" relevance of a feature for the problem. Regarding the computational cost, wrapper approaches are computationally more expensive than filter ones.

A different scheme is used in the *embedded* approach, where the SML algorithm has its own FS procedure (either implicit or explicit). Decision trees and several FS methods for MLPs, for example, can be seen from this point of view.

2.7.1.2 Search Procedures

Concerning the search procedure, there exists a wide range of methods to avoid the computationally prohibitive (in the general case) exhaustive search. Some of them are able to determine the optimal feature subset under certain assumptions, such as the Branch and Bound algorithm [Narendra and Fukunaga 1977], which needs the evaluation criterion to be monotone. Most methods seek for a suboptimal solution heuristically (the accuracy of an SML system is not necessarily monotone).

Rather well-known heuristic methods are the sequential ones, where features are deleted from (or added to) the partial solution at every step. The simplest ones are the Sequential Backward Selection (SBS) and the Sequential Forward Selection (SFS) procedures (see [Kittler 1978], for example). SBS is a top-down process. Starting from the complete set of available features, one feature is deleted at every step of the algorithm. The selected feature is that whose removal gives rise to the best value of the evaluation criterion. SFS is a bottom-up process. The procedure begins with an empty subset of selected features. At every step, the feature which gives rise, together with the already selected features, to the best value of the evaluation criterion is added to the subset. Ideally, it is expected that performance improves as far as features are deleted (added), but at some point the elimination (inclusion) of further features results in performance degradation [Kittler 1986]. Several features may be deleted (added) at the same step. On the basis of these two algorithms, bidirectional methods apply SFS and SBS alternatively or simultaneously, as in a

Plus-m-Minus-n procedure. As an example, the Sequential Backward Floating Selection method defined in [Pudil et al. 1994] works at every iteration in two steps. In the first step (unconditional exclusion), the least significant feature is deleted. In the second step (conditional inclusion), the most significant of the remaining features is added only if the resulting subset is the best subset with the same size found so far. This second step is repeated until no improvement is found. The procedure stops when the number of features required is obtained.

Non-sequential heuristic methods are usually characterized by the property that the next state is determined in a nondeterministic manner. The simplest idea is to generate random subsets of features [Liu and Setiono 1996], but there also exist more sophisticated strategies, based on simulated annealing or genetic algorithms, as in [Siedlecki and Sklansky 1988; Yang and Honavar 1998]. Other FS methods weight the features in a continuous way, so that the search is performed in the (continuous) weight space rather than in the (discrete) feature space [Kira and Rendell 1992].

The search stopping criterion is dependent on the particular feature subset evaluation and search procedure used. For wrapper approaches, the most commonly stopping criterion used depends on the accuracy of the induced SML system with the feature subset (the training error or an estimation of the generalization error, for example).

2.7.2 Specific Feature Selection Algorithms with MLPs

Many FS algorithms for MLPs share underlying ideas with pruning algorithms [Reed 1993]. In fact, some pruning procedures can be used to perform FS. In this sense, many FS procedures for MLPs use as evaluation criterion some variation of the concept of *saliency* used in some pruning methods. A feature is considered more important whenever its saliency is larger, so that input units with small saliencies can be eliminated. There exist other FS algorithms for MLPs that also define a saliency but they are not based on pruning. Regarding the search procedure, most FS schemes for MLPs use the SBS algorithm.

The next subsections are devoted to a more detailed description of specific FS methods with MLPs, classified according to their saliencies. Our aim has been to carry out a review as complete as possible. A discussion of several important aspects of the described schemes is also included, which motivates our work presented in chapter 4.

2.7.2.1 FS Using Unit Saliencies Based on Weight Saliencies

The saliency of a weight is defined with the aim to be a measure of its relative importance in the network. The weights with lower saliency are the candidates to be pruned (weight saliency was originally defined for pruning algorithms). The

saliency of a unit based on weight saliencies is defined as the summation of the saliencies of the weights of that unit.

Some definitions of weight saliencies can be found in the literature with different criteria:

1. Depending only on the magnitudes of the weights, as in [Lee et al. 1993; Wikel and Dow 1993; Tetko et al. 1996; Messer and Kittler 1998]. These models consider that small weights are less important than large ones.

2. As a function of the variance of the weights during the training process, as in [Lee et al. 1993; El-Deredy and Branston 1995; Cibas et al. 1994a, 1996]. The idea behind these models is that weights with small variance are not very active, and can be eliminated.

3. Approximating the difference in the loss function (usually the sum-of-squares) when the weight is set to zero, as in [Cibas et al. 1994b, 1996; Stahlberger and Riedmiller 1997; Attik et al. 2004], related to the wrapper approach. These saliencies are based on the Optimal Brain Damage model [Le Cun et al. 1990] and its later improvements [Hassibi and Stork 1993; Pedersen et al. 1996], where an approximation to the difference in the loss function is given in terms of the Hessian matrix.

2.7.2.2 FS Using Unit Saliencies Not Based on Weight Saliencies

The saliency of a unit may also be defined independently on the weights magnitudes:

1. As the estimated relative strength of the connections of the input unit, computed using the weights and the activation values of a trained network on a data set [Tetko et al. 1994]. Input units with low relative strengths are the candidates to be removed.

2. As the difference between the respective output vectors when the features are removed from a trained network (equivalently, set to 0) [De et al. 1997; Bahbah and Girgis 1999]. The underlying idea is that the difference will not differ too much for less important attributes.

3. As the minimum variation of the weights when the input unit is temporarily removed [Castellano and Fanelli 2000]. The selected input unit is such that, when deleted, the weights in the first layer can be analytically modified as least as possible so that the net-input of the hidden units remains approximately unchanged.

4. As the contribution of the input unit to the variance of the net-input of the hidden units [Boger and Guterman 1997; Boger 2003]. The underlying hypothesis is that input units that make relatively small contributions to the variance of the net-input of the hidden units can be considered as constants and replaced by a fixed bias.

5. Computing (or approximating) the sensitivity (in terms of the derivative) of the outputs with respect to the input units, either in the whole space or a data set [Ruck et al. 1990a; Priddy et al. 1993; Sano et al. 1993; Engelbrecht and Cloete 1996; Zurada et al. 1997; Hsu et al. 2002]. These methods are based on the hypothesis that irrelevant features produce smaller variations in the output values than relevant ones.

6. Back-propagating an estimation of the mutual information between the outputs and the targets [Sindhwani et al. 2004]. This model tries to take profit of information measures.

7. As the value in the loss function when the feature values are substituted by its average value (in an already trained network) [Moody and Utans 1992; Lee et al. 1993; Baesens et al. 2000], following the wrapper approach. This heuristic assumes that the replacement of a variable by its average value removes its influence on the network output.

8. Simply as the value (or an approximation of it) in the loss function when the feature is removed from the network (after the network has already been trained) [Mozer and Smolensky 1989; Mao et al. 1994; Setiono and Liu 1997; Egmont-Petersen et al. 1998; Van de Laar et al. 1999; Verikas and Bacauskiene 2002], following the wrapper approach. As an exception, in [Onnia et al. 2001] every candidate feature is temporarily added before training the network. Some of them approximate this value in several ways, instead of directly computing it from the data set [Mozer and Smolensky 1989; Mao et al. 1994; Egmont-Petersen et al. 1998].

2.7.2.3 Other Feature Selection Schemes for MLPs

In [Belue and Bauer 1995; Steppe and Bauer 1996], the definition of saliency in [Ruck et al. 1990a] is used to compare the saliency of an artificially introduced noisy variable with that of the original features. The average saliency of the noisy variable is assumed to be normally distributed, and features whose saliency falls inside a certain confidence interval are removed. In [Bauer et al. 2000], another saliency measure is defined where the magnitudes of the weights of every input unit are compared to those of an injected noise feature.

Similar to a pruning procedure, hidden and input units are removed in an integrated way in [Steppe et al. 1996]. Starting with a full network, hidden and input units are sequentially removed from the network whenever the performance of the reduced network improve. The procedure stops when the elimination of any feature does not lead to a better performance. A similar idea can be found in [Rivals and Personnaz 2003b], where the elimination of hidden and input units are based on statistical tests.

In [Grandvalet 2000], the input values are perturbed with the injection of Gaussian noise. The variance of the injected noise is allowed to be different for every feature, and it is modified during the training procedure. This variance gives a measure of the relevance of the variable.

An ensemble of neural networks is the basis for the FS procedure described in [Van de Laar and Heskes 2000]. A set of subsets of features is maintained at every step (initially the set of all possible subsets with one variable less than the original features). The basis of the scheme is the SBS procedure but, similar to bidirectional FS methods, subsets of features may also be added. For every subset of features, the generalization performance is estimated for each neural network in the ensemble. The procedure stops when all variables are removed.

2.7.2.4 Critical Points of Feature Selection with MLPs

Many variants of FS procedures with MLPs have been proposed. The following list describes the most interesting points, in our opinion, for FS with MLPs, namely the saliency (evaluation criterion), the search procedure, the stopping criterion of the network training, the data set where the value of the loss function is measured and the network retraining previous to computing the saliency.

1. Regarding the saliency, trying to minimize the value of the loss function is surely the optimal criterion for FS from an SML point of view [Liu and Motoda 1998]. As explained in the previous sections, it is the most commonly used. Saliencies different from this one are mainly motivated and justified by computational cost reasons. Although there are situations where they may work, there is a lack of theoretical results that support them.

2. Regarding the search procedure:

   (a) Some authors only define a saliency measure, and do not define a real FS procedure [Sano et al. 1993; Tetko et al. 1994; Engelbrecht and Cloete 1996; De et al. 1997; Grandvalet 2000].

   (b) Sometimes, the features are ranked according to their saliency, and the higher ranked features are selected in a single step as in a preprocessing

procedure [Ruck et al. 1990a; Moody and Utans 1992; Priddy et al. 1993; Wikel and Dow 1993].

(c) In general, FS with MLPs follows the scheme of the SBS procedure. It starts by training a network with the whole set of features. Then, the saliencies are computed and the input variable with the lowest saliency is removed. The weights of the network are modified, and the loop starts again until a certain criterion is satisfied. Some particular variations of the procedure are:

i. Sometimes, a regularization term is added to the original loss function in order to encourage the network to contain removable units (see, for example, [Cibas et al. 1994a; Setiono and Liu 1997; Verikas and Bacauskiene 2002]).

ii. After a feature has been definitively removed, the most usual way to modify the weights of the network is by retraining the whole network (either from scratch or from the current state), although several models recalculate analytically the weights (see, for example [Stahlberger and Riedmiller 1997; Egmont-Petersen et al. 1998; Van de Laar et al. 1999; Castellano and Fanelli 2000; Attik et al. 2004]).

iii. Sometimes, several features are eliminated in the same step, as in [Cibas et al. 1994a,b; Belue and Bauer 1995; Steppe and Bauer 1996; Zurada et al. 1997; Boger 2003], maybe losing the possibility of finding the interactions among the eliminated variables.

Only exceptionally the SBS procedure is not used. In [Onnia et al. 2001], for example, a classical SFS procedure is performed. In [Hsu et al. 2002], a heuristic subset selection is used where at every step a certain percentage of the best features are selected according to the defined saliency and the results in previous steps. In [Sindhwani et al. 2004], a direct search is performed among feature subsets of the desired size. Other models that do not use the SBS procedure [Steppe et al. 1996; Van de Laar and Heskes 2000] have also been briefly described (see previous section). In general, we also think that the SBS procedure is a proper search procedure, since it may help to detect irrelevant variables in the first steps.

3. Regarding the stopping criterion of the network training, the networks are trained, in most cases, until a local minimum of the loss function with the training set, where the saliencies definitions are supposed to work well[9]. There

[9]Many authors do not specify this point in their papers. We suppose that the training process under expressions like "Train a network for a number of epochs on the training data" or "After the network was trained,..." tries to find a local minimum of the loss function with the training set.

are several exceptions [Wikel and Dow 1993; Tetko et al. 1996; Steppe et al. 1996; Van de Laar et al. 1999; Van de Laar and Heskes 2000; Baesens et al. 2000; Onnia et al. 2001; Hsu et al. 2002; Verikas and Bacauskiene 2002], where an early stopping procedure is performed (usually with a validation set).

4. Regarding the data set where the saliency is measured (when a data set is needed to compute the saliency), most of existing methods only use the training set. Several exceptions are [Moody and Utans 1992; Cibas et al. 1994b; Steppe et al. 1996; Van de Laar et al. 1999; Van de Laar and Heskes 2000; Onnia et al. 2001; Hsu et al. 2002; Verikas and Bacauskiene 2002; Sindhwani et al. 2004], where a validation or test set is used to compute the saliency (usually to estimate the generalization error).

5. Regarding the network retraining previous to computing the saliency, most of the existing procedures compute the saliency of every feature by looking at the behavior of a network trained with that feature. For example, the input values are temporarily modified (removed or substituted by its average value) in a trained network. The only models that, in order to compute the saliency, retrain the network at every step with every feature temporarily removed/added are (to our knowledge) those described in [Steppe et al. 1996; Onnia et al. 2001], and none of them is a pure SBS procedure.

In general, there is no commonly accepted criterion about some of the previously described issues when performing FS with MLPs:

1. When to stop the training phase.

2. Which is the best data set (when needed) to measure the saliency.

3. Whether it is better to compute the saliency of a variable when the network has been trained with that feature, as opposed to the alternative of computing the saliency in a network trained without it.

There exist a lack of comparative results among these issues in the literature. In [Leray and Gallinari 1999] several methods are compared with artificial problems. In [Fernández and Hernández 1999] a more exhaustive comparison is performed with benchmark data sets. But these studies are focused on the performances of the original methods rather than testing the aforementioned points. Chapter 4 is devoted to a comparison of these criteria when performing FS with MLPs.

2.8 Margin Maximization, Support Vector Machines and AdaBoost

2.8.1 Margin Maximization

As previously said (see (2.15) and (2.16) in section 2.5), some theoretical results in Statistical Learning Theory provide probabilistic bounds on the distance between the empirical and the expected risk. These bounds depend on a measure of the flexibility of the class of functions under study, such as the VC-dimension, and they are related to the Bias/Variance trade-off. In practice, unfortunately, this measure is often neither easily computable nor very helpful [Müller et al. 2001]. But for the class of hyperplanes, the VC-dimension can be bounded by another quantity, the inverse of the *margin*, which is related to the minimal distance of a vector to a hyperplane [Vapnik 1995]. This combination of results allows to consider the margin maximization as a good heuristic to try to minimize the VC-dimension, which in turn bounds the expected risk.

2.8.2 Support Vector Machines

Suppose the classification task given by a data set D as in (2.1), where each y_i belongs to $\{-1,+1\}^C$ and C is the number of classes[10].

SVMs can be described as follows [Boser et al. 1992; Cortes and Vapnik 1995; Vapnik 1995; Cristianini and Shawe-Taylor 2000]: the input vectors are mapped into a (usually high-dimensional) inner product space through some non-linear mapping ϕ, chosen *a priori*. In this space (the *feature space*), an optimal separating hyperplane is constructed. By using a kernel function $K(u,v)$ the non-linear mapping can be implicit, since the inner product needed to define the hyperplane can be evaluated as $\langle \phi(u), \phi(v) \rangle = K(u,v)$. In SVMs, an optimal separating hyperplane means a hyperplane with maximal normalized margin with respect to the data set. The (functional) margin of a point (x,y) with respect to a function f^o is defined as $mrg(x,y,f^o) = yf^o(x)$. The margin of a function f^o with respect to a data set D is the minimum of the margins of the points in the data set. If f^o is a hyperplane, the normalized (or geometric) margin is defined as the margin divided by the norm of the orthogonal vector to the hyperplane. Thus, the absolute value of the normalized margin of a point is the distance of that point to the hyperplane, and the hyperplane with maximal normalized margin with respect to the data set is the hyperplane whose minimum distance to the points of every class is maximum.

When the data set D is linearly separable, the maximal (hard) margin hyperplane

[10] For 2-class problems, usually $y_i \in \{-1,+1\}$.

is obtained by solving

$$\begin{array}{ll} \text{Minimize}_{\mathrm{w},b} & \langle \mathrm{w}, \mathrm{w} \rangle \\ \text{subject to} & y_i(\langle \mathrm{w}, x_i \rangle + b) \geqslant 1 \qquad i = 1, \cdots, L \end{array} \tag{2.26}$$

When the data set is not linearly separable (neither in the input space nor in the feature space), some tolerance to noise is introduced in the model. In this case, slack variables ξ_i are introduced to allow the margin constraints to be violated [Cortes and Vapnik 1995]. The optimization problem related to the 1-norm soft margin is

$$\begin{array}{lll} \text{Minimize}_{\mathrm{w},b,\xi} & \langle \mathrm{w}, \mathrm{w} \rangle + C \sum_{i=1}^{L} \xi_i & \\ \text{subject to} & y_i(\langle \mathrm{w}, x_i \rangle + b) \geqslant 1 - \xi_i & i = 1, \cdots, L \\ & \xi_i \geqslant 0 & i = 1, \cdots, L \end{array} \tag{2.27}$$

for a certain constant C. Using Lagrangian and Kuhn-Tucker theory, the resulting SVM for a binary classification problem has the form

$$f^o_{SVM}(x) = \sum_{i=1}^{L} y_i \alpha_i K(x_i, x) + b \tag{2.28}$$

where the inner product has been replaced by its general kernel version and the vector $(\alpha_i)_{i=1}^L$ is the (1-norm soft margin) solution of the following constrained optimization problem in the dual space:

$$\begin{array}{ll} \text{Maximize}_{\alpha} & M(D) = -\frac{1}{2} \sum_{i,j=1}^{L} y_i \alpha_i y_j \alpha_j K(x_i, x_j) + \sum_{i=1}^{L} \alpha_i \\ \text{subject to} & \sum_{i=1}^{L} y_i \alpha_i = 0 \qquad \text{(bias constraint)} \\ & 0 \leqslant \alpha_i \leqslant C \qquad i = 1, \cdots, L \end{array} \tag{2.29}$$

In many implementations, b is treated apart (fixed *a priori*, for example) in order to avoid the bias constraint. A point is well classified if and only if its margin with respect to f^o_{SVM} is positive. The points x_i with $\alpha_i > 0$ (active constraints) are named *support vectors*. *Bounded support vectors* have $\alpha_i = C$. Regarding their margin value, non-bounded support vectors have margin 1, while bounded support vectors have margin less than 1. By setting $C = \infty$, one obtains the hard margin hyperplane. The cost function $-M(D)$ is (plus a constant) the squared norm of the error function $y(x) - f^o_{SVM}(x)$ in the Reproducing Kernel Hilbert Space associated with $K(u, v)$ [Cristianini and Shawe-Taylor 2000]. The most usual kernel functions $K(u, v)$ are polynomial, Gaussian-like or some particular sigmoids. In contrast to FNNs, note that the appearance of the SVM solution in (2.28) (a linear combination of the inner products with the support vectors as frequencies) is a consequence of the optimization problem solved.

The Bias/Variance trade-off is controlled by the parameter C. It expresses the importance given to the violation of the margin constraints, as can be seen in (2.27).

In the solution (2.28) of (2.29), C bounds the coefficient of every term. When $C = 0$, any constant function is a solution of (2.27). Constant functions have low variance but high bias. When $C = \infty$, the hard margin hyperplane (if exists) is the solution with lowest bias (it allows to classify correctly every point in the data set), but probably with high variance. In some sense, the parameter C acts as a regularization parameter [Smola et al. 1998]. Related to the Structural Risk Minimization principle, increasing values of C define a nested sequence of hypothesis classes with increasing complexity. The hard margin SVM tries to minimize the VC-dimension (by maximizing the margin) while keeping null empirical risk. The soft margin SVM minimizes a combination of the complexity measure and some function related to the empirical risk. Other soft margin SVMs can be obtained by changing the second term of this combination or the constraints (see [Cortes and Vapnik 1995; Suykens and Vandewalle 1999a], for example).

Recent years have seen a quick growing of research in SVMs, motivated by both interesting theoretical and experimental results (see, for example [Cortes and Vapnik 1995; Schölkopf et al. 1997; Vapnik 1998a; Pontil and Verri 1998; Chapelle et al. 1999; Drucker et al. 1999; Schölkopf et al. 1999; Smola et al. 2000; Cristianini and Shawe-Taylor 2000; Brown et al. 2000; Ramaswamy et al. 2001; Hua and Sun 2001; Lodhi et al. 2002; Herbrich 2002; Schölkopf and Smola 2002; Van Gestel et al. 2004], other references in `http://kernel-machines.org` and some links in the web page of Isabelle Guyon `http://www.clopinet.com/SVM.applications.html`). In some cases, however, it has been shown that the maximal margin classifier is not the optimal one [Raudys 1998b, 2000].

2.8.3 AdaBoost

The purpose of boosting [Freund 1990; Schapire 1990] is to find a highly accurate classification rule by combining many *weak classifiers* (or weak hypotheses), each of which may be only moderately accurate. The AdaBoost algorithm is a particular boosting algorithm introduced in [Freund and Schapire 1996, 1997] and later improved in [Schapire and Singer 1999]. In AdaBoost, the weak hypotheses are learned sequentially, one at a time. Conceptually, at each iteration the weak hypothesis is biased to classify the examples which were most difficult to classify by the preceding weak hypotheses. The margin of every example is considered in order to construct every weak hypothesis, which are linearly combined into a single output rule named the *combined hypothesis*.

The generalized AdaBoost algorithm for binary classification [Schapire and Singer 1999] maintains a vector of weights as a distribution D_t over the examples. At round t, the goal of the weak learner algorithm is to find a weak hypothesis $h_t : \mathcal{X} \to \mathbb{R}$ with moderately low error with respect to the distribution D_t. In this setting, weak hypotheses $h_t(x)$ make real-valued confidence-rated predictions. Initially,

the distribution D_1 is uniform, but after each iteration, the algorithm increases (or decreases) the weights $D_t(i)$ for which $h_t(x_i)$ makes a bad (or good) prediction, with a variation proportional to the confidence $|h_t(x_i)|$. The final hypothesis, $f^o_{AB} : \mathcal{X} \to \mathbb{R}$, computes its predictions using a weighted vote of the weak hypotheses $f^o_{AB}(x) = \sum_{j=1}^{t} \alpha_j h_j(x)$. For each example x, the sign of $f^o_{AB}(x)$ is interpreted as the predicted class (-1 or $+1$), and the magnitude $|f^o_{AB}(x)|$ is interpreted as a measure of confidence in the prediction. The concrete weight updating rule can be expressed as

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)\cdot\exp(-\alpha_t y_i h_t(x_i))}{Z_t} = \dots \\ &= \frac{\exp(-\sum_{j=1}^{t} \alpha_j y_i h_j(x_i))}{\prod_{j=1}^{t} Z_j} = \frac{\exp(-y_i f^o_{AB}(x_i))}{\prod_{j=1}^{t} Z_j} = \frac{\exp(-mrg(x_i, y_i, f^o_{AB}))}{\prod_{j=1}^{t} Z_j} \end{aligned} \tag{2.30}$$

In [Schapire and Singer 1999] it is proved that the training error of the AdaBoost algorithm decreases exponentially with the normalization factor Z_t computed at round t. This property is used to guide the design of the weak learner, which attempts to find a weak hypothesis h_t minimizing $Z_t = \sum_{i=1}^{L} D_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))$.

There is not a direct interpretation of AdaBoost within the Bias/Variance decomposition. It seems clear that bias is reduced when the number of rounds grows up (in fact, that was the original goal of boosting). The behavior of variance is not so clear. On the one hand, variance tends to increase as the number of terms increases (specially if the examples with the smallest margins are noisy). On the other, the aggregated nature of AdaBoost controls in some way the variance term. Several experiments on this issue have been performed in [Schapire et al. 1998; Bauer and Kohavi 1999], but they cannot be considered as conclusive, since they use definitions of bias and variance for the decomposition of the 0/1 loss function which may not be very adequate (see [Domingos 2000a]).

The learning bias of AdaBoost is proven to be very aggressive on maximizing the margin of the training examples [Schapire et al. 1998; Schapire and Singer 1999], since it concentrates on the examples with the smallest margins. This makes a clear connection to the SVMs learning paradigm. From (2.30) and the previous expression of Z_t, it can be said that AdaBoost is a stagewise procedure for minimizing a certain error function which depends on the functional margin $-mrg(x_i, y_i, f)$. More specifically, AdaBoost tries to asymptotically minimize $g(b) = \sum_i \exp\left(-\frac{1}{2} y_i \sum_t b_t h_t(x_i)\right)$ [Rätsch et al. 1999], a function that depends on the negative exponential of the margin of the combined classifier. More details about the relationship between AdaBoost and SVMs can be found in [Rätsch et al. 2001, 2002].

2.8.4 Mixed Models Between FNNs and SVMs

The method that we propose in chapter 5 includes a modification of the quadratic loss function for classification problems related to SVMs. This section is devoted to

a brief description of mixed models between FNNs and SVMs, related to our work. We did not find many publications in this line.

For the linearly separable case, a learning algorithm that asymptotically obtains the maximum margin classifier is described in [Raudys 1998a]. The architecture used is a Single-Layer Perceptron (an MLP without hidden units and with sigmoidal units in the output layer), trained with BP. In order to work, the weights must be large, so that only the points closest to the decision hyperplane contribute significantly to the error (and to the final determination of the hyperplane location). In practice, learning rates are exponentially increased. No modification of the sum-of-squares error function is done. A similar idea is described in [Cid-Sueiro and Sancho-Gómez 2001].

The Least Squares Support Vector Machine described in [Suykens and Vandewalle 1999a] is a soft margin SVM where the 1-norm of the slack vector in (2.27) is replaced by the 2-norm. In addition, inequality constraints are substituted by equality constraints. As a consequence, the whole minimization problem can be seen as that of minimizing the quadratic loss function with a regularization term (the inverse of the geometric margin). However, the solution is obtained in an *SVM manner*, so that the resulting model uses kernel functions and shares with SVMs the existence of support vectors.

In [Suykens and Vandewalle 1999b], a training procedure for MLPs based on SVMs is described. The activation function is not necessarily a kernel function, and the learning process is guided by the minimization of the estimation of an upper bound of the VC-dimension.

The work in [Vincent and Bengio 2000] investigates learning architectures in which the kernel function can be replaced by more general similarity measures which can have internal parameters. The cost function is modified to be dependent on the margin. In particular, the cost function $M(D) = \sum_{i=1}^{L}[0.65 - \tanh(mrg(x_i, y_i, f))]^2$ is used in the performed experiments. The frequencies are forced to be a subset of the points in the data set, as in SVMs.

Chapter 3

SAOCIF: A Sequential Algorithm with Optimal Coefficients and Interacting Frequencies

In this chapter we describe an algorithm for sequential approximation with Feedforward Neural Networks (FNNs) with optimal coefficients and interacting frequencies. The contribution of the new frequency is measured in terms of its capability of approximation to the target vector together with the previously selected frequencies. There is no explicit intention to match the residue. The idea behind the proposed algorithm can be extended to approximation in Hilbert spaces, maintaining orthogonal-like properties. The algorithm is tested with several artificial and benchmark data sets.

3.1 Introduction

Most of the sequential models found in the literature keep the previously selected frequencies fixed and train only the new frequency and the coefficients. Among these sequential models, many of them choose the new frequency so that it matches the previous residue as best as possible. Equivalently, the selected frequency tries to maximize the Fourier transform of the residue at every step. After the selection of the frequencies, the coefficients are usually optimized (*back-projection*). Important exceptions to the idea of approximating the residue are the OLSL algorithm [Chen et al. 1991a], ZM98 [Zhang and Morris 1998] and the KMP with *pre-fitting* algorithm [Vincent and Bengio 2002], where an (implicit or explicit) orthogonalization procedure is performed (see section 2.6).

Although the strategy of approximating the residue leads to approximations convergent towards the target function, it may be far from being the best strategy,

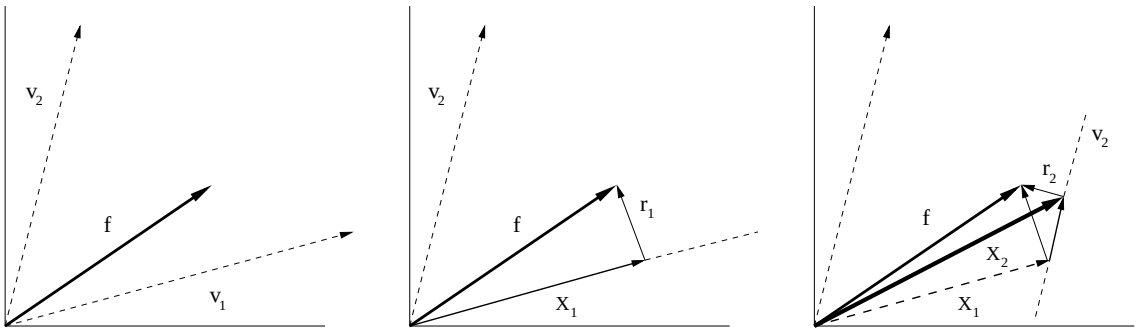

Figure 3.1: Sequence of the approximation of a vector f in $\mathbb{R}^2$ with v_1 and v_2 by matching the previous residue without recalculating the coefficients. In the first step (middle), X_1 is obtained. In the second step (right), r_1 is approximated with v_2. The resulting vector (X_2) is not the best approximation that can be achieved with v_1 and v_2. In this case, optimizing the coefficients (*back-projection*) allows to obtain f.

as it can be observed in the example in figure 3.1. When approximating the vector f with v_1 and v_2 we obtain X_2. Clearly, this is not the best possible approximation, since v_1 and v_2 form a basis of $\mathbb{R}^2$. In this case, optimizing the coefficients of the previously added terms (*back-projection*) would lead to a much better approximation (exact, in fact) of the target function. Other examples in this line can be found in [Diaconis and Shahshahani 1984; DeVore and Temlyakov 1996]. But recalculating the coefficients is not enough, as illustrated in the example in figure 3.2. Suppose that X_1 is a partial approximation of f, and h is the vector which best matches the residue r_1. Since h does not lie on the plane that contains X_1 and f, it is not necessarily the vector that, together with X_1, best approximates the target vector f. Any vector lying on the plane that contains X_1 and f (g, for example) allows an exact approximation of f. Regardless of the coefficients optimization, trying to approximate the residue does not take into account the interactions with the previously selected terms. Any vector lying on the plane that contains f and a vector of the subspace spanned by the previous terms allows an exact approximation of the target vector. The vector that best matches the residue does not necessarily satisfy this property.

In this chapter we describe an algorithm for *Sequential Approximation with Optimal Coefficients and Interacting Frequencies* (*SAOCIF*) for FNNs, which combines these two key ideas. On the one hand, it optimizes the coefficients (the linear weights), so that the best approximation with the selected vectors is always achieved, as in figure 3.1. On the other, the frequencies (the non-linear weights) are selected at every step taking into account the interactions with the previously selected terms, as in figure 3.2. The interactions are discovered by means of the optimal coefficients. The contribution of the new frequency is measured in terms

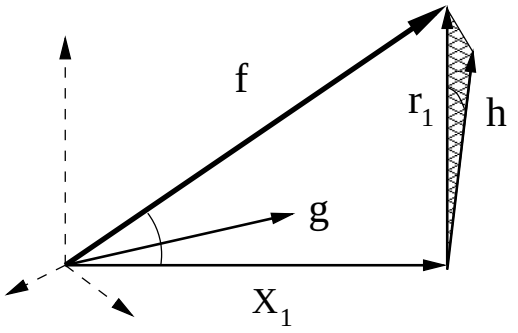

Figure 3.2: Approximation of a vector f in $\mathbb{R}^3$ matching the previous residue and recalculating the coefficients. The vector h is the vector that best matches the residue r_1. The vector g, which lies on the plane that contains f and X_1, allows an exact approximation to f when combined with X_1. The vector h (not on this plane) does not have this property. In this case g is preferred to h and optimizing the coefficients is not enough if h is selected.

of its capability of approximation to the target vector together with the previously selected frequencies. There is no explicit intention to match the residue. In the example in figure 3.2, *SAOCIF* would select g (instead of h), because it allows a better approximation of f when combined (interacts) with X_1. That is the idea of interacting frequencies. Whereas the approximation of the residue and the posterior optimization of the coefficients can be seen as two consecutive steps, *SAOCIF* performs both steps simultaneously. The resulting scheme combines the locality of sequential approximations, where only one frequency is found at every step, with the totality of non-sequential methods, such as Back-Propagation (BP), where every frequency interacts with the others.

In the proposed algorithm, a number of candidate frequencies are obtained at every step using different heuristics. Every candidate frequency is installed in the network, and the whole set of coefficients are optimized, in order to test the real contribution of the frequency to the approximation to the target vector. The candidate frequency that, together (interacting) with the previously selected frequencies, allows a better approximation of the target vector is finally selected. Algorithms that try to approximate the residue only optimize the coefficient of the new frequency, while keeping the rest of coefficients fixed. The (*back-projection*) procedure optimizes the coefficients after the frequency has been selected, but it is not involved in the process of selection of the frequency.

The proposed algorithm for FNNs can be seen as an extension and generalization of the OLSL, ZM98 and KMP with *pre-fitting* algorithms in several ways. First, it is not restricted to select the candidate frequencies from the points in the data set. In this sense, a number of different heuristics can be used. Second, it is possible to further tune the selected frequency.

The idea of *interacting frequencies* for sequential approximations may have a positive influence not only for approximation, but also for generalization abilities of FNNs. As it can be seen in figure 3.2, the importance of the interacting frequencies lies in the hypothesis that they allow to find better partial approximations, with the same number of hidden units, than frequencies selected just to match the residue as best as possible. Likewise, the same level of approximation may be achieved with less hidden units. In terms of the Bias/Variance decomposition, it will be possible to obtain simpler models with the same bias. Therefore, an improvement in the generalization performance is expected.

The idea behind *SAOCIF* can be extended to approximation in Hilbert spaces, maintaining orthogonal-like properties. The theoretical results obtained prove that, under reasonable conditions, the residue of the approximation is (in the limit) the best one that can be obtained with any subset of the given set of vectors. In this case, the importance of the interacting frequencies lies in the expectation of increasing the rate of approximation. In the particular case of L^2, *SAOCIF* can be applied to approximations by polynomials, Fourier series, wavelets and neural networks, among others.

Experimental results using a wide range of benchmark data sets show a very satisfactory performance when compared to other approaches. In particular, it works better than methods that select the new frequencies based on the idea of matching the residue, confirming the suitability of the idea of interacting frequencies. The selection of the frequencies among the points in the data set, as in OLSL and KMP, also seems to be a promising heuristic for both Multi-Layer Perceptrons (MLPs) and Radial Basis Function Networks (RBFNs). In this case, the resulting model shares with Support Vector Machines the property that their frequencies are a subset of the data set.

Additionally, the use of sinusoidal MLPs (i.e., MLPs where the activation function of the hidden units was either the sine or the cosine) showed a very good behavior. In [Lee and Kil 1991], several simulation results indicate that sinusoidal activation functions provide better approximation capability when compared to sigmoidal and Gaussian ones. Theoretical results in [Suzuki 1998] show that sinusoidal MLPs have better properties, regarding the number of terms in the approximation, than sigmoidal ones. In practice, the use of sinusoidal activation functions for MLPs led to very promising results [Sopena et al. 1999a].

The definition and the proposed algorithm for *SAOCIF* can be found in section 3.2. In section 3.3, *SAOCIF* is briefly compared with other sequential schemes. The extension to approximation in Hilbert spaces is described in section 3.4. The experimental work is carried out in sections 3.5, 3.6 and 3.7. Finally, section 3.8 is devoted to the proof of the theoretical results.

3.2 Definition of *SAOCIF* and Algorithm

3.2.1 Definition

Definition. Let H be the Hilbert space $\mathbb{R}^L$, where L is the number of patterns in a data set $D = \{x_1, \cdots, x_L\}$, $f = (f_1, \cdots, f_L) \in H$ the target vector and Ω a space of frequencies. A *Sequential Approximation with Optimal Coefficients and Interacting Frequencies* (*SAOCIF*) for FNNs is a sequence of vectors $\{X_N\}_{N \geqslant 0}$ in H, whose terms are defined as:

1. $X_0 = 0$.
2. $X_N = \sum_{k=1}^{N-1} \lambda_k^N v_{\omega_k} + \lambda_N^N v_{\omega_N}$, so that
   (a) The coefficients $\lambda_1^N, \cdots, \lambda_{N-1}^N, \lambda_N^N$ are optimal. That is, the vector X_N is the best approximation of f with vectors $v_{\omega_1}, \cdots, v_{\omega_{N-1}}, v_{\omega_N}$.
   (b) The frequency ω_N is selected taking into account the interactions of v_{ω_N} with $v_{\omega_1}, \cdots, v_{\omega_{N-1}}$ in order to minimize $\|f - X_N\|$.

Remarks.

1. In FNNs terminology, every frequency $\omega_k \in \Omega$ is associated with a hidden unit $\varphi_k(\omega_k, x)$. The i-th component of v_{ω_k} is the value of the hidden unit $\varphi_k(\omega_k, x)$ at the i-th point in D. That is, $v_{\omega_k} = (\varphi_k(\omega_k, x_1), \cdots, \varphi_k(\omega_k, x_L))$. The output function of the network with N hidden units is $f_{FNN}^o(x) = \sum_{k=1}^{N} \lambda_k^N \varphi_k(\omega_k, x)$.
2. At step N, a new frequency is considered (ω_N), the number of terms of the approximation is increased by one ($\lambda_N^N v_{\omega_N}$), and the coefficients $\lambda_1^N, \cdots, \lambda_{N-1}^N$ are optimized in order to obtain the best approximation of f with vectors $v_{\omega_1}, \cdots, v_{\omega_{N-1}}, v_{\omega_N}$. The frequencies $\omega_1, \cdots, \omega_{N-1}$ are kept fixed.
3. As it is well known [Achieser 1956], since X_N is the best approximation of f with $v_{\omega_1}, \cdots, v_{\omega_{N-1}}, v_{\omega_N}$, it holds that
$$\forall k : 1 \leqslant k \leqslant N \quad \langle f - X_N, v_{\omega_k} \rangle = 0, \tag{3.1}$$
where $\langle , \rangle$ is the inner product in H. That is, the residue $f - X_N$ is orthogonal to the space generated by $v_{\omega_1}, \cdots, v_{\omega_{N-1}}, v_{\omega_N}$. Equivalently, X_N is the orthogonal projection of f onto the space spanned by $v_{\omega_1}, \cdots, v_{\omega_{N-1}}, v_{\omega_N}$. By definition of inner product, (3.1) is equivalent to the following linear equations system:
$$\begin{pmatrix} \langle v_{\omega_1}, v_{\omega_1} \rangle & \langle v_{\omega_2}, v_{\omega_1} \rangle & \cdots & \langle v_{\omega_N}, v_{\omega_1} \rangle \\ \langle v_{\omega_1}, v_{\omega_2} \rangle & \langle v_{\omega_2}, v_{\omega_2} \rangle & \cdots & \langle v_{\omega_N}, v_{\omega_2} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle v_{\omega_1}, v_{\omega_N} \rangle & \langle v_{\omega_2}, v_{\omega_N} \rangle & \cdots & \langle v_{\omega_N}, v_{\omega_N} \rangle \end{pmatrix} \cdot \begin{pmatrix} \lambda_1^N \\ \lambda_2^N \\ \vdots \\ \lambda_N^N \end{pmatrix} = \begin{pmatrix} \langle f, v_{\omega_1} \rangle \\ \langle f, v_{\omega_2} \rangle \\ \vdots \\ \langle f, v_{\omega_N} \rangle \end{pmatrix}. \tag{3.2}$$

Solving (3.2) is equivalent to solving the Least Squares problem associated with the data set. Consequently, once the frequencies $\omega_1, \cdots, \omega_{N-1}, \omega_N \in \Omega$ have been selected, the optimal coefficients $\lambda_1^N, \cdots, \lambda_{N-1}^N, \lambda_N^N$ can be calculated by solving (3.2). From a geometrical point of view, it is equivalent to find the approximation directions. It can be easily proved that the system has only one solution if and only if $v_{\omega_1}, \cdots, v_{\omega_{N-1}}, v_{\omega_N}$ are linearly independent. Otherwise, the system has more than one solution. Since the frequencies $\omega_1, \cdots, \omega_{N-1}$ are kept fixed, the proposed system at step N is equal to the system at step $N-1$, but with a new row and a new column. The system solution is a continuous function of the matrix and the independent vector elements at any point where the matrix is nonsingular [Ortega 1972].

4. The vectors $v_{\omega_1}, \cdots, v_{\omega_{N-1}}, v_{\omega_N}$ are not necessarily mutually orthogonal. The approximation with orthogonal vectors has been widely studied (see, for example, [Achieser 1956]). The coefficients of the best approximation of $f \in H$ by means of an orthogonal system $g_1, \cdots, g_N$ only depend on the projections of f onto the vectors of the system:

$$Y_N = \sum_{k=1}^{N} \lambda_k \, g_k = \sum_{k=1}^{N} \frac{\langle f, g_k \rangle}{\|g_k\|^2} \, g_k.$$

In this case, we have

$$\begin{aligned} \|f - Y_N\|^2 &= \|f\|^2 - \sum_{k=1}^{N} |\lambda_k|^2 \|g_k\|^2 \\ &= \|f\|^2 - \left\| \sum_{k=1}^{N} \lambda_k \, g_k \right\|^2 = \|f\|^2 - \|Y_N\|^2. \end{aligned}$$

In particular, $\forall N \geqslant 1 \quad \|f - Y_{N+1}\|^2 \leqslant \|f - Y_N\|^2$. As it can be easily seen, the information provided by every term is independent on the others. This allows, for instance, the construction of the approximation in a sequential manner, where the terms are added one at a time until the approximation is satisfactory. If the orthogonal system is infinite, we may wonder about the behavior of the series

$$g = \sum_{k=1}^{\infty} \frac{\langle f, g_k \rangle}{\|g_k\|^2} \, g_k.$$

If the system spans H, then the series is always convergent, and $\|f - g\| = 0$. The main problem of approximating with a fixed system (even if it is orthogonal) resides on the lack of flexibility. Linear expansions in a single

basis are not flexible enough. The information can be diluted across the whole basis [Mallat and Zhang 1993]. In L^2, for example, the approximation error with a fixed basis cannot be made smaller than $O(1/(d\sqrt[d]{n}))$, where d is the dimension of the input to the function [Barron 1993]. This happens even with an orthogonal basis. For this reason, to achieve a good approximation, a very large number of vectors may be needed, even if we order them by $|\langle f, g_k\rangle|$. *SAOCIF* keeps the idea of adding terms one at a time, but the residue can be reduced in a flexible and (in some sense) optimal manner. So we can expect to reduce the necessary number of terms to achieve the same degree of approximation.

3.2.2 Basic properties

As a first result, the approximations that satisfy (3.1) are characterized in Lemma 1. As an immediate consequence, every element X_N of a *SAOCIF* satisfies these properties. As it can be observed, there is a great parallelism between these properties and those satisfied by an approximation with orthogonal vectors. The only differences are in (L1a) and (L1b), and both are a generalization. Lemma 2 gives some insight on the best approximation of the residue with only one vector. Although the results shown in this section are probably known[1], they are included here to make the work self-contained. These results are given for the general case of a complex inner product, but the same results hold for a real inner product, simply by replacing $\mathbb{C}$ with $\mathbb{R}$. The proofs can be found in section 3.8.

Lemma 1. Let H be a Hilbert space, $f \in H$ and $X_N = \sum_{k=1}^{N} \lambda_k^N v_{\omega_k}$, such that its vectors and coefficients satisfy (3.1). Then,

(L1a) For every $1 \leqslant j \leqslant N$

$$\lambda_j^N = \frac{\left\langle f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k}, v_{\omega_j} \right\rangle}{\left\| v_{\omega_j} \right\|^2}.$$

(L1b) For every $1 \leqslant j \leqslant N$

$$\|f - X_N\|^2 = \left\| f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} \right\|^2 - \left|\lambda_j^N\right|^2 \left\| v_{\omega_j} \right\|^2.$$

[1] We did not find all of them in the reviewed literature. However, we think that they cannot be considered as relevant contributions.

(L1c) $\|f - X_N\|^2 = \|f\|^2 - \|X_N\|^2$ (energy conservation).

(L1d) $\|X_N\|^2 = \sum_{k=1}^{N} \lambda_k^N \overline{\langle f, v_{\omega_k} \rangle}$.

(L1e) $\langle f - X_N, f \rangle = \|f - X_N\|^2$.

Lemma 2. Let H be a Hilbert space, $f \in H$. Then, the function $P_N : v(\Omega) \to \mathbb{R}$ defined as

$$P_N(v_\omega) = \inf_{\mu \in \mathbb{C}} \|f - (X_{N-1} + \mu v_\omega)\|^2$$

is always well defined $\left(\mu_\omega = \frac{\langle f - X_{N-1}, v_\omega \rangle}{\|v_\omega\|^2}\right)$, it can be computed as

$$P_N(v_\omega) = \|f - X_{N-1}\|^2 - \frac{|\langle f - X_{N-1}, v_\omega \rangle|^2}{\|v_\omega\|^2}, \tag{3.3}$$

and it is continuous on $v(\Omega)$.

3.2.3 Practical properties

SAOCIF satisfies a number of interesting properties to implement it in an efficient and reasonably simple fashion.

Since the frequencies $\omega_1, \cdots, \omega_{N-1}$ are kept fixed, the proposed system at step N is equal to the system at step $N-1$, but with a new row and a new column. Therefore, (3.2) can be solved efficiently with bordered systems techniques [Faddeeva 1959].

In addition, the residue $\|f - X_N\|^2$ can be computed avoiding one pass through the data set. By (L1c) we have $\|f - X_N\|^2 = \|f\|^2 - \|X_N\|^2$, with $\|f\|^2$ constant. Therefore, the frequency that minimizes the error is such that maximizes $\|X_N\|^2$. By (L1d), we know that

$$\|X_N\|^2 = \sum_{k=1}^{N} \lambda_k^N \overline{\langle f, v_{\omega_k} \rangle}. \tag{3.4}$$

The values of $\{\langle f, v_{\omega_k} \rangle\}_{1 \leqslant k \leqslant N}$ are the independent vector of the linear equations system (3.2) just solved to obtain $\{\lambda_k^N\}_{1 \leqslant k \leqslant N}$, which can be kept stored in memory. Hence, once the coefficients have been obtained, we can compute $\|X_N\|^2$ with cost $O(N)$. Note that the cost of computing $\|X_N\|^2$ or $\|f - X_N\|^2$ directly from the data set would be $O(L \cdot N)$.

Finally, it can be easily verified that the goodness of a new frequency ω_0 with regard to its approximation capability does not depend on the norm of the vector

v_{ω_0}. Suppose we are at the step N, and we have just selected ω_N and calculated $\lambda_1^N, \cdots, \lambda_{N-1}^N, \lambda_N^N$. Suppose that we modify the norm of the vector v_{ω_N} by defining $v'_{\omega_N} = h \cdot v_{\omega_N} \quad h \in \mathbb{R}, h \neq 0$ ($\|v'_{\omega_N}\| = |h| \cdot \|v_{\omega_N}\|$). Proposing again the linear equations system (3.2), we must find $\lambda_1'^N, \cdots, \lambda_{N-1}'^N, \lambda_N'^N$ such that

$$\begin{pmatrix} \langle v_{\omega_1}, v_{\omega_1} \rangle & \langle v_{\omega_2}, v_{\omega_1} \rangle & \cdots & \langle v'_{\omega_N}, v_{\omega_1} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle v_{\omega_1}, v_{\omega_{N-1}} \rangle & \langle v_{\omega_2}, v_{\omega_{N-1}} \rangle & \cdots & \langle v'_{\omega_N}, v_{\omega_{N-1}} \rangle \\ \langle v_{\omega_1}, v'_{\omega_N} \rangle & \langle v_{\omega_2}, v'_{\omega_N} \rangle & \cdots & \langle v'_{\omega_N}, v'_{\omega_N} \rangle \end{pmatrix} \cdot \begin{pmatrix} \lambda_1'^N \\ \vdots \\ \lambda_{N-1}'^N \\ \lambda_N'^N \end{pmatrix} = \begin{pmatrix} \langle f, v_{\omega_1} \rangle \\ \vdots \\ \langle f, v_{\omega_{N-1}} \rangle \\ \langle f, v'_{\omega_N} \rangle \end{pmatrix}.$$

This system solution is $(\lambda_1'^N, \cdots, \lambda_{N-1}'^N, \lambda_N'^N) = (\lambda_1^N, \cdots, \lambda_{N-1}^N, \lambda_N^N/h)$. Computing the norm of the new $X'_N = \sum_{k=1}^{N-1} \lambda_k^N v_{\omega_k} + \lambda_N'^N v'_{\omega_N}$ using (3.4) we have

$$\begin{aligned} \|X'_N\|^2 &= \sum_{k=1}^{N-1} \lambda_k^N \overline{\langle f, v_{\omega_k} \rangle} + \lambda_N'^N \overline{\langle f, v'_{\omega_N} \rangle} \\ &= \sum_{k=1}^{N-1} \lambda_k^N \overline{\langle f, v_{\omega_k} \rangle} + \frac{\lambda_N^N}{h} \overline{\langle f, h \cdot v_{\omega_N} \rangle} = \|X_N\|^2. \end{aligned}$$

Therefore, $\|f - X_N\|^2 = \|f - X'_N\|^2$, as desired. Therefore, we can normalize the vectors without loss of generality.

3.2.4 Algorithm and Implementation Details

3.2.4.1 An Algorithm for *SAOCIF*

In figure 3.3 we describe a sequential training algorithm for FNNs following the ideas of *SAOCIF* definition. Hidden units are added one at a time, choosing the frequencies in a flexible manner, so as to adjust the network until we have a satisfactory model. The algorithm works as follows. Suppose that we are at step N and we have a procedure to generate frequencies. Every candidate frequency is installed in the network and the whole set of coefficients are optimized, in order to test (together with the previously selected $N-1$ frequencies) the real contribution of the frequency to the approximation to the target vector. There is no explicit intention to match the residue. That is the idea of interacting frequencies. When the frequency is satisfactory or there are no more candidate frequencies (criterion 1, see below), the selected frequency can be optionally tuned, as in MP (see section 2.6).

Concerning the architecture needed to construct the approximation, it must have the following characteristics:

1. It must be a feed-forward architecture with a hidden layer of units (including both MLPs with one hidden layer and RBFNs).

Algorithm
  repeat
    Increase by 1 the number of hidden units N
    Pick an activation function for the new hidden unit
    repeat
      Assign a candidate frequency ω to the new hidden unit
      Compute the optimal coefficients $\{\lambda_k\}_{1\leqslant k\leqslant N}$ by solving (3.2)
      Compute the *output norm* $\|X_N\|^2$ with (3.4)
      Set $\omega_N := \omega$ if $\|X_N\|^2$ is maximized
    until the frequency ω_N is satisfactory or there are no more
          candidate frequencies (criterion 1)
    Optionally, tune the selected frequency ω_N
    Fix the frequency ω_N in the network
  until the network is satisfactory (criterion 2)
end Algorithm

Figure 3.3: An algorithm to construct an FNN following the ideas of *SAOCIF*.

2. There are no restrictions about the dimension of the input and the output. There will be as many as the target function have. If there are several outputs, their optimal coefficients can be computed by solving their respective linear equations systems, and the total error can be computed as the summation of the individual errors for every output.

3. There is no restriction about the weights in the hidden units. The output units cannot have biases, but this is not a real restriction, since they can be considered as frequencies with a simple transformation. Another solution consists in adding a new hidden unit with constant activation function or, as we will see later, fixing them in advance.

4. There is no restriction about the activation functions in the hidden units. In particular, they can be sigmoidal, Gaussian, sines, cosines, polynomial, wavelets, etc. Obviously, different units may have different activation functions, as suggested in CC, ILQP, ZM98 or KMP (see section 2.6). The output units must have a linear activation function.

As we can see, the only restriction in the architecture is the linear activation function in the output units. Some important aspects of the algorithm are:

1. The linear system (3.2) can be solved efficiently with bordered systems techniques, as explained in section 3.2.3.

2. Since $\|f - X_N\|^2 = \|f\|^2 - \|X_N\|^2$, to find out the minimum error $\|f - X_N\|^2$ we only need to calculate the maximum output $\|X_N\|^2$, and this can be done efficiently using (3.4).

3. The frequency and the activation function of the new hidden unit could also be jointly selected.

4. As it will be shown, the theoretical conditions that guarantee the convergence of *SAOCIF* to the target function f involve a global minimization problem (see (3.6) in section 3.4). We decided to tackle it in a different way. In this sense, the new frequency is selected from a set of candidates generated following different strategies. This is probably the most important part of the algorithm, and some of the experiments are designed to test the different strategies, which are defined in the next subsection.

5. Regarding the criterion 1 in figure 3.3, every strategy to select the frequencies has its own one.

6. Regarding the criterion 2 in figure 3.3, any stopping criterion can be used: percentage of learned patterns or maximum squared error in the training set, early stopping with a validation set, low relative rate of decrease of the error, etc.

The resulting algorithm combines the locality of sequential approximations, where only one frequency is found at every step, with the totality of non-sequential methods, such as BP, where every frequency interacts with the others. The interactions are discovered by means of the optimal coefficients. The importance of the interacting frequencies lies in the hypothesis that they allow to find better partial approximations, with the same number of hidden units, than frequencies selected just to match the residue as best as possible. Likewise, the same level of approximation may be achieved with less hidden units. In terms of the Bias/Variance decomposition, it will be possible to obtain simpler models with the same bias. Therefore, an improvement in the generalization performance is expected. Results in section 3.6 experimentally seem to confirm this hypothesis.

3.2.4.2 Strategies to select the frequencies

In the experiments (see, for example, section 3.6) four strategies are introduced in order to test the algorithm:

1. **Random strategy**: the frequencies are selected at random within a certain range, up to an *a priori* fixed number of frequencies. This number of frequencies is relative to the number of inputs: when we say "N random frequencies"

we mean $N \cdot I$ frequencies, where I is the input dimension. This strategy is closely related to ZM98 (see section 3.3). In its simplest form, it can be considered as a baseline for the other strategies.

2. **BGA strategy**: this is a more sophisticated strategy derived from the field of Evolutionary Algorithms, where a population of frequencies evolves driven by a Breeder Genetic Algorithm (BGA) [Mühlenbein and Schlierkamp-Voosen 1993] with the squared error as the fitness function. The BGA works roughly as follows. First, an initial population of μ individuals is created. The best q individuals are directly inserted into the new generation (q-elitism), so that the new generation contains an individual which is at least as good as the best individual of the previous generation. Then, a percentage (determined by the truncation rate τ) of the best individuals of the population are selected to be recombined and mutated, as the basis to construct a new generation. The recombination operator is applied by randomly (and uniformly) selecting two parents. Then, the mutation operator is applied to every offspring with a certain probability, and the loop starts again. The BGA finishes when a maximum number of evaluations of the fitness function, determined *a priori*, is achieved.

   The parameters of the BGA must be set in advance. In particular, we need to establish:

   (a) The size of the population μ.

   (b) The q-elitism.

   (c) The truncation rate τ.

   (d) The maximum number of evaluations of the fitness function. This quantity is, as for the Random strategy, relative to the number of inputs.

   (e) The initial population and the range of values that every individual (frequency) can take.

   Other internal parameters for recombination and mutation in the BGA algorithm were fixed following the suggestions of the empirical study performed in [Belanche 1999].

3. **Input strategy**: the frequencies are selected from the training points in the data set (as it is often the case in RBFNs) in a deterministic manner: for every hidden unit to be added, every point in the training set is tested as a candidate frequency. The number of points in the data set determines the number of frequencies to test. This strategy does not need any parameter, and it is the equivalent of OLSL and KMP with *pre-fitting* (see section 3.3). Therefore, the

resulting model shares with Support Vector Machines the property that their frequencies are a subset of the data set.

4. **Grid strategy**: the frequencies are chosen from the points in a regular grid of a hypercube of the input space. This strategy is deterministic and it can only be used with a low computational cost when the dimension of the input space is small. The number of points in the grid determines the number of frequencies to test, and the only parameters needed for this strategy are the number of points in the hypercube and the length of the edge.

The computational cost (in terms of the number of candidate frequencies) of every strategy may be very different. Whereas the Input one has fixed computational cost (given the data set), the Random, Grid and BGA strategies can be parameterized so that their respective computational costs may be very different.

3.2.4.3 Unifying Parameters for Different Strategies: The Gain Factor

For MLP units, the Random, BGA and Grid strategies need a range where looking for frequencies. This value must be set in advance, and it will characterize the maximum and minimum value of any frequency (or every component, when the input dimension is greater than 1). For the Input strategy, the net-input of the hidden units must be multiplied by a certain factor, in order to adapt the net-input to the activation function. This factor must also be fixed *a priori*, and it is equivalent to the so called "gain" factor for sigmoidal functions or the inverse of the "temperature" factor in simulated annealing.

For RBF units, the range of frequencies should be the range of the input values, whereas the width of the activation function must be set in advance.

These *a priori* different treatments to select the frequencies for MLP and RBF units can be unified as follows:

1. The range of frequencies always is:
   (a) For MLP units, the interval $[-0.5, +0.5]$, except for the Input strategy.
   (b) For RBF units, the range of the input values (the Input strategy already satisfies this condition).

2. A preprocessing procedure must be run in order to set the gain factor, both for MLP and RBF units. The gain factor multiplies the net-input of every hidden unit, previous to the application of the activation function. Therefore, the gain factor is related to the "real" range of weights for MLP units and to the width of the activation function for RBF units.

3.2.4.4 Computation of the MLP-Bias or RBF-Width in the New Hidden Units

The MLP-biases and RBF-widths of the hidden units, as a part of the frequencies, deserve special attention. They could be selected with the same principle as the rest of the frequency, but it is not clear how to do it in some cases (for the Input strategy, for example, it is not easy to obtain a bias term from the data set). Another possibility is to compute them in a deterministic way as a function of the previously selected frequencies, and fix it before selecting the new frequency. This idea allows to compare the different strategies to select frequencies in a fair way. Parenthetically, it could reduce the global computational cost of the algorithm.

We decided to implement the following heuristics:

1. For MLPs, the bias of the new hidden unit is such that, if the rest of the frequency is 0, the output of the hidden unit is the mean value of the error obtained with the previously selected hidden units. It can be easily obtained by computing the residue and then inverting the obtained value with respect to the activation function of the hidden unit. Finally, it must be scaled dividing by the gain factor.

2. For RBFNs, the width of the new hidden unit is simply the standard deviation of the residue. This value will typically be large for the first hidden units, when the system has not learned too much, and smaller as the number of hidden units grows. In other words, the new hidden units become more specialized as the learning process advances. In this case, and different from MLP units, the obtained value must not be scaled by the gain factor.

When they were not computed, the values for the MLP-bias and the RBF-width were selected as the rest of the frequency for the Random and BGA strategies. For the Input and Grid strategies, their values were fixed to 0 for the MLP-bias and 1 for the RBF-width.

3.2.4.5 Computation of the Bias in the Output Units

The bias of every output unit was computed to be the mean of the respective target values, as in [Hwang et al. 1994] (as previously said, output units have linear activation functions). In classification problems, it is equivalent to the frequency of every class. The underlying idea of this heuristic is that the important function that must be learned by the network is the deviation from the mean. Therefore, every real target value can be replaced by "target value - target mean" everywhere in the algorithm in figure 3.3.

3.2.4.6 Selection of the Activation Function

The algorithm in figure 3.3 also selects the activation function for every new hidden unit, as suggested in CC, ILQP, ZM98 or KMP (see section 2.6). Although this activation function can be always the same, in some of the experiments we allowed to selected different activation functions for different hidden units, with the expectation that a better approximation could be obtained with less hidden units. In this case, a list of predefined activation functions is set, and the process of selecting the frequency is repeated for every activation function in the list. The non-linear MLP activation functions used in some of our experiments were:

1. Hyperbolic tangent (*tnh*).

2. Sine (*sin*).

3. Cosine (*cos*).

4. Cosine of the Squared Root and Sine (*crzsin*):

$$crzsin\ (x) = \begin{cases} \cos(\pi\sqrt{-x}) & \text{if } x \leqslant -1 \\ \sin(\frac{\pi}{2}x) & \text{if } -1 < x < +1 \\ \cos(\pi\sqrt{+x}) & \text{if } x \geqslant +1 \end{cases}$$

5. Sine of the Squared Root and Cosine (*srzcos*):

$$srzcos\ (x) = \begin{cases} \sin(\pi\sqrt{-x}) & \text{if } x \leqslant -1 \\ \cos(\frac{\pi}{2}x) & \text{if } -1 < x < +1 \\ \sin(\pi\sqrt{+x}) & \text{if } x \geqslant +1 \end{cases}$$

6. Gaussian (*gau*):

$$gau\ (x) = \frac{1}{\sqrt{2\pi}}\ e^{-\frac{x^2}{2}}.$$

7. Polynomial kernel of degree 2 (*kerpol2*):

$$kerpol2\ (x) = (x+1)^2.$$

3.2.4.7 Hidden Units with Linear Activation Function

Similar to the idea of using direct connections between the input and output layer (see section 2.6), linear hidden units were used in several of our experiments. In

this case, the optimal frequencies can be calculated analytically, solving a linear equations system similar to (3.2), with as many rows as the input dimension I:

$$\begin{pmatrix} \langle x_1, x_1 \rangle & \langle x_2, x_1 \rangle & \cdots & \langle x_I, x_1 \rangle \\ \langle x_1, x_2 \rangle & \langle x_2, x_2 \rangle & \cdots & \langle x_I, x_2 \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_1, x_I \rangle & \langle x_2, x_I \rangle & \cdots & \langle x_I, x_I \rangle \end{pmatrix} \cdot \begin{pmatrix} \omega_k^1 \\ \omega_k^2 \\ \vdots \\ \omega_k^I \end{pmatrix} = \begin{pmatrix} \langle f_k, x_1 \rangle \\ \langle f_k, x_2 \rangle \\ \vdots \\ \langle f_k, x_I \rangle \end{pmatrix} . \tag{3.5}$$

A different linear system must be solved for every output k in the network. Therefore, there must be as many linear hidden units as outputs in the network. The bias term may be computed simply by adding a new row and a new column with the inner products of the constant function with respect to every input x_i. The coefficients of the output k are set to 1 for the k-th linear hidden unit, and 0 for the rest. When new non-linear units are added, the coefficients of the linear hidden units are modified with the rest of the non-linear units, in order to obtain a mixed model.

3.2.4.8 Tuning of the Selected Frequency

Similar to MP (see section 2.6.5.2), an optional tuning of the selected frequencies was allowed. It was implemented with standard BP modified following the technique proposed in [Vogl et al. 1988; Battiti 1989], where the learning rates change dynamically as follows. If the error function between two consecutive epochs has increased, then it is supposed that the learning rates must be too large. In this case the weights change is undone, and the learning rates are decreased. Otherwise, the learning rates are increased ($\Delta E = E_{t+1} - E_t$):

$$\eta_{new} = \begin{cases} \rho \cdot \eta_{old} & \text{if } \Delta E < 0 \\ \sigma \cdot \eta_{old} & \text{otherwise} \end{cases}$$

The parameter ρ uses to be slightly larger than 1 (1.1 is a typical value), whereas σ is chosen to be significantly less than 1 (0.5, for example).

The reason to use this technique was to avoid the adjustment of the parameters of the network. It seems quite clear that different learning rates should be used depending on the number of hidden units already defined in the network (N), the activation function and the range of values of the frequencies. Therefore, an automatic adjustment of the parameters was considered adequate. The momentum was always set to 0. After every step of BP, the optimal coefficients with the new candidate frequency are computed and the learning rates are modified (and the weights change undone if necessary). After a minimum number of epochs, the tuning was stopped when the relative rate of decrease of the error between two successful epochs was less than a certain δ.

3.2.4.9 Bounded Coefficients

The implementation we used had the possibility to upper bound the 1-norm of the coefficients (normalized by the number of coefficients). This option was widely used in the experiments, and its purpose is twofold. On the one hand, it prevents numerical problems derived from bad-conditioned linear systems. On the other, it allows to control the complexity of the resulting model, with the aim of obtaining better generalization results. This idea is present in several theoretical results, such as [Barron 1994; Koiran 1994; Lee et al. 1996; Bartlett 1998; Niyogi and Girosi 1999] (see section 2.5).

Similar to the gain factor, a preprocessing procedure must be run in order to set the maximum 1-norm of the coefficients.

3.2.4.10 Algorithms for *MFT* and *OCMFT*

With the same scheme of the algorithm described in figure 3.3, we can define two additional algorithms:

1. *MFT* ("Maximum Fourier Transform") is a version "matching the residue" of *SAOCIF*: the previous coefficients are not recalculated, and the new term tries to approximate the residue as best as possible. That is, λ_N and ω_N are defined such that $\|f - X_{N-1} - \lambda_N v_{\omega_N}\|^2$ is nearly minimum. This is equivalent to say that $\lambda_N = \langle f - X_{N-1}, v_{\omega_N} \rangle / \|v_{\omega_N}\|^2$ and $|\langle f - X_{N-1}, v_{\omega_N} \rangle|^2 / \|v_{\omega_N}\|^2$ is nearly maximum (see Lemma 2). Therefore, the selected frequency tries to maximize, at every step, the absolute value of the Fourier transform of the residue normalized by the vector norm. The coefficient of the new frequency is the maximum Fourier transform of the residue normalized by the squared vector norm. *MFT* can be implemented with a slight change in the algorithm in figure 3.3: the selected frequency should maximize $|\langle f - X_{N-1}, v_{\omega_N} \rangle|^2 / \|v_{\omega_N}\|^2$ instead of $\|X_N\|^2$ and the coefficient of the new hidden unit is computed as $\lambda_N = \langle f - X_{N-1}, v_{\omega_N} \rangle / \|v_{\omega_N}\|^2$. The previous coefficients are not recalculated. The idea behind *MFT* is exactly the same as in PPR, PPLN or MP without *back-projection* (see section 2.6). When the tuning procedure is performed, and after every step of BP, the coefficient of the new frequency is computed as previously explained. The rest of the tuning procedure is shared with the *SAOCIF* algorithm.

2. *OCMFT* ("Optimal Coefficients after Maximum Fourier Transform") follows the same idea as *MFT*, but the coefficients are recalculated after the optimal *MFT*-frequency is already selected (i.e., just when the frequency is fixed in the network). This optimization of the coefficients is shared with methods like MP with *back-projection*, OMP or the linear version of the ILQP algorithm (see section 2.6).

These two algorithms were tested (see section 3.6) in order to be compared with the algorithm for *SAOCIF* in figure 3.3.

3.2.4.11 Other Implementation Details

The net-input of the hidden units was, in the implementation of the algorithm, always normalized dividing by the number of inputs. The aim of this normalization was making the parameters of the resulting systems (ranges of weights for MLP units, distances for RBF units, etc) more independent on the input dimension.

The linear equations system (3.2) was solved as a bordered system [Faddeeva 1959]. Since the proposed system at step N is equal to the system at step $N-1$, but with a new row and a new column, the system to be solved at step N can be stated as

$$\left(\begin{array}{c|c} A & v \\ \hline v^t & \gamma \end{array}\right) \cdot \left(\frac{x}{\alpha}\right) = \left(\frac{b}{\beta}\right),$$

where $A \cdot x = b$ is the system already solved at step $N-1$, v is a vector with $N-1$ components, and α, β and γ are scalars. The above system is equivalent to

$$\left\{\begin{array}{l} A \cdot x + \alpha v = b \\ v^t \cdot x + \alpha\gamma = \beta \end{array}\right.$$

Therefore, $x = A^{-1} \cdot b - \alpha A^{-1} \cdot v$, and $\alpha = \frac{\beta - v^t \cdot A^{-1} \cdot b}{\gamma - v^t \cdot A^{-1} \cdot v}$. This allows to take profit from the solution of the system at the previous step. In this sense, the matrix A can be inverted only once at every step, previous to the selection of the first frequency, and kept in memory. Whereas the inversion of a matrix A has cubic complexity, the computational cost of $A^{-1} \cdot u$ is quadratic. Instead of directly inverting the matrix, it was decomposed as a QR product, and the corresponding triangular system was solved when needed ($A^{-1} \cdot u$ is the solution of $A \cdot x = u$), also with quadratic cost. The QR decomposition has good numerical properties [Golub and Van Loan 1996].

3.3 Comparison of *SAOCIF* with other Sequential Schemes

Apart from some obvious ones, the main difference of *SAOCIF* with most of the existing sequential methods for regression lays on the criterion to select the new frequencies: whereas most of the systems (as, for example, PPR, MP, PPLN, ILQP or CC, see section 2.6) try to select the frequency that approximates the residue (or the correlation with the residue) at best, *SAOCIF* takes into account the interactions with the previously selected frequencies in order to select the new one. As previously noted, the frequencies such that their associated vectors match the residue are not always the best, even if the coefficients are subsequently optimized (see figure 3.2).

With regard to OLSL, ZM98, KMP with *pre-fitting* and other similar methods that do not try to approximate the residue, *SAOCIF* can be seen as an extension and generalization in several ways. First, it is not restricted to select the candidate frequencies from the points in the data set. In this sense, a number of different heuristics can be used. Second, it is possible to further tune the selected frequency with any non-linear optimization technique. More specifically,

1. OLSL and KMP with *pre-fitting* are equivalent to *SAOCIF* with the Input strategy and without tuning.

2. ZM98 is equivalent to *SAOCIF* with the Random strategy with only 1 candidate frequency and tuning of the frequency.

3.4 Extension to Hilbert Spaces

In this section, we will extend *SAOCIF* definition to general Hilbert spaces. The problem of approximation in Hilbert spaces can be defined as follows: "Let H be a Hilbert space with inner product $\langle , \rangle : H \times H \to \mathbb{C}$, a space of parameters Ω, and $f \in H$ a vector to approximate with vectors $v_\omega = v(\omega)$, $v : \Omega \to H$, $\omega \in \Omega$, such that $\|v_\omega\| \neq 0$ for every $\omega \in \Omega$. We want to find $\omega_1, \omega_2, \cdots \in \Omega$ and $\lambda_1, \lambda_2, \cdots \in \mathbb{C}$ such that

$$\lim_{N \to \infty} \left\| f - \sum_{k=1}^{N} \lambda_k v_{\omega_k} \right\| = 0.$$

As in the previous sections, the term *frequency* refers to every $\omega_1, \omega_2, \cdots \in \Omega$, and *coefficient* to $\lambda_1, \lambda_2, \cdots \in \mathbb{C}$".

This definition is, in essence, the traditional one in approximation of vectors in Hilbert spaces. We can suppose that $\|f\| \neq 0$. If not, the approximation is trivial. The condition $\|v_\omega\| \neq 0$ is equivalent, by the inner product properties, to that of $v_\omega \neq 0$. Observe that every vector $v_\omega \in H$ depends on a parameter $\omega \in \Omega$. Once we fix the parameter, we have a vector in the Hilbert space. In L^2, usually, $\Omega \subseteq \mathbb{C}^p$.

In this setting, the definition of *SAOCIF* stated in section 3.2.1 is also valid for a general Hilbert space H.

3.4.1 Convergence

As it is reasonably expected from its definition, *SAOCIF* converges towards the target vector f under mild conditions. The proofs can be found in section 3.8.

Proposition 1. Let H be a Hilbert space, and $f \in H$. Any *SAOCIF* $\{X_N\}_{N \geqslant 0}$ satisfies the following properties:

(P1a) $\forall N \geqslant 0 \quad \|f - X_{N+1}\|^2 \leqslant \|f - X_N\|^2$.

(P1b) If $M \geqslant N$, then

(P1b1) $\|X_M\|^2 \geqslant \|X_N\|^2$.

(P1b2) $\langle f - X_M, f - X_N \rangle = \|f - X_M\|^2$.

(P1b3) $\langle X_M, f - X_N \rangle \geqslant 0$.

(P1c) Suppose that $\lambda_N^N \neq 0$. The vector v_{ω_N} is orthogonal to the space spanned by $\{v_{\omega_1}, \cdots, v_{\omega_{N-1}}\}$ if and only if the previously selected coefficients do not change between the steps $N-1$ and N, that is, if

$$\forall j : 1 \leqslant j \leqslant N-1 \quad \lambda_j^N = \lambda_j^{N-1}.$$

(P1d) $\{X_N\}_{N \geqslant 0}$ is convergent in H. That is,

$$\exists g \in H \quad \lim_{N \to \infty} \|g - X_N\| = 0.$$

Observe that, by (P1c), the only directions that guarantee that the approximation is optimal without recalculating the coefficients are the orthogonal directions. Hence, if the approximation vectors are not mutually orthogonal, the coefficients must be recalculated. In addition, in order to satisfy (P1d), it is enough for $\|f - X_N\|^2$ to be decreasing and positive, and X_N to satisfy (3.1).

Theorem 1. Let H be a Hilbert space, $f \in H$ and a *SAOCIF* $\{X_N\}_{N \geqslant 0}$. Let g be such that (by P1d) $\lim_{N \to \infty} \|g - X_N\| = 0$, and suppose that for every $\mu \in \mathbb{C}$ and every $\omega_0 \in \Omega$ we have

$$\|f - X_{N+1}\|^2 \leqslant \|f - (X_N + \mu v_{\omega_0})\|^2 + \alpha_N \tag{3.6}$$

for every $N \geqslant 0$. That is, the approximation of f with X_{N+1} is better (up to α_N) than the best approximation of the residue $f - X_N$ that one could achieve with only one vector $v_{\omega_0} \in v(\Omega)$. If $\limsup_{N \to \infty} \alpha_N \leqslant 0$, then:

(T1a) The vector g satisfies:

(T1a1) $\forall \omega_0 \in \Omega \quad \lim_{N \to \infty} \langle g - X_N, v_{\omega_0} \rangle = 0$.

(T1a2) $\forall \omega_0 \in \Omega \quad \langle f, v_{\omega_0} \rangle = \langle g, v_{\omega_0} \rangle$. In particular, we have

(T1a21) $\forall N \geqslant 1 \quad \langle f - g, X_N \rangle = 0$.

(T1a22) $\forall N \geqslant 1 \quad \forall j : 1 \leqslant j \leqslant N \quad \forall M \geqslant N \quad \langle g - X_M, v_{\omega_j} \rangle = 0$.

(T1a3) $\langle f-g, g\rangle = 0$.

(T1a4) There is no subset of vectors in $v(\Omega)$ that approximate f more than g. That is,

$$\|f-g\| = \inf_{\substack{\mu_k\in\mathbb{C}\\ \psi_k\in\Omega}} \left\| f - \sum_k \mu_k v_{\psi_k} \right\|.$$

(T1b) If there exists $A\subseteq\Omega$ such that the set of vectors $\{v_\psi : \psi\in A\}$ spans H, then $\{X_N\}_{N\geqslant 0}$ converges towards f. That is,

$$\lim_{N\to\infty} \|f - X_N\| = 0.$$

(T1c) The rate of approximation of *SAOCIF* is such that for every $N\geqslant 0$

$$\|f - X_{N+1}\|^2 \leqslant (1 - C_{f,N})\cdot\|f - X_N\|^2 + \alpha_N,$$

where $0\leqslant C_{f,N}\leqslant 1$ is an indicator of the best approximation of $f - X_N$ in $v(\Omega)$, that is

$$C_{f,N} = \sup_{\omega_0\in\Omega} \frac{|\langle f - X_N, v_{\omega_0}\rangle|^2}{\|f - X_N\|^2\|v_{\omega_0}\|^2}.$$

Observe that these results are not very restrictive, since the universal approximation capability is a necessary condition for the convergence property. Hence, *SAOCIF* allows us (by selecting Ω and $v(\Omega)$) to choose any (or some) of the multiple vector families satisfying this property. The hypothesis about the tolerance α_N is, in essence, the same as in [Jones 1992], [Barron 1993], [Kůrková and Beliczyński 1995b] or [Kůrková 1998].

The rate of approximation has a general structure where an exponential behavior depending on $1 - C_{f,N}$ combines with α_N. The term $C_{f,N}$ is the maximum squared cosine between $f - X_N$ and its best approximation in $v(\Omega)$. If $v(\Omega)$ contains a basis of H, then $C_{f,N}$ is strictly greater than 0 for every $N\geqslant 0$. But the real rate of approximation may depend on the target function f. In L^2, for example, for any basis and any predetermined threshold $\varepsilon > 0$ there exist functions f such that $C_{f,0}\leqslant\varepsilon$. On the other hand, suppose a space of parameters Ω such that the vectors $v_\omega\in v(\Omega)$, form an orthonormal basis. Let $f = \sum_{k=1}^N \lambda_k v_{\omega_k}$. In this case, $\left|\langle f, v_{\omega_j}\rangle\right|^2 = \left|\lambda_j\right|^2$ and $\|f\|^2 = \sum_{k=1}^N |\lambda_k|^2$, and therefore $C_{f,0} = \max_j \left|\lambda_j\right|^2 / \sum_{k=1}^N |\lambda_k|^2$. Imposing $\max_j \left|\lambda_j\right|^2 \geqslant A$ and $\|f\|^2\leqslant B$ we obtain $C_{f,0}\geqslant \frac{A}{B}$.

Imposing conditions on α_N, as in [Barron 1993; Kůrková and Beliczyński 1995b; Lee et al. 1996], different expressions of the rate of approximation can be obtained. Some of them are summarized in the following result.

Corollary 1. In the same conditions of Theorem 1, the following results hold:

(C1a) If $\alpha_N \leqslant \frac{C_{f,N}}{2} \cdot \|f - X_N\|^2$, then for every $N \geqslant 0$ we have

$$\|f - X_{N+1}\|^2 \leqslant \|f\|^2 \cdot \prod_{i=0}^{N} \left(1 - \frac{C_{f,i}}{2}\right).$$

(C1b) Suppose that there exist constants $A, B > 0$ such that for every $N \geqslant 0$

(a) $C_{f,N} \geqslant \frac{1}{N+B}$

(b) $\alpha_N \leqslant \frac{(A-B+1)(A+1)}{(N+B)(N+A)(N+A+1)} \cdot \|f\|^2$.

Then, for every $N \geqslant 0$ we have

$$\|f - X_{N+1}\|^2 \leqslant \frac{A+1}{N+A+1} \cdot \|f\|^2.$$

This rate of approximation has the same order as the optimal ones that can be found in the literature (see section 2.6), but it has been obtained with a different approximation scheme.

An important remark must be pointed out. The previous results (see the proofs for details) are a consequence of the optimality of the coefficients in *SAOCIF* (property (a) in the definition). The importance of the interacting frequencies (property (b)) lies in the hypothesis that, as it can be seen in figure 3.2, it seems more plausible to find better partial approximations (or equivalently, smaller α_N) selecting the new frequency taking into account the interactions with the previously selected frequencies than, for example, matching the residue as best as possible. Therefore, the rate of approximation is expected to be improved with this strategy.

3.4.2 Specific vectors in $H = L^2$

In theory, the approximation of a function in the Hilbert space L^2 may consist of an infinite number of terms. In practical applications, however, this is not possible. In addition, linear expansions in a single basis are not flexible enough. The information can be diluted across the whole basis [Mallat and Zhang 1993], and the approximation error cannot be made smaller than $O(1/(d\sqrt[d]{n}))$, where d is the dimension of the input to the function [Barron 1993]. This happens even with an orthogonal basis. Therefore, sequential schemes can be used as an alternative to approximations with fixed basis in L^2.

Suppose that the vector to approximate is a square integrable function $f(\vec{x})$. In the definition of *SAOCIF* there are hardly any restrictions about the vectors

$v_{\omega_k}(\vec{x}) = v(\omega_k, \vec{x}) \in H$ used to approximate f. The only required condition is to have a norm different from 0. Thus, the method can be applied to a number of vector families very common in the literature: FNNs, including both MLPs and RBFNs, Fourier series, algebraic polynomials, wavelets, etc. In fact, the universal approximation capability of a family of functions is enough to apply *SAOCIF* with guarantee of convergence to f (whenever the hypotheses of Theorem 1 are satisfied).

If we only have a data set D, the inner products can be approximated with:

$$\left\langle v_{\omega_i}, v_{\omega_j} \right\rangle \cong \frac{1}{L} \sum_{x \in D} v_{\omega_i}(x) \overline{v_{\omega_j}(x)}$$

$$\left\langle f, v_{\omega_j} \right\rangle \cong \frac{1}{L} \sum_{x \in D} f(x) \overline{v_{\omega_j}(x)}.$$

In this case we will suppose that the integral is defined with regard to the probability measure of the problem represented by the data set. This is similar to approximating the expectation of a random variable by the arithmetic mean.

3.5 Experimental Motivation

We performed some experiments on both artificial (section 3.6) and benchmark data sets (section 3.7). The main objectives of these experiments were twofold.

On the one hand, to compare *SAOCIF* with *MFT* and *OCMFT* (see section 3.2.4.10 for a description of *MFT* and *OCMFT*). In particular, the effect of the interacting frequencies for both approximation and generalization purposes was studied. According to the hypothesis made in the previous sections, the rate of approximation should be improved. In addition, it would be possible to obtain simpler models (regarding the number of hidden units) with the same bias, so that, in terms of the Bias/Variance decomposition, an improvement in the generalization performance is also expected.

On the other hand, to test the algorithm for *SAOCIF* and the implementation details described in section 3.2.4. The main points we were interested to test were:

1. The strategies to select the frequencies (see section 3.2.4.2).

2. The activation functions in the hidden layer (see section 3.2.4.6).

3. The tuning of the selected frequency (see section 3.2.4.8).

The experiments were designed so as to keep a reduced number of parameters fixed. The rest of parameters were found experimentally. As a consequence, there was a great number of configurations to test, and very different configurations with

very similar results were sometimes found. For every parameter decision, we followed the strategy that "the (numerically) best one is selected", regardless of similar results with different parameters.

3.6 Experiments on Artificial Data Sets

3.6.1 HEA Data Sets

A brief description of the HEA functions [Hwang et al. 1994] and the data sets used in our experiments can be found in appendix A. These data sets were mainly used to compare *SAOCIF* with *MFT* and *OCMFT*.

3.6.1.1 Methodology

The following methodology was used in these experiments:

1. All the experiments were performed:
   (a) With MLP units, and selecting the activation function for every unit, in addition to the frequencies (see section 3.2.4.6).
   (b) In the same conditions for *MFT*, *OCMFT* and *SAOCIF*. That is, the only differences among the three algorithms were the selection of the frequency together with the computation of the coefficients. The rest of parameters were the same and they were selected for every model in an independent way.
2. First, we selected
   (a) The gain factor (see section 3.2.4.3),
   (b) The bound on the 1-norm of the coefficients (see section 3.2.4.9),
   (c) Whether the bias is computed in a deterministic way or not for every new hidden unit (see section 3.2.4.4).

   for every model (*MFT*, *OCMFT* and *SAOCIF*). To this end, the Grid strategy (without tuning) was used to select the frequencies. The grid consisted of 625 candidate frequencies. A maximum of 30 hidden units were added to the initial architecture. This step was performed with the original training and test sets in [Hwang et al. 1994], considering the latter as a validation set. We selected the parameter values which allowed to obtain the minimum validation set error, chosen to be a local minimum among a finite number of values. The deterministic computation of the bias was selected for most of the models and data sets.

3. Second, we tested the Random and BGA strategies with the parameters selected in the previous step for every model (*MFT*, *OCMFT* and *SAOCIF*). A maximum of 50 hidden units were added to the initial architecture. The parameters for every strategy were:

   (a) Random: 5,000 random frequencies in $[-0.5, +0.5] \times [-0.5, +0.5]$.

   (b) BGA:

      i. Population size: 100.

      ii. q-elitism: 5.

      iii. Truncation rate: 40%.

      iv. Maximum number of evaluations of the fitness function: 5,000.

      v. The initial population was chosen in $[-0.5, +0.5] \times [-0.5, +0.5]$ at random.

      vi. Every component of the frequency is limited to take values within $[-0.5, +0.5]$.

   Observe that the total number of candidate frequencies tested for the Random and BGA strategies was exactly the same, with identical ranges.

   Every strategy was tested, in addition, with and without tuning the selected frequencies at every step. The parameters for the tuning procedure (see section 3.2.4.8) were: $\rho = 1.05$, $\sigma = 0.5$ and $\delta = 0.0005$. The minimum number of epochs was 100, in batch mode. For every function, every parameter configuration was trained with every one of the 10 different training sets, and tested on the test set constructed for this problem (see appendix A). Note that the test set is different from the original one in [Hwang et al. 1994], which was used in the previous step as validation set.

3.6.1.2 Results

Results are shown in tables 3.1 to 3.4 as the average, over 10 runs with the respective data sets for every function, of the minimum squared test set errors (i.e., at the optimal hidden unit). Every table is related to a different strategy and tuning configuration. Figures in boldface indicate the best results for every function. Numbers in brackets are $\hat{\sigma_n}/\sqrt{n}$, the standard errors[2] estimated from the sample standard deviation $\hat{\sigma_n}$. The average number of hidden units where these minima are achieved is also shown. For the HEA1-NF data set, *SAOCIF* stopped with a fewer

[2]Under normality assumptions, the confidence interval can be computed from this value. For example, the deviation of the true value from the observed mean $\overline{x_n}$ will be less than $1.96\hat{\sigma_n}/\sqrt{n}$ with a probability of 0.95 [Flexer 1996].

	Test Error			**Num. Hidden Units**		
Data Set	*MFT*	*OCMFT*	*SAOCIF*	*MFT*	*OCMFT*	*SAOCIF*
HEA1-NF	0.06 (0.1)	**0.00 (0.0)**	**0.00 (0.0)**	49.3	20.3	9.6
HEA2-NF	29.81 (2.6)	1.18 (0.5)	0.18 (0.1)	49.1	47.1	44.1
HEA3-NF	694.11 (27.2)	31.41 (6.9)	5.57 (2.2)	49.2	45.0	49.1
HEA4-NF	69.29 (6.9)	24.32 (1.8)	9.80 (3.7)	49.4	45.8	47.3
HEA5-NF	49.22 (7.8)	14.79 (3.4)	7.76 (1.0)	49.8	49.4	46.2
HEA1-WN	14.37 (3.1)	**10.70 (1.6)**	15.37 (2.9)	39.1	4.5	4.4
HEA2-WN	110.85 (6.8)	99.13 (7.1)	89.47 (6.3)	30.9	19.1	15.4
HEA3-WN	704.35 (50.5)	303.65 (18.9)	256.01 (32.6)	49.9	33.6	31.6
HEA4-WN	213.67 (8.0)	202.91 (14.7)	**120.58 (11.5)**	47.0	27.5	17.4
HEA5-WN	174.63 (9.0)	211.82 (18.0)	173.35 (5.7)	35.0	23.7	19.4

Table 3.1: HEA data sets: Squared test set error and number of hidden units for the Random strategy without tuning of the selected frequency.

	Test Error			**Num. Hidden Units**		
Data Set	*MFT*	*OCMFT*	*SAOCIF*	*MFT*	*OCMFT*	*SAOCIF*
HEA1-NF	1.33 (1.2)	**0.00 (0.0)**	**0.00 (0.0)**	49.2	17.1	8.0
HEA2-NF	29.50 (2.6)	14.02 (10.4)	0.33 (0.1)	50.0	41.9	46.1
HEA3-NF	689.72 (48.7)	24.35 (5.0)	12.55 (2.3)	49.8	47.4	49.3
HEA4-NF	56.90 (6.0)	28.86 (4.3)	7.13 (1.6)	48.8	49.6	47.0
HEA5-NF	50.88 (6.8)	17.43 (1.4)	14.54 (5.3)	49.7	47.9	47.3
HEA1-WN	64.80 (17.3)	12.44 (2.3)	14.37 (2.9)	25.2	4.6	4.1
HEA2-WN	131.01 (10.0)	99.05 (5.9)	88.38 (12.8)	21.6	18.7	13.7
HEA3-WN	802.32 (33.4)	302.49 (36.3)	232.30 (18.8)	48.7	33.4	31.8
HEA4-WN	274.25 (10.9)	223.88 (10.6)	137.13 (8.9)	40.1	31.4	19.3
HEA5-WN	214.71 (10.6)	217.80 (10.4)	172.53 (6.5)	32.9	28.1	19.3

Table 3.2: HEA data sets: Squared test set error and number of hidden units for the Random strategy with tuning of the selected frequency.

number of hidden units because the training error was less than 0.000001, so that it made no sense adding new hidden units. Figures 3.4 and 3.5 show a comparison of the evolution of the average training and test errors of *OCMFT* and *SAOCIF* with respect to the number of hidden units for the BGA strategy without tuning. Noise free data sets results are shown in figure 3.4. Noisy data sets results are shown in figure 3.5.

Some conclusions of these experiments can be summarized as follows:

1. Regarding the overall behavior, *SAOCIF* obtains better results than *OCMFT* which in turn compares favorably with *MFT*. This fact can be understood by looking at the number of hidden units of the obtained results:

	Test Error			Num. Hidden Units		
Data Set	*MFT*	*OCMFT*	*SAOCIF*	*MFT*	*OCMFT*	*SAOCIF*
HEA1-NF	**0.00 (0.0)**	**0.00 (0.0)**	**0.00 (0.0)**	49.6	20.6	9.6
HEA2-NF	31.03 (2.9)	4.73 (2.4)	**0.08 (0.0)**	48.8	41.6	42.6
HEA3-NF	687.01 (26.4)	31.24 (10.3)	**3.61 (0.9)**	49.5	47.8	48.4
HEA4-NF	81.00 (7.4)	32.89 (5.9)	8.14 (2.1)	48.8	43.6	45.4
HEA5-NF	49.00 (7.1)	13.53 (2.2)	8.91 (1.1)	50.0	48.3	48.3
HEA1-WN	12.44 (3.0)	12.15 (2.5)	13.74 (2.9)	35.3	4.7	4.1
HEA2-WN	113.58 (6.2)	102.57 (12.0)	93.49 (8.9)	31.9	20.5	14.9
HEA3-WN	717.01 (47.4)	369.00 (66.4)	**216.18 (27.6)**	49.4	38.0	27.3
HEA4-WN	219.25 (8.9)	227.41 (14.9)	132.94 (9.3)	45.8	28.1	17.3
HEA5-WN	174.99 (8.9)	203.58 (13.9)	**165.82 (5.7)**	34.6	26.8	20.1

Table 3.3: HEA data sets: Squared test set error and number of hidden units for the BGA strategy without tuning of the selected frequency.

	Test Error			Num. Hidden Units		
Data Set	*MFT*	*OCMFT*	*SAOCIF*	*MFT*	*OCMFT*	*SAOCIF*
HEA1-NF	3.33 (2.3)	**0.00 (0.0)**	**0.00 (0.0)**	49.7	19.0	7.8
HEA2-NF	31.93 (3.3)	20.62 (8.8)	0.28 (0.1)	49.7	43.4	45.5
HEA3-NF	684.79 (59.4)	25.60 (4.2)	8.83 (2.0)	49.6	48.4	46.9
HEA4-NF	60.73 (4.2)	34.32 (5.4)	**5.61 (1.2)**	48.5	46.5	46.9
HEA5-NF	50.69 (6.6)	18.21 (2.4)	**6.88 (0.7)**	49.8	48.3	48.2
HEA1-WN	64.07 (20.4)	12.14 (2.4)	14.49 (2.7)	29.0	4.7	3.9
HEA2-WN	127.65 (7.4)	96.87 (6.6)	**86.89 (13.2)**	19.6	20.2	13.7
HEA3-WN	780.89 (28.3)	304.99 (26.5)	219.34 (19.4)	48.4	37.1	34.3
HEA4-WN	270.23 (8.1)	213.52 (8.0)	128.35 (11.7)	40.2	29.9	17.9
HEA5-WN	210.53 (11.2)	228.97 (15.9)	168.24 (4.4)	35.3	28.5	19.7

Table 3.4: HEA data sets: Squared test set error and number of hidden units for the BGA strategy with tuning of the selected frequency.

(a) For noise free data sets the number of hidden units may be around 50, the maximum number of allowed hidden units. This is due to the fact that overfitting was not observed during the learning process with these data sets (see figure 3.4). Therefore, the best results are obtained by those models that are able to fit more accurately the data. According to the claims previously made (see section 3.4), *SAOCIF* allows to find better approximations with the same number of hidden units as *OCMFT* or *MFT*. The same happens when *OCMFT* is compared to *MFT*.

(b) For noisy data sets, there is a high correlation between the number of hidden units and the goodness of the model: those models that attain

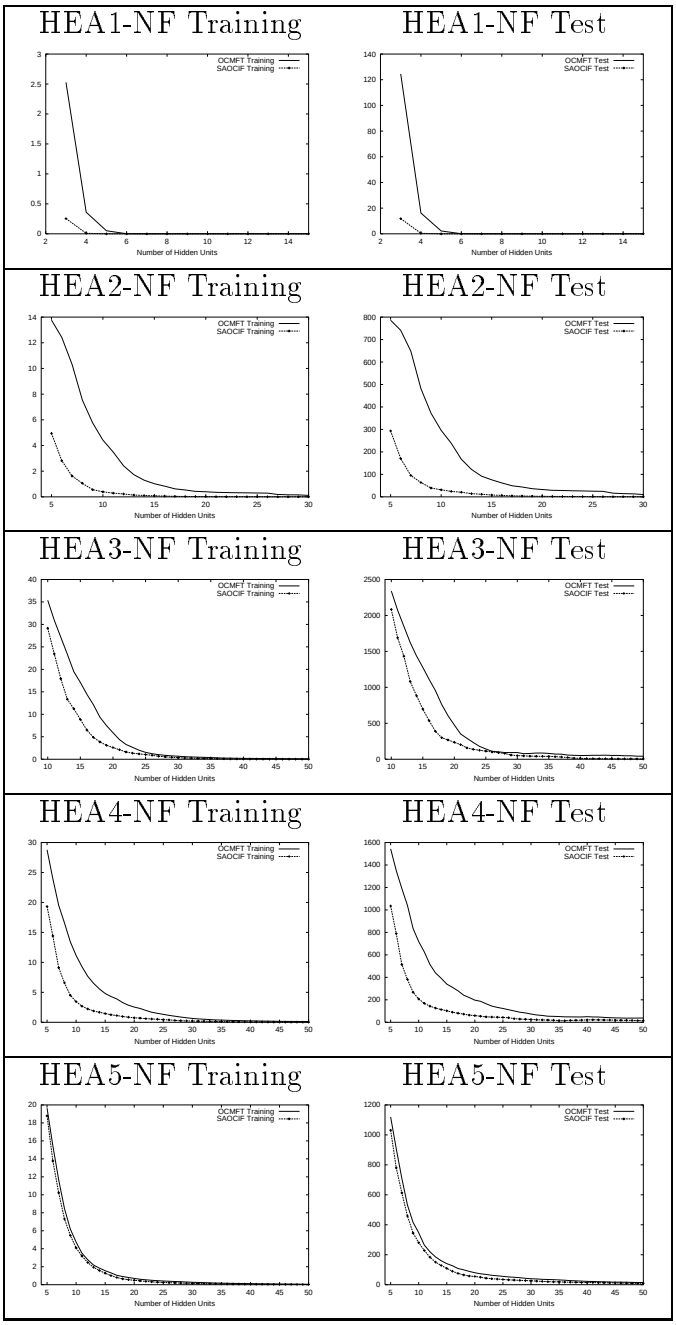

Figure 3.4: Noise free HEA data sets: Comparison of the evolution of the average training and test error of *OCMFT* and *SAOCIF* with respect to the number of hidden units for the BGA strategy without tuning.

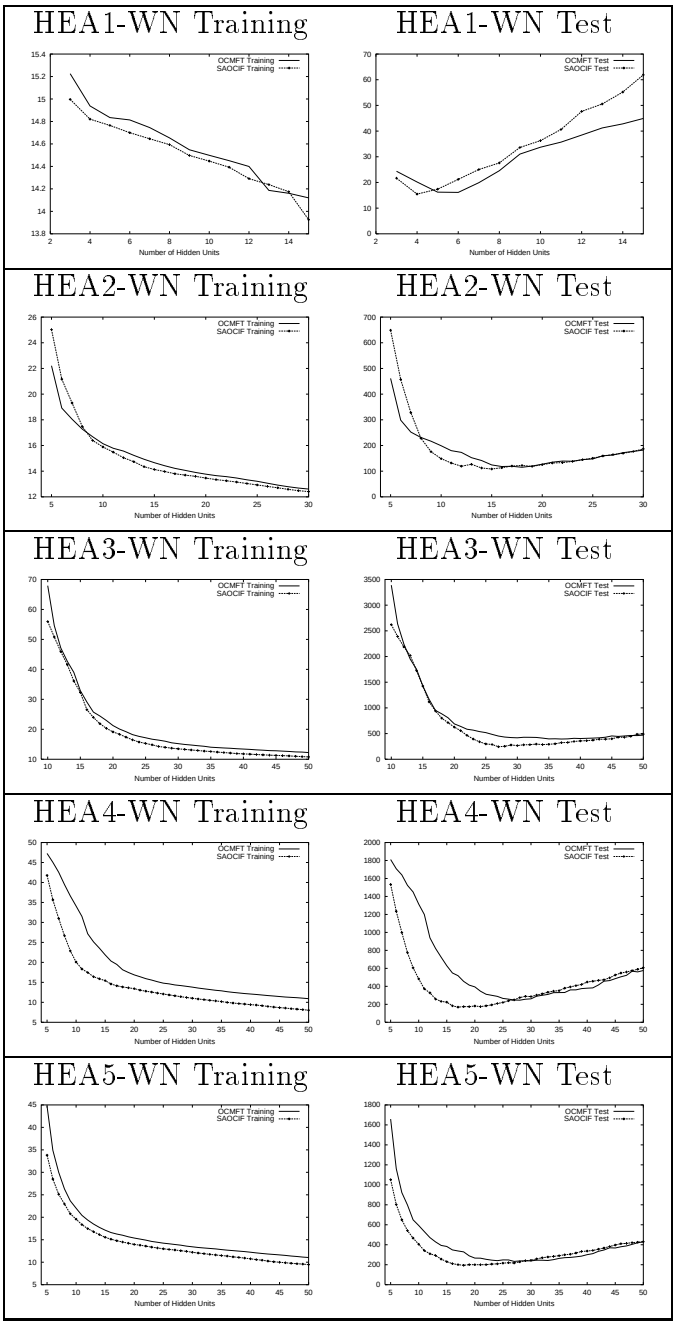

Figure 3.5: Noisy HEA data sets: Comparison of the evolution of the average training and test set error of *OCMFT* and *SAOCIF* with respect to the number of hidden units for the BGA strategy without tuning.

their minima with less hidden units usually obtain better results. Looking at figure 3.5, we can see that *SAOCIF* obtains simpler models (in terms of the number of hidden units), with the same empirical risk, than *OCMFT*. According to the Bias/Variance trade-off, the minimum test set error is expected to be smaller for *SAOCIF* than for *OCMFT*. The same happens with respect to *MFT*.

Therefore, the effect of the interacting frequencies for both approximation and generalization purposes is confirmed. It can be observed that the relative differences among methods are greater for noise free data sets than for noisy ones.

2. Regarding the strategy to select the frequencies, the BGA strategy appears to be the best one, specially for *SAOCIF*. Surprisingly, the Random strategy achieves results very similar to the BGA strategy, and in some cases superior. This can be due to the fact that the probability of finding a good frequency among 5,000 random attempts may be high. This is not so probable in problems where the input dimension is larger (see section 3.7).

3. In these experiments, tuning the selected frequencies does not seem to help to improve consistently the results, although the obtained networks are better in some cases. As we will see, however, there exist other problems where an improvement is obtained when the frequencies are tuned (see section 3.6.2).

Comparing the computational cost among the models, we observed that *MFT* is faster than *OCMFT*, which in turn is faster than *SAOCIF*, as expected. Taking the computational cost of *MFT* as 1, the relative mean computational costs of *OCMFT* and *SAOCIF* are 1.08 and 1.64, respectively.

In the subsequent experiments, only the *SAOCIF* scheme will be tested.

3.6.2 The *Two Spirals* Data Set

A description of the *Two Spirals* data set can be found in appendix A.

3.6.2.1 Methodology

The following methodology was used to test the *Two Spirals* problem:

1. All the experiments were performed:

   (a) For the following four activation functions: hyperbolic tangent (tnh), sine (sin), cosine (cos), in the MLP model and Gaussian (gau) in the RBFN one. Hidden units had the same activation function ('Act.F.' in the

tables). The motivation for the choice of these activation functions was to test the performance of sinusoidal MLPs and to compare the obtained results with classical MLP and RBFN activation functions.

(b) Computing the bias for every new hidden unit in a deterministic way, and fixing it before selecting the new frequency, as explained in section 3.2.4.4, except for the hyperbolic tangent function, where the bias was computed as the rest of the frequency[3].

2. First, we selected the gain factor for every activation function, together with the bound on the 1-norm of the coefficients. To this end, the Random strategy (500 random frequencies) was used to select the frequencies, with the original training and validation sets. We selected the gain factor and bound of the 1-norm of the coefficients that allowed to obtain the minimum validation set error over the mean of 5 runs. The gain factor and the maximum 1-norm were chosen to be a local minimum among a finite number of values. The addition of hidden units was stopped when the training set was completely learned.

3. The second step consisted in selecting the parameters for the three strategies under study: Random, BGA and Input. These parameters were:

   (a) For the Random strategy, the final number of random frequencies[4]. was selected among 500, 750, 1,000 and 2,000. The range of values to look for component frequencies was $[-0.5, +0.5]$, as explained in section 3.2.4.2.

   (b) For the BGA strategy:

      i. Population size: 50, 75 or 100 frequencies.
      ii. q-elitism: 1 or 5.
      iii. Truncation rate: 20%, 30% or 40%.
      iv. The initial population was randomly chosen in the same range as the Random strategy. Every component of the frequency is limited to take values in that range.

   This step was performed with 500 evaluations of the fitness function. For the subsequent step, the maximum number of evaluations of the fitness function was 5,000. The BGA procedure may also be stopped when the relative error did not decrease more than 0.01% in several consecutive generations. This number of generations was selected by looking at the evolution of the BGA procedure for the chosen parameters (population size, q-elitism and truncation rate).

[3]For the hyperbolic tangent, we obtained better results without computing the bias of the new hidden unit. For the Input strategy with this activation function, the bias was set to 0.

[4]In our experiments, only exceptionally the final number of random frequencies selected was 2,000.

Act.F.	**Strategy**	**Tuning**	**Test**	**NHid**	**Tuning**	**Test**	**NHid**
tnh	Random	No	100.00%	103.80	Yes	100.00%	94.40
tnh	BGA	No	100.00%	92.60	Yes	100.00%	88.60
tnh	Input	No	NP	-	Yes	NP	-
cos	Random	No	99.48%	53.80	Yes	100.00%	48.80
cos	BGA	No	100.00%	51.20	Yes	100.00%	47.40
cos	Input	No	100.00%	58.00	Yes	100.00%	31.00
sin	Random	No	99.48%	51.20	Yes	100.00%	46.40
sin	BGA	No	100.00%	53.00	Yes	100.00%	47.60
sin	Input	No	100.00%	58.00	Yes	100.00%	30.60
gau	Random	No	100.00%	93.40	Yes	100.00%	89.80
gau	BGA	No	100.00%	86.20	Yes	100.00%	95.20
gau	Input	No	100.00%	112.00	Yes	100.00%	78.40

Table 3.5: Percentage of correctly classified patterns on the test set for the *Two Spirals* data set with *SAOCIF* and different parameter configurations. An 'NP' value means "Not Possible", indicating that the learning of the training set was unsatisfactory.

(c) For the Input strategy, the gain factor was selected again, since we observed that the value obtained in the previous step was not always well suited for this strategy.

This step was performed with the same data sets as the first step. Similarly, we selected the parameters that allowed to obtain the minimum validation set error over the mean of 5 runs.

4. Third, we tested the three aforementioned strategies of selection of frequencies for every activation function with the selected parameters in the previous steps. This step was performed with the original training, validation and test sets for this problem. Every strategy was tested with and without tuning the selected frequencies at every step. The parameters for the tuning procedure (see section 3.2.4.8) were: $\rho = 1.05$, $\sigma = 0.5$ and $\delta = 0.0005$. The minimum number of epochs was 100, in batch mode. For every function, every parameter configuration was trained 5 times. No more hidden units were added when the mean sum-of-squares error in the training set was less than 0.01 or when numerical problems were encountered (this only happened for the Input strategy with the hyperbolic tangent function).

3.6.2.2 Results

Results are shown in table 3.5 as the percentage of correctly classified patterns on the test set by the average-output committee of the networks obtained when the

validation error was minimum (column 'Test'). The average number of hidden units where this minimum is achieved is also shown (column 'NHid').

In this case, we can observe that:

1. All the strategies achieve similar (and very good) results with all the activation functions and strategies tested, except for the hyperbolic tangent function with the Input strategy. It seems very difficult to learn this problem with the hyperbolic tangent function when the points in the data set are used as frequencies. We also obtained unsatisfactory results after an exhaustive experimentation with Support Vector Machines for this activation function.

2. Consistently, tuning the frequency allows to obtain similar solutions with less hidden units.

3. The behavior of the Input strategy is very good for the sine, cosine and Gaussian activation functions, specially when the frequency is tuned: the solutions have less hidden units and they can be obtained with a much lower computational cost than the Random and BGA strategies (see below).

4. As expected, this is a very hard problem for sigmoidal activation functions, but it could be adequately learned with *SAOCIF* and a suitable strategy.

The computational cost was quite variable among the activation functions, since it depends strongly (and non-linearly) on the number of hidden units of the obtained solutions. Given an activation function, in contrast, the relative computational cost among the different strategies was more stable. Anyway, the Input strategy was always the fastest one. As an example, taking the computational cost of the Input strategy[5] as 1, the relative computational cost for the sine activation function with tuning (the simplest model in our experiments) was 3.78 and 22.36 for the Random and the BGA strategies, respectively.

3.7 Experiments on Benchmark Data Sets

In this section the results of the experiments for *SAOCIF* with benchmark data sets are shown. A brief description of the data sets used in these experiments can be found in appendix A.

3.7.1 Methodology

The methodology used to test these benchmark data sets with *SAOCIF* was quite similar to that of the *Two Spirals* problem:

[5] With the current version of the program, it took 96 seconds on a Pentium IV processor.

1. All the experiments were performed:

   (a) For the following four activation functions: hyperbolic tangent (tnh), sine (sin), cosine (cos), in the MLP model and Gaussian (gau) in the RBFN one. The motivation for the choice of these activation functions was to test the performance of sinusoidal MLPs and to compare the obtained results with classical MLP and RBFN activation functions.

   (b) Computing the bias for every new hidden unit in a deterministic way, and fixing it before selecting the new frequency, as explained in section 3.2.4.4.

   (c) With stratified Cross-Validation (CV) [Stone 1974; Kohavi 1995]. Previous to every CV, the examples in the data set were randomly shuffled. For every training, no more hidden units were added when the error on the validation set (see below) did not improve for 5 consecutive hidden units. A maximum of 50 hidden units was allowed.

2. First, we selected the gain factor for every activation function, together with the bound on the 1-norm of the coefficients. To this end, the Random strategy (500 random frequencies) was used to select the frequencies. We performed 5 runs of a 5-fold CV with the whole data set. We selected the gain factor and bound of the 1-norm of the coefficients that allowed to obtain the minimum mean validation error over the 5 runs[6]. The gain factor and the maximum 1-norm of the coefficients were chosen to be a local minimum among a finite number of values.

3. Second, we selected the parameters for the three strategies (Random, BGA and Input) under study. The parameters were selected as for the *Two Spirals* data set (see section 3.6.2). 5 runs of a 5-fold CV with the whole data set were performed. Similar to the previous step, we selected the parameters that allowed to obtain the minimum mean validation error over the 5 runs. For the Input strategy, in addition, the computation (or not) of the bias in the new hidden unit was reconsidered.

4. The third step was devoted to test the three aforementioned strategies of selection of frequencies for every activation function with the parameters selected in the previous steps. We constructed FNNs where all the hidden units had the same activation function and mixed models combining linear and non-linear activation functions, as explained in section 3.2.4.7. This is indicated in the

[6]In this CV experiments, we use the folds that are not in the training set as validation data. In sections 4.5 and 4.6 they are used as test data.

Act.F.	Strategy	Tuning	Test	Mse	NHid
gaulin	BGA	No	77.75% (0.68)	0.64	6.27
coslin	BGA	Yes	77.73% (0.54)	0.65	5.15
gau	Input	Yes	77.65% (0.60)	0.63	6.74
gaulin	Input	No	77.59% (0.48)	0.63	8.62
gaulin	Input	Yes	77.54% (0.71)	0.63	8.02
sin	Random	No	77.52% (0.42)	0.64	5.75
sin	BGA	No	77.49% (0.52)	0.64	4.68
sinlin	Random	Yes	77.44% (0.83)	0.65	5.96
tnhlin	Random	Yes	77.33% (0.62)	0.63	7.20
linear	-	-	77.29% (0.40)	0.65	-
sin	Random	Yes	77.28% (0.53)	0.64	4.66
cos	BGA	No	77.28% (0.58)	0.64	6.01
coslin	BGA	No	77.25% (0.63)	0.64	5.78
tnhlin	Random	No	77.20% (0.58)	0.64	8.03
tnhlin	Input	No	77.20% (0.52)	0.63	8.35
tnhlin	Input	Yes	77.10% (0.40)	0.64	4.99
coslin	Random	Yes	76.97% (0.43)	0.65	6.33

Table 3.6: Test set results for the *Pima Indians Diabetes* data set with *SAOCIF* and different parameter configurations.

tables of results (column 'Act.F.') as 'fun' or 'funlin', where 'fun' is the non-linear activation function. Every strategy was tested with and without tuning the selected frequencies at every step, giving 18 parameter configurations for every activation function. The parameters for the tuning procedure (see section 3.2.4.8) were: $\rho = 1.05$, $\sigma = 0.5$ and $\delta = 0.0005$. The minimum number of epochs was 100, in batch mode. At this step we performed a double 5-4-fold CV [Ripley 1995] as follows. We performed a 5-fold CV (the outer CV) to obtain five folds (4 folds to "learn" and 1 fold to test). Then, the 4 folds of the "learning set" of the outer CV were used as follows: 3 folds to train and 1 fold to validate, as in a 4-fold CV (the inner CV). Therefore, the number of trained models in a double 5-4-fold CV is 20. For every activation function, 5 runs of every parameter configuration were performed with the double 5-4-fold CV procedure previously explained, giving a total of 100 runs per model tested.

3.7.2 Results

Results are shown in tables 3.6 to 3.10 as the average percentage of correctly classified patterns on the test sets of the double 5-4-fold CV by the average-output committee of the networks obtained when the validation error was minimum (col-

Act.F.	**Strategy**	**Tuning**	**Test**	**Mse**	**NHid**
sin	Input	No	97.12% (0.21)	0.10	5.46
tnh	Random	No	96.98% (0.22)	0.11	5.02
tnh	BGA	No	96.95% (0.26)	0.11	4.94
tnh	BGA	Yes	96.95% (0.29)	0.11	5.37
gau	Input	No	96.83% (0.22)	0.11	9.20
cos	BGA	Yes	96.78% (0.24)	0.11	4.12
sin	BGA	No	96.78% (0.25)	0.11	4.83
sin	BGA	Yes	96.75% (0.34)	0.11	4.41
sin	Input	Yes	96.75% (0.26)	0.11	4.85
gau	Random	No	96.75% (0.27)	0.11	6.75
cos	Random	No	96.75% (0.25)	0.11	7.25
coslin	Random	Yes	96.72% (0.25)	0.11	6.01
coslin	Input	No	96.69% (0.33)	0.11	8.96
gau	Input	Yes	96.66% (0.23)	0.11	7.21
gaulin	BGA	Yes	96.66% (0.26)	0.12	7.96
tnh	Random	Yes	96.60% (0.27)	0.11	4.87
linear	-	-	95.86% (0.24)	0.15	-

Table 3.7: Test set results for the *Wisconsin Breast Cancer* data set with *SAOCIF* and different parameter configurations.

umn 'Test'). More specifically, every committee is formed by the different models obtained in the inner CV (with the same test set). The mean squared error on the test set (column 'Mse') and the average number of hidden units where this minimum is achieved (column 'NHid') are also shown. Numbers in brackets are $\hat{\sigma_n}/\sqrt{n}$, the standard errors[7] estimated from the sample standard deviation $\hat{\sigma_n}$.

Every table shows four configurations for every non-linear activation function: the two best configurations with and without tuning the selected frequencies. The tables are ordered top-down by the percentage of correctly classified patterns. Results for least-squares linear FNNs with the same experimental setting are also shown ('linear' in the tables).

Some conclusions of these experiments can be summarized as follows:

1. Regarding the overall generalization, the results are very satisfactory. The resulting solutions have good generalization results with a few number of hidden units. *SAOCIF* results in tables 3.6 to 3.10 are competitive with other results found in the literature (see table A.2 in appendix A).

[7] Under normality assumptions, the confidence interval can be computed from this value. For example, the deviation of the true value from the observed mean $\overline{x_n}$ will be less than $1.96\hat{\sigma_n}/\sqrt{n}$ with a probability of 0.95 [Flexer 1996].

Act.F.	Strategy	Tuning	Test	Mse	NHid
sinlin	BGA	No	88.65% (1.07)	0.42	3.63
gaulin	BGA	No	88.39% (0.83)	0.42	5.42
coslin	Input	No	88.39% (0.93)	0.44	7.19
gaulin	BGA	Yes	88.13% (0.87)	0.41	5.41
sinlin	Input	No	88.00% (0.78)	0.43	4.85
gaulin	Random	Yes	88.00% (0.99)	0.42	5.87
coslin	Input	Yes	87.87% (1.28)	0.45	4.41
gaulin	Random	No	87.87% (1.06)	0.45	5.38
coslin	BGA	Yes	87.74% (0.99)	0.45	4.45
coslin	BGA	No	87.48% (0.94)	0.45	5.62
linear	-	-	87.42% (0.83)	0.43	-
sin	BGA	Yes	87.35% (0.95)	0.43	1.56
sinlin	Input	Yes	87.35% (0.83)	0.44	2.70
tnhlin	Random	No	87.35% (0.83)	0.45	6.98
tnhlin	Random	Yes	87.23% (1.20)	0.43	5.47
tnhlin	BGA	No	87.23% (1.16)	0.45	5.51
tnhlin	Input	Yes	86.84% (0.91)	0.47	4.62

Table 3.8: Test set results for the *Hepatitis* data set with *SAOCIF* and different parameter configurations.

2. We can observe that there exist data sets, such as the *Pima Indians Diabetes*, where the results are very similar among all the systems. Linear FNNs also achieve similar results. In our opinion, it is quite difficult to obtain conclusions from the experiments with this data set. Results in the literature also support this claim. The *Wisconsin Breast Cancer* and *Hepatitis* problems also have quite similar results among the methods, but it seems clear that they can take some profit from non-linear activation functions. The *Ionosphere* and *Sonar* problems are, in our opinion, the best ones in order to obtain interesting conclusions, since they are clearly non-linear and with important differences among different configuration parameters.

3. Tuning the selected frequency sometimes helps to improve the overall results, but it does not do it either significantly or consistently. Most of the times, solutions obtained with tuning have less hidden units than without it. However, this fact is not always translated into a better performance, a somewhat surprising result, and different from the experiments with artificial data sets.

4. Non-linear MLP activation functions different from classical sigmoidal ones, such as sines or cosines, may be satisfactorily used (the hyperbolic tangent function obtains, globally, worse results than sinusoidal activation functions).

Act.F.	Strategy	Tuning	Test	Mse	NHid
gau	Input	No	95.37% (0.44)	0.19	16.19
gau	Input	Yes	94.57% (0.49)	0.24	14.51
gau	BGA	No	92.69% (0.57)	0.28	7.79
gaulin	BGA	Yes	92.17% (0.63)	0.33	12.48
cos	Input	No	91.37% (0.52)	0.31	10.40
sin	Input	No	90.00% (0.68)	0.33	10.19
tnh	Input	No	89.77% (0.70)	0.35	10.75
coslin	Input	No	89.31% (0.61)	0.38	11.33
sinlin	Input	No	88.97% (0.76)	0.41	9.92
tnhlin	Input	No	88.97% (0.79)	0.42	11.83
tnh	Random	Yes	88.34% (0.55)	0.44	4.07
tnh	BGA	Yes	88.17% (0.69)	0.40	5.66
cos	BGA	Yes	87.37% (0.82)	0.45	2.78
cos	Random	Yes	87.31% (0.53)	0.46	3.09
sin	BGA	Yes	87.20% (0.93)	0.46	2.50
sin	Random	Yes	87.20% (0.60)	0.45	3.14
linear	-	-	87.00% (0.52)	0.54	-

Table 3.9: Test set results for the *Ionosphere* data set with *SAOCIF* and different parameter configurations.

In some particular cases, it seems that there are activation functions better suited for some problems (for example, Gaussian RBFs for the *Ionosphere* data set). Linear hidden units have, in some cases, a positive influence on the results when combined with non-linear ones.

5. Regarding the number of hidden units, we can see that there is not a clear tendency, and it depends to a great extent on the activation function, the strategy to select the new frequency and the tuning procedure. Solutions with Gaussian RBFs, for example, tend to have more hidden units than MLPs. Solutions obtained using the BGA strategy are usually smaller than for the Input strategy. Different from the HEA data sets (see section 3.6.1), there is not a clear correlation between the number of hidden units and the goodness of the model. We think that the main reason for this fact is the small number of examples in the data sets with respect to the input dimension. Not to forget that these results are taken in the minimum of the validation set. The presence of noisy or meaningless variables in the data sets may also help to introduce more confusion on this aspect.

6. Regarding the strategy to select the frequencies, both the BGA and the Input strategies appear to be superior to the Random selection. This is more clear

Act.F.	Strategy	Tuning	Test	Mse	NHid
gau	Input	Yes	82.93% (1.48)	0.62	5.75
sinlin	BGA	No	81.76% (0.96)	0.73	10.35
gau	Input	No	81.07% (1.20)	0.54	20.48
sin	BGA	Yes	80.98% (1.21)	0.59	3.49
coslin	BGA	No	80.29% (1.14)	0.72	11.26
coslin	BGA	Yes	80.20% (1.25)	0.75	9.77
sinlin	Random	No	80.00% (0.94)	0.77	15.13
sinlin	BGA	Yes	79.80% (1.19)	0.75	10.45
gaulin	Input	Yes	79.80% (1.16)	0.83	11.55
cos	BGA	No	79.71% (1.38)	0.64	3.47
gaulin	Input	No	78.54% (1.31)	0.80	18.75
cos	BGA	Yes	78.34% (1.63)	0.64	3.12
tnh	Input	Yes	78.24% (1.14)	0.69	3.72
tnhlin	BGA	Yes	77.85% (1.27)	0.90	5.28
tnhlin	BGA	No	77.27% (1.00)	0.96	5.20
tnh	Random	No	77.27% (1.00)	0.72	5.45
linear	-	-	74.39% (0.83)	0.99	-

Table 3.10: Test set results for the *Sonar* data set with *SAOCIF* and different parameter configurations.

if we look only to the *Ionosphere* and *Sonar* problems, the most interesting ones. In these cases, the dimension of the input space makes more difficult to find a good frequency at random. In some cases, selecting the frequencies from the points in the data set seems well suited not only for RBFNs, as commonly used, but also for MLPs. Since the computational cost for the Input strategy is quite smaller than that of the BGA one[8], the Input strategy is, in our opinion, the most interesting one for further research.

[8]The relative computational costs of both strategies was quite variable, although the Input strategy was always much faster than the BGA one. In the results shown, the Input strategy only took from a few seconds to a few minutes on a Pentium IV processor, and it was from 30 to 300 times faster than the BGA strategy.

3.8 Proofs of the Theoretical Results

3.8.1 Proof of Lemma 1

(L1a) By (3.1) we have $\left\langle f - \left(\sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} + \lambda_j^N v_{\omega_j}\right), v_{\omega_j} \right\rangle = 0$, and as a consequence

$$\left\langle f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k}, v_{\omega_j} \right\rangle = \lambda_j^N \left\langle v_{\omega_j}, v_{\omega_j} \right\rangle = \lambda_j^N \left\| v_{\omega_j} \right\|^2.$$

(L1b) Decomposing $X_N = \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} + \lambda_j^N v_{\omega_j}$ we have

$$\begin{aligned} \|f - X_N\|^2 = & \left\| f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} \right\|^2 + \left|\lambda_j^N\right|^2 \left\| v_{\omega_j} \right\|^2 \\ & - 2\mathrm{Re}\left(\left\langle f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k}, \lambda_j^N v_{\omega_j} \right\rangle \right). \end{aligned}$$

By (3.1) we can state

$$\begin{aligned} & \|f - X_N\|^2 = \\ & = \left\| f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} \right\|^2 + \left|\lambda_j^N\right|^2 \left\| v_{\omega_j} \right\|^2 - \\ & \qquad 2\mathrm{Re}\left(\left\langle X_N - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k}, \lambda_j^N v_{\omega_j} \right\rangle \right) \\ & = \left\| f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} \right\|^2 + \left|\lambda_j^N\right|^2 \left\| v_{\omega_j} \right\|^2 - 2\mathrm{Re}\left(\left\langle \lambda_j^N v_{\omega_j}, \lambda_j^N v_{\omega_j} \right\rangle \right) \\ & = \left\| f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} \right\|^2 + \left|\lambda_j^N\right|^2 \left\| v_{\omega_j} \right\|^2 - 2\left|\lambda_j^N\right|^2 \left\| v_{\omega_j} \right\|^2 \\ & = \left\| f - \sum_{k=1,k\neq j}^{N} \lambda_k^N v_{\omega_k} \right\|^2 - \left|\lambda_j^N\right|^2 \left\| v_{\omega_j} \right\|^2. \end{aligned}$$

(L1c) Expressing f as $(f - X_N) + X_N$ we have

$$\|f\|^2 = \|f - X_N\|^2 + \|X_N\|^2 + 2\mathrm{Re}\left(\langle f - X_N, X_N \rangle\right).$$

By (3.1), $2\mathrm{Re}\left(\langle f - X_N, X_N \rangle\right) = 0$ holds.

(L1d) By the definition of X_N we have

$$\|X_N\|^2 = \langle X_N, X_N\rangle = \left\langle \sum_{k=1}^{N} \lambda_k^N v_{\omega_k}, X_N \right\rangle = \sum_{k=1}^{N} \lambda_k^N \langle v_{\omega_k}, X_N\rangle.$$

Since X_N satisfies (3.1) we have

$$\forall k : 1 \leqslant k \leqslant N \quad \langle v_{\omega_k}, X_N\rangle = \langle v_{\omega_k}, f\rangle,$$

Hence,

$$\|X_N\|^2 = \sum_{k=1}^{N} \lambda_k^N \langle v_{\omega_k}, f\rangle = \sum_{k=1}^{N} \lambda_k^N \overline{\langle f, v_{\omega_k}\rangle}.$$

(L1e) $\|f - X_N\|^2 = \langle f - X_N, f - X_N\rangle = \langle f - X_N, f\rangle - \langle f - X_N, X_N\rangle$.

By (3.1), $\langle f - X_N, X_N\rangle = 0$ holds.

□

3.8.2 Proof of Lemma 2

Since $P_N(v_\omega)$ is the residue of the best approximation of $f - X_{N-1}$ with the vector v_ω, the value of $P_N(v_\omega)$ can be obtained as follows. First, impose (3.1) in order to obtain the optimal coefficient $\mu_\omega \in \mathbb{C}$. In this case, μ_ω is such that $\langle f - X_{N-1} - \mu_\omega v_\omega, v_\omega\rangle = 0$. By inner product's properties, $\mu_\omega = \frac{\langle f - X_{N-1}, v_\omega\rangle}{\|v_\omega\|^2}$. This is always well defined, since the vectors norms do not vanish in $v(\Omega)$. Using (L1c) and (L1d) with $f = f - X_{N-1}$ and $X_N = \mu_\omega v_\omega$, we have

$$\begin{aligned} P_N(v_\omega) &= \|f - X_{N-1} - \mu_\omega v_\omega\|^2 \\ &= \|f - X_{N-1}\|^2 - \|\mu_\omega v_\omega\|^2 \\ &= \|f - X_{N-1}\|^2 - \mu_\omega \overline{\langle f - X_{N-1}, v_\omega\rangle} \\ &= \|f - X_{N-1}\|^2 - \frac{|\langle f - X_{N-1}, v_\omega\rangle|^2}{\|v_\omega\|^2}. \end{aligned}$$

In particular, P_N is always well defined. Since the norm and the inner product are continuous functions with regard to every one of its arguments, P_N is continuous at $v(\Omega)$.

□

3.8.3 Proof of Proposition 1

(P1a) Evident, since the coefficients of X_{N+1} are optimal:
$\|f - X_{N+1}\|^2 \leqslant \left\|f - X_N + 0 \cdot v_{\omega_{N+1}}\right\|^2 = \|f - X_N\|^2$.

(P1b1) Evident, combining (P1a) and (L1c).

(P1b2) Expressing $f - X_N$ as $f - X_M + X_M - X_N$ we have

$$\langle f - X_M, f - X_N\rangle = \|f - X_M\|^2 + \langle f - X_M, X_M - X_N\rangle .$$

By (3.1), $\langle f - X_M, X_M - X_N\rangle = 0$ holds.

(P1b3) By (P1b2) and (L1e) we have

$$\begin{aligned}\|f - X_M\|^2 &= \langle f - X_M, f - X_N\rangle = \langle f, f - X_N\rangle - \langle X_M, f - X_N\rangle \\ &= \|f - X_N\|^2 - \langle X_M, f - X_N\rangle .\end{aligned}$$

The proof finishes with (P1a).

(P1c) The necessity is clear by (3.1). To prove the sufficiency, suppose that $\forall j : 1 \leqslant j \leqslant N-1 \;\; \lambda_j^N = \lambda_j^{N-1}$. Hence $X_N = X_{N-1} + \lambda_N^N v_{\omega_N}$ holds. By (3.1) we have

$$\forall j : 1 \leqslant j \leqslant N \quad \left\langle f - X_N, v_{\omega_j}\right\rangle = 0,$$

$$\forall j : 1 \leqslant j \leqslant N-1 \quad \left\langle f - X_{N-1}, v_{\omega_j}\right\rangle = 0.$$

Therefore, $\forall j : 1 \leqslant j \leqslant N-1$ we have

$$0 = \left\langle f - X_N, v_{\omega_j}\right\rangle = \left\langle f - X_{N-1} - \lambda_N^N v_{\omega_N}, v_{\omega_j}\right\rangle = \left\langle \lambda_N^N v_{\omega_N}, v_{\omega_j}\right\rangle .$$

Since $\lambda_N^N \neq 0$, the vectors v_{ω_N} and v_{ω_j} are orthogonal for every j between 0 and $N-1$.

(P1d) Since H is complete, it is enough to prove that $\lim_{N,M\to\infty} \|X_M - X_N\|^2 = 0$. Suppose that $M > N$. Expressing $X_M - X_N$ as $(X_M - f) + (f - X_N)$, and using (P1b2) we have

$$\begin{aligned}\|X_M - X_N\|^2 &= \|f - X_M\|^2 + \|f - X_N\|^2 - 2\mathrm{Re}\left(\langle f - X_M, f - X_N\rangle\right) \\ &= \|f - X_M\|^2 + \|f - X_N\|^2 - 2\|f - X_M\|^2 \\ &= \|f - X_N\|^2 - \|f - X_M\|^2.\end{aligned}$$

Since the sequence $\{\|f - X_N\|^2\}_{N\geqslant 0}$ is decreasing and positive (see (P1a)), it is convergent. Hence,

$$\lim_{N,M\to\infty} \|X_M - X_N\|^2 = \lim_{N,M\to\infty} \left(\|f - X_N\|^2 - \|f - X_M\|^2\right) = 0. \qquad \square$$

3.8.4 Proof of Theorem 1

(T1a1) By Schwartz inequality we have

$$\forall \omega_0 \in \Omega \quad |\langle g - X_N, v_{\omega_0} \rangle| \leqslant \|g - X_N\| \, \|v_{\omega_0}\| \, .$$

Using (P1d),

$$\forall \omega_0 \in \Omega \quad \lim_{N \to \infty} |\langle g - X_N, v_{\omega_0} \rangle| \leqslant \|v_{\omega_0}\| \lim_{N \to \infty} \|g - X_N\| = 0.$$

(T1a2) Let $\omega_0 \in \Omega$. By hypothesis, for every $N \geqslant 0$ and every $\mu \in \mathbb{C}$

$$\begin{aligned} \|f - X_{N+1}\|^2 &\leqslant \|f - (X_N + \mu v_{\omega_0})\|^2 + \alpha_N \\ &= \|f - X_N\|^2 - 2\mathrm{Re}\left(\langle f - X_N, \mu v_{\omega_0} \rangle\right) + |\mu|^2 \|v_{\omega_0}\|^2 + \alpha_N \end{aligned}$$

holds. Expressing $f - X_N$ as $f - g + g - X_N$ we have

$$\begin{aligned} &\|f - X_{N+1}\|^2 - \|f - X_N\|^2 \leqslant \\ &\quad |\mu|^2 \|v_{\omega_0}\|^2 - 2\mathrm{Re}\left(\langle f - g, \mu v_{\omega_0} \rangle\right) - 2\mathrm{Re}\left(\langle g - X_N, \mu v_{\omega_0} \rangle\right) + \alpha_N. \end{aligned}$$

Hence,

$$\begin{aligned} &2\mathrm{Re}\left(\langle f - g, \mu v_{\omega_0} \rangle\right) - |\mu|^2 \|v_{\omega_0}\|^2 \leqslant \\ &\quad \leqslant \|f - X_N\|^2 - \|f - X_{N+1}\|^2 - 2\mathrm{Re}\left(\langle g - X_N, \mu v_{\omega_0} \rangle\right) + \alpha_N \\ &\quad \leqslant \|f - X_N\|^2 - \|f - X_{N+1}\|^2 + 2\,|\langle g - X_N, \mu v_{\omega_0} \rangle| + \alpha_N \\ &\quad = \|f - X_N\|^2 - \|f - X_{N+1}\|^2 + 2\,|\mu|\,|\langle g - X_N, v_{\omega_0} \rangle| + \alpha_N. \end{aligned}$$

for every $\mu \in \mathbb{C}$. Let $\mu_0 = \frac{\langle f-g, v_{\omega_0} \rangle}{\|v_{\omega_0}\|^2}$, and $\varepsilon > 0$.

Using (P1a), (T1a1), and the hypothesis about α_N, there exists N_0 such that $\forall N \geqslant N_0$,

$$\begin{gathered} \|f - X_N\|^2 - \|f - X_{N+1}\|^2 \leqslant \varepsilon/3, \\ 2\,|\mu_0|\,|\langle g - X_N, v_{\omega_0} \rangle| \leqslant \varepsilon/3, \\ \alpha_N \leqslant \varepsilon/3. \end{gathered}$$

Thus we have

$$2\mathrm{Re}\left(\langle f - g, \mu_0 v_{\omega_0} \rangle\right) - |\mu_0|^2 \|v_{\omega_0}\|^2 \leqslant \varepsilon.$$

Since

$$\langle f - g, \mu_0 v_{\omega_0} \rangle = \overline{\mu_0} \, \langle f - g, v_{\omega_0} \rangle = \overline{\mu_0} \mu_0 \|v_{\omega_0}\|^2 = |\mu_0|^2 \|v_{\omega_0}\|^2,$$

$2\mathrm{Re}\left(\langle f - g, \mu_0 v_{\omega_0} \rangle\right) = 2|\mu_0|^2 \|v_{\omega_0}\|^2$ holds, and therefore

$$\forall \varepsilon \geqslant 0 \quad |\mu_0|^2 \|v_{\omega_0}\|^2 = 2\mathrm{Re}\left(\langle f - g, \mu_0 v_{\omega_0} \rangle\right) - |\mu_0|^2 \|v_{\omega_0}\|^2 \leqslant \varepsilon.$$

Hence, $|\mu_0|^2 \|v_{\omega_0}\|^2 = 0$. Since $\|v_{\omega_0}\|^2 \neq 0$, we have $\mu_0 = 0$. Thus, by definition of μ_0, $\langle f - g, v_{\omega_0} \rangle = 0$ for every $\omega_0 \in \Omega$. In particular we have

(T1a21) $\forall N \geqslant 1 \quad \langle f-g, X_N\rangle = 0$

(T1a22) Using (3.1), $\forall j : 1 \leqslant j \leqslant N \quad \forall M \geqslant N$

$$\left\langle g - X_M, v_{\omega_j}\right\rangle = \left\langle g, v_{\omega_j}\right\rangle - \left\langle X_M, v_{\omega_j}\right\rangle = \left\langle g, v_{\omega_j}\right\rangle - \left\langle f, v_{\omega_j}\right\rangle = 0.$$

(T1a3) Expressing g as $g - X_N + X_N$, and using (T1a21), we can derive

$$\langle f-g, g\rangle = \langle f-g, g-X_N\rangle + \langle f-g, X_N\rangle = \langle f-g, g-X_N\rangle .$$

By Schwartz inequality we have

$$|\langle f-g, g-X_N\rangle| \leqslant \|f-g\| \, \|g-X_N\| .$$

Using (P1d), $\lim_{N\to\infty} |\langle f-g, g-X_N\rangle| = 0$. Therefore, $\langle f-g, g\rangle = 0$.

(T1a4) By (T1a2), any combination $\sum_k \mu_k v_{\psi_k}$ in $v(\Omega)$ satisfies $\langle f-g, \sum_k \mu_k v_{\psi_k}\rangle = 0$. Hence we have

$$\|f-g\|^2 = \langle f-g, f-g\rangle = \left\langle f-g, f - \sum_k \mu_k v_{\psi_k}\right\rangle - \langle f-g, g\rangle .$$

Using (T1a3) and Schwartz inequality we have

$$\|f-g\|^2 = \left|\left\langle f-g, f-\sum_k \mu_k v_{\psi_k}\right\rangle\right| \leqslant \|f-g\| \left\|f - \sum_k \mu_k v_{\psi_k}\right\| .$$

Therefore, $\|f-g\| \leqslant \|f - \sum_k \mu_k v_{\psi_k}\|$ for any vector combination. The other inequality is clear, since for every $N \geqslant 0$

$$\inf_{\substack{\mu_k \in \mathbb{C} \\ \psi_k \in \Omega}} \left\|f - \sum_k \mu_k v_{\psi_k}\right\| \leqslant \|f - X_N\| \leqslant \|f-g\| + \|g - X_N\| .$$

The proof finishes using (P1d).

(T1b) It is derived immediately from (T1a4).

(T1c) By hypothesis we have

$$\|f - X_{N+1}\|^2 \leqslant \inf_{\mu\in\mathbb{C}, \omega_0 \in \Omega} \|f - (X_N + \mu v_{\omega_0})\|^2 + \alpha_N = \inf_{\omega_0 \in \Omega} P_{N+1}(v_{\omega_0}) + \alpha_N .$$

Using Lemma 2 and the definition of $C_{f,N}$ we have

$$\begin{aligned}\|f - X_{N+1}\|^2 &\leqslant \inf_{\omega_0 \in \Omega} \left(\|f - X_N\|^2 - \frac{|\langle f - X_N, v_{\omega_0} \rangle|^2}{\|v_{\omega_0}\|^2} \right) + \alpha_N \\ &= \|f - X_N\|^2 - \sup_{\omega_0 \in \Omega} \frac{|\langle f - X_N, v_{\omega_0} \rangle|^2}{\|v_{\omega_0}\|^2} + \alpha_N \\ &\leqslant \|f - X_N\|^2 - C_{f,N} \|f - X_N\|^2 + \alpha_N \\ &= (1 - C_{f,N}) \|f - X_N\|^2 + \alpha_N.\end{aligned}$$

□

3.8.5 Proof of Corollary 1

(C1a) If $\alpha_N \leqslant \frac{C_{f,N}}{2} \cdot \|f - X_N\|^2$, then by (T1c) we have

$$\begin{aligned}\|f - X_{N+1}\|^2 &\leqslant (1 - C_{f,N}) \cdot \|f - X_N\|^2 + \frac{C_{f,N}}{2} \cdot \|f - X_N\|^2 \\ &= \left(1 - \frac{C_{f,N}}{2}\right) \cdot \|f - X_N\|^2.\end{aligned}$$

An induction argument finishes the proof, using that $X_0 = 0$.

(C1b) This property can also be proved by an induction argument. By (L1c), the result is true for $N = 0$:

$$\|f - X_1\|^2 = \|f\|^2 - \|X_1\|^2 \leqslant \|f\|^2 = \frac{A+1}{A+1} \cdot \|f\|^2.$$

Assume as inductive hypothesis that

$$\|f - X_N\|^2 \leqslant \frac{A+1}{N+A} \cdot \|f\|^2.$$

Then, using (T1c) and the hypotheses we have

$$\begin{aligned}
\|f - X_{N+1}\|^2 &\leqslant (1 - C_{f,N}) \cdot \|f - X_N\|^2 + \alpha_N \\
&\leqslant \left(1 - \frac{1}{N+B}\right) \cdot \frac{A+1}{N+A} \cdot \|f\|^2 + \\
&\qquad \frac{(A-B+1)(A+1)}{(N+B)(N+A)(N+A+1)} \cdot \|f\|^2 \\
&= \|f\|^2 \cdot \frac{A+1}{N+A} \cdot \left[\frac{N+B-1}{N+B} + \frac{A-B+1}{(N+B)(N+A+1)}\right] \\
&= \|f\|^2 \cdot \frac{(A+1)\left[(N+B-1)(N+A+1) + A - B + 1\right]}{(N+A)(N+B)(N+A+1)} \\
&= \|f\|^2 \cdot \frac{(A+1)(N+B)(N+A)}{(N+A)(N+B)(N+A+1)} \\
&= \|f\|^2 \cdot \frac{A+1}{N+A+1}
\end{aligned}$$

as desired. □

Chapter 4

Performing Feature Selection with Multi-Layer Perceptrons

This chapter is devoted to the comparison of different criteria to perform Feature Selection (FS) with Multi-Layer Perceptrons (MLPs) and the Sequential Backward Selection (SBS) procedure within the *wrapper* approach. The experimental results suggest that performance can be significantly improved when some critical decision points are properly set.

4.1 Introduction

In general, there is no reason to think that, during the learning process of an MLP, irrelevant variables will not be used by the system in order to fit the training set. For new data, the performance of a system that takes into account this kind of variables can be far to be optimal. This problem can be worsened if some important features are missing or the number of available examples for the problem is small. Unfortunately, it is not possible to know *a priori* whether we are in these situations or not. Therefore, FS procedures become necessary for MLPs.

From an SML point of view, trying to minimize the value of the loss function is surely the optimal criterion for FS [Liu and Motoda 1998], and that will be the saliency used in this work. Saliencies different from this one are mainly motivated and justified by computational cost reasons. Although this kind of saliencies may work in certain situations, there is a lack of theoretical results that support them. The sensitivity of the outputs with respect to the input units, or the relative variance in the hidden layer net-input with respect to the inputs, for example, may be large for an irrelevant variable if the system uses that variable to learn the training set. As previously mentioned, it may also depend on whether the number of available examples for the problem is large enough to filter irrelevant variables or not.

As many other authors (see section 2.7.2), we will also use the SBS procedure. We think that the SBS procedure may help to detect irrelevant variables, specially in the first steps[1]. As pointed out in section 2.7.2, there exist several models that remove more than one feature in the same step. Although it can be justified by computational cost reasons, the elimination of several features in the same step may lose the interactions among the variables. We will consider the SBS procedure with the elimination of one feature at every step.

Regarding the evaluation of an FS procedure with MLPs, there are at least three critical decision points when, as in our case, the saliency used is the value of the loss function:

1. The stopping criterion in the training phase. That is, which properties must satisfy the trained network? For example, the network could be trained either until a (probably local) minimum in the training set or a minimum in the validation set. Surely, the features to be eliminated using the same saliency definition may be very different depending on the stopping criterion. Suppose that the model presents the negative effect of overfitting. Although it seems that performing early stopping with a validation set may be a promising idea, it could also be argued that training the network until a (probably overtrained) local minimum of the training set forces the system to use all the available variables as much as possible [Romero et al. 2003]. In this situation, irrelevant variables could stand out more than in the minimum of a validation set when the system is not allowed to use them.

2. Where and how should the loss function be measured? In the training set? In a validation set? As an approximation not only dependent on the data set? The easiest one is to measure it either in the training or a validation set. The measurement of the loss function in a validation set is, probably, the most reasonable choice, since it can be considered as an estimator of the generalization error. But selecting the minimum number of features that allows to fit the training set as well as the whole set of variables does could also be thought as a quite reasonable way to eliminate useless features. In this case, the loss function should be measured in the training set.

3. The network retraining with respect to the computation of the saliency. The saliency of a feature can be computed following two approaches:

   (a) First, the network is trained. Then, every feature is temporarily removed and the value of the loss function is computed. Therefore, the saliency of every feature is computed in the same trained network. Most of the

[1]Recently, the SBS procedure has also been successfully applied with linear Support Vector Machines [Guyon et al. 2002]

FS procedures for MLPs described in the literature compute the saliency in this way (see below). The whole SBS procedure involves to train N_f networks, where N_f is the number of variables.

(b) For every feature, the network is trained with that feature temporarily removed. For every trained network, the value of the loss function is computed. This procedure is, clearly, computationally more expensive than the previously described one, since it involves to train $\frac{N_f \cdot (N_f+1)}{2}$ networks.

Note that these two ways of computing the saliency may give very different values for the same feature. Suppose that a trained network uses a certain feature to fit the data and a new network is trained without it. The new network will use other features to fit the data, obtaining a different solution than that obtained with all the variables. There is no reason to think that the saliencies of every feature remain unchanged with respect to the original network. The same happens if the feature values are substituted by its average value. Therefore, a more reliable estimation of the saliency is obtained by retraining the network with every feature temporarily removed. To our knowledge, the only models that retrain the network at every step with every feature temporarily removed/added are those described in [Steppe et al. 1996; Onnia et al. 2001], and none of them is a pure SBS procedure (see section 2.7.2). The reason for this fact may be, similar to the existence of so many saliency definitions, the high computational cost of the whole process.

There exists a lack of comparative studies among these issues in the literature. In this chapter we describe an experimental evaluation of the SBS procedure for MLPs designed on the basis of the previously explained points: the stopping criterion in the training phase, the data set where the loss function is measured and the retraining of the network with every feature temporarily removed previous to computing the saliency. The experiments were performed for linear and non-linear models. We consider that this study can shed light to the further design of FS methods for MLPs.

Experimental results (see sections 4.5 and 4.6) suggest that the increase in the computational cost associated with retraining the network with every feature temporarily removed previous to computing the saliency can be rewarded with a significant performance improvement, specially if non-linear models are used. Regarding the data set where the value of the loss function is measured, it seems clear that the SBS procedure for MLPs takes profit from measuring it in a validation set. A somewhat non-intuitive conclusion is drawn by looking at the stopping criterion, where it can be seen that forcing overtraining may be as useful as using early stopping within the SBS procedure for MLPs. There exist an important improvement in the

Algorithm
Let V_1 be the whole set of N_f features
for $N = 1$ **up to** $N_f - 1$ **do**
Train the network with V_N until a certain stopping criterion is satisfied, and keep its generalization performance
for each $v \in V_N$ **do**
Set $V = V_N - \{v\}$
Optionally, train the network with V (and the same stopping criterion)
Obtain the saliency of v by computing the value of the loss function E_v on a certain data set (see the text for details)
end for
Set $V_{N+1} = V_N - \{v^*\}$, where v^* is the feature associated to the lowest value of the loss function E_{v^*} in the previous loop
end for
Return V_{N^*}, where N^* corresponds to the best generalization performance of the network at any step of the previous loop
end Algorithm

Figure 4.1: A basic SBS procedure for MLPs with the value of the loss function as the saliency.

overall results with respect to learning with the whole set of variables. Although the model can be further improved, the good results obtained are mainly due, in our opinion, to a proper detection of irrelevant variables.

The basic SBS algorithm for MLPs is described in section 4.2, together with the critical points tested. The experimental work is described in sections 4.3, 4.4, 4.5 and 4.6.

4.2 A Basic SBS Procedure for MLPs

4.2.1 The Basic Scheme

The basic SBS procedure for MLPs that we used can be seen in figure 4.1. It works roughly as follows. The outer loop follows the scheme of the classical SBS procedure, where a feature is permanently eliminated at every step. The inner loop selects the variable to eliminate: every feature is temporarily removed, the network is optionally trained, and the value of the loss function is computed (on a certain data set). The variable such that, when removed, gives the lowest value of the loss function is permanently eliminated. Typically, it is expected that, starting from the

whole set of features, performance improves until a subset of features remains where the elimination of further variables results in performance degradation. This is the subset of features returned by the algorithm.

4.2.2 Critical Decision Points in the Algorithm

The algorithm in figure 4.1 has three critical decision points:

1. The stopping criterion of the network training. This is the first critical decision point discussed in the previous section. We tested two different stopping criteria. The first one is to stop where a minimum of a validation set is achieved. The second one is to stop at the (probably overtrained) point where a minimum of the training set is obtained.

2. The data set where the value of the loss function is measured to compute the saliency. This is the second critical decision point previously discussed. In our experiments, the loss function was computed either in the training set or a validation set.

3. Whether the network is retrained or not after the feature is temporarily removed and previous to computing the saliency. That is, whether every feature is temporarily removed before retraining the network or the network is first trained and then every feature is temporarily removed in the same trained network. We tested both schemes. This is the third critical decision point discussed in the previous section.

Therefore, there are four combinations of stopping criterion/loss measurement data set: Training/Training, Training/Validation, Validation/Training and Validation/Validation:

1. Training/Training: The network is trained until a minimum of the training set, and the loss function is computed in the training set. Therefore, variables that are not necessary to fit the training set will be removed. It could be thought that this configuration belongs to the family of methods that only use the training set to compute the saliency (see section 2.7.2).

2. Training/Validation: The network is trained until a (probably overtrained) minimum of the training set. The system is forced to use all the available variables as much as possible. In this situation, a validation set is used to remove the variables.

3. Validation/Training: It makes no sense.

4. Validation/Validation: The network is trained until a minimum of a validation set. The loss function is also computed in the validation set. Intuitively, this seems the most suitable strategy.

Combined with the two possibilities regarding the retraining of the network, we have a total of six configurations to test and compare.

4.3 Experimental Motivation

We performed some experiments on both artificial (section 4.5) and benchmark data sets (section 4.6) for classification tasks. For every data set, the six aforementioned configurations were tested with the SBS procedure for MLPs described in figure 4.1:

1. Training/Training with and without retraining.
2. Training/Validation with and without retraining.
3. Validation/Validation with and without retraining.

The experiments were designed so that every configuration was tested with the same network architecture and parameters, in order to introduce the least external variability in the experiments. Similar to *SAOCIF*, we followed the strategy that "the (numerically) best one is selected", regardless of similar results with different parameters.

A computational cost as small as possible was also needed in the whole process. From our experience (see [Sopena et al. 1999a]), MLPs using sine activation functions (and an appropriate choice of initial parameters, namely range of weights and different learning rates for every layer) usually needs less hidden units and learns faster than MLPs with sigmoid functions when both types are trained with Back-Propagation (BP). Several theoretical results also reinforce this claim [Suzuki 1998]. So then, for non-linear models we decided to use MLPs with one hidden layer of sinusoidal units, the hyperbolic tangent in the output layer, and trained with standard BP in pattern mode.

4.4 General Methodology

The experimental methodologies for the different data sets were very similar among them, although they were not exactly equal. In the following, their common features are described:

1. All the experiments were performed with stratified Cross-Validation (CV). Previous to every CV, the examples in the data set were randomly shuffled.

2. First, the number of hidden units, the initial range of weights and the learning rates were chosen so as to achieve a small and smoothly decreasing training error in a reasonable number of epochs and an acceptable validation error. 5 runs of a 5-fold CV were performed. These parameters were used in the subsequent steps, and they are different for every data set.

3. Second, we tested the six aforementioned configurations with the SBS procedure for MLPs described in figure 4.1. For the Training/Training configuration, 5 runs of a 5-fold CV were conducted. For the Training/Validation and Validation/Validation configurations, 5 runs of a double 5-4-fold CV were performed [Ripley 1995][2] (for a description of the double 5-4-fold CV, see section 3.7). In order to obtain different models for every configuration, we tested them with several learning combinations:

   (a) With and without bias in the hidden units (bias was always used in the output units). The main reason for this was the observation that, while biases in the hidden units are commonly used, they are not always necessary with sinusoidal hidden units [Sopena et al. 1999a].

   (b) Either increasing the number of epochs (in a fixed amount) after the elimination of every variable or maintaining it constant over the whole SBS procedure. The reason to increase the number of epochs was to compensate the loss in the capability of fitting the data set when the features are eliminated (the network parameters had been adjusted with the whole set of variables and they were not modified during the SBS procedure). These parameters are also different for every data set.

4.5 Experiments on Artificial Data Sets

4.5.1 The *Augmented XOR* Data Set

We performed an experimental comparison of the aforementioned configurations with an augmented version of the *XOR* data set, where we artificially added 11 irrelevant features to the two original variables. Some of them were noisy. The details of the whole set of features can be found in appendix A.

[2] When we use CV, we consider that the folds that are not in the training set can be used as validation or test data, depending on the case. For a double CV there is no confusion. For a single CV, in contrast, they can be test data (as in this step) or validation data (as in the previous step).

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	Var
Yes	Training/Training	99.40%	0.06	x_2 x_1
No	Training/Training	99.33%	0.06	x_1 x_6 x_7
No	Training/Validation	99.28%	0.06	x_1 x_7 x_6
Yes	Training/Validation	99.22%	0.05	x_7 x_1 x_2
No	Validation/Validation	99.15%	0.07	x_1 x_6
Yes	Validation/Validation	99.08%	0.06	x_2 x_1

Table 4.1: Test set results for the *Augmented XOR* data set after the SBS procedure for MLPs and different combinations of retraining/stopping criterion/loss measurement data set.

4.5.1.1 Methodology

These are the specific parameters for this data set in the general methodology described in section 4.4:

1. In the first step, the selected number of hidden units was 20, with an initial range of weights of 3.0 and 500 epochs.

2. In the second step, the number of epochs to increase was 50, when applied.

4.5.1.2 Results

The best results among the different learning combinations for every configuration are shown in table 4.1 (column 'Test') as the average percentage of correctly classified patterns on the respective test sets in the following trained networks:

1. For the Training/Training configuration, the networks with minimum squared test set error among the networks with minimum training set error after every variable is permanently eliminated.

2. For the Training/Validation and Validation/Validation configurations, the networks with minimum squared test set error among the networks with minimum validation set error after every variable is permanently eliminated.

The mean squared error on the test set (column 'Mse') and the variables (column 'Var') which allowed to obtain these results are also shown. The values in table 4.1 are computed as the mean over the different folds in the respective CV. Figure 4.2 shows, for every configuration, the evolution of the percentage of correct examples in the training and test sets with respect to the number of eliminated variables in the SBS procedure.

As expected, the addition of irrelevant features affects very negatively to the performance of sinusoidal MLPs for this problem, even if we try to control the

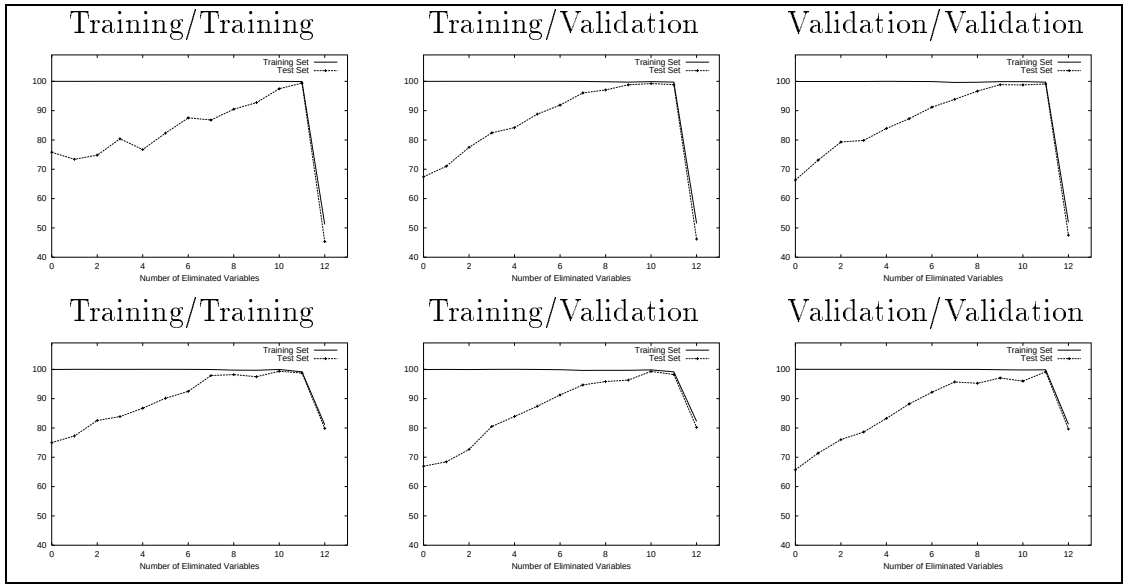

Figure 4.2: Percentage of correct examples for the *Augmented XOR* data set in the training and test sets with respect to the number of eliminated variables in the SBS procedure for MLPs with retraining (top) and without it (bottom). These values are computed as the mean over the different folds in the respective CV.

overfitting. The information needed to learn the problem is present, but the system is not able to use it in a proper way. The reason for this fact may be the relatively small number of examples in the data set, that did not allow to filter this kind of features. As far as variables are eliminated, performance improves. However, many variables must be eliminated to obtain good performance. We also observed this behavior in several preliminary experiments with the hyperbolic tangent function.

If we look at the final selected variables (see table 4.1), we can see that there is a strong coincidence among the different configurations, which select the original variables or some equivalent ones. Results in table 4.1 seem to indicate that there is no difference among the different configurations[3]. But, as we will see in the rest of the chapter, this behavior is not the general rule. However, it shows that there exist problems where any of these configurations can be used to eliminate irrelevant variables.

4.5.2 The *Augmented Two Spirals* Data Set

Similar to the *Augmented XOR* data set, we performed an experimental comparison of the aforementioned configurations with an augmented version of the *Two Spirals*

[3]The small differences between models which select the same variables could be explained by the initial random weights and the different number of points in the respective training sets.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
Yes	Training/Validation	99.89%	0.01	3
Yes	Validation/Validation	99.66%	0.02	3
Yes	Training/Training	93.76%	0.19	4
No	Training/Validation	92.30%	0.25	6
No	Training/Training	92.10%	0.24	7
No	Validation/Validation	87.10%	0.39	6

Table 4.2: Test set results for the *Augmented Two Spirals* data set after the SBS procedure for MLPs and different combinations of retraining/stopping criterion/loss measurement data set.

data set, where we artificially added 13 irrelevant features to the two original variables. Some of them were noisy. The details of the added features can be found in appendix A.

4.5.2.1 Methodology

We joined the three original data sets (training, validation and test) into a single data set, in order to perform the experiments with stratified Cross-Validation.

These are the specific parameters for this data set in the general methodology described in section 4.4:

1. In the first step, the selected number of hidden units was 40, with an initial range of weights of 5.0 and $1,500$ epochs.

2. In the second step, the number of epochs to increase was 100, when applied.

4.5.2.2 Results

The best results among the different learning combinations for every configuration are shown in table 4.2 (column 'Test') as the average percentage of correctly classified patterns on the respective test sets in the following trained networks:

1. For the Training/Training configuration, the networks with minimum squared test set error among the networks with minimum training set error after every variable is permanently eliminated.

2. For the Training/Validation and Validation/Validation configurations, the networks with minimum squared test set error among the networks with minimum validation set error after every variable is permanently eliminated.

The mean squared error on the test set (column 'Mse') and the number of variables (column 'NVar') which allowed to obtain these results are also shown. Values

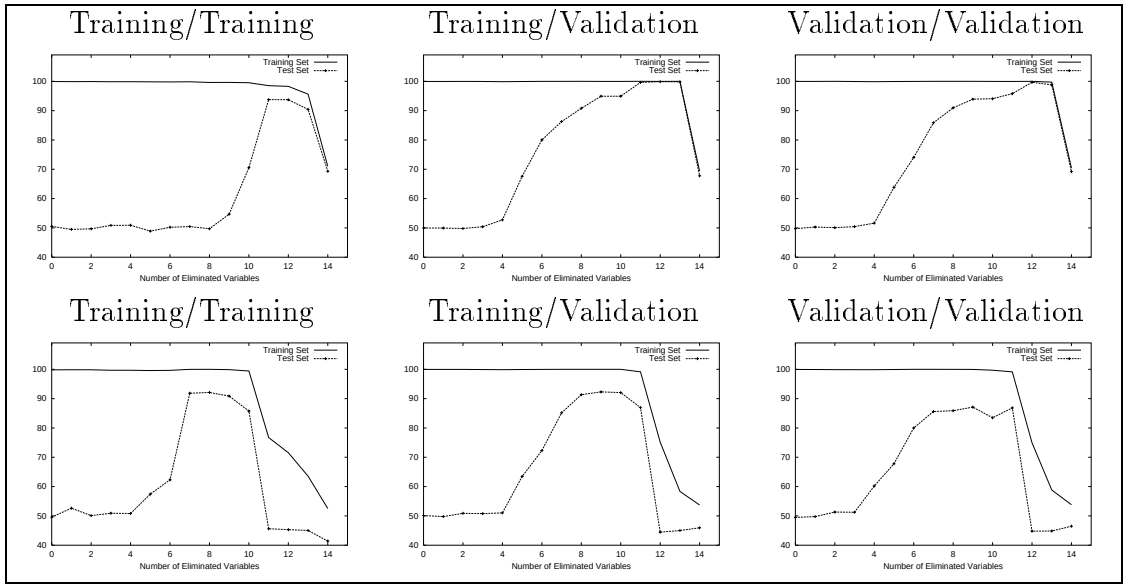

Figure 4.3: Percentage of correct examples for the *Augmented Two Spirals* data set in the training and test sets with respect to the number of eliminated variables in the SBS procedure for MLPs with retraining (top) and without it (bottom). These values are computed as the mean over the different folds in the respective CV.

in table 4.2 are computed as the mean over the different folds in the respective CV. Figure 4.3 shows, for every configuration, the evolution of the percentage of correct examples in the training and test sets with respect to the number of eliminated variables in the SBS procedure.

From these results, we can conclude that:

1. Similar to the *Augmented XOR* data set, the addition of irrelevant features affects very negatively to the performance of sinusoidal MLPs for this problem, even if we try to control the overfitting (see figure 4.3). As far as variables are eliminated, performance improves, although many variables must be eliminated to obtain good performance. We observed the same behavior in several preliminary experiments with the hyperbolic tangent function.

2. Confirming our intuition, retraining the network with every feature temporarily removed previous to computing the saliency has a positive effect on the SBS procedure for MLPs, both for the number of selected features and the overall performance.

3. When retraining is performed:

   (a) It seems that the SBS procedure uses the validation set to construct a better subset of variables than if only the training set is used, although

Retrain	Stop. Crit./Loss Meas.	Order of Variable Elimination
Yes	Training/Validation	4 11 13 14 15 12 10 9 8 3 5 2 ▶ 1 6 7
Yes	Validation/Validation	9 7 15 13 14 12 11 10 8 4 3 5 ▶ 6 1 2
Yes	Training/Training	1 10 12 6 9 8 7 11 14 13 15 ▶ 4 5 3 2
No	Training/Validation	9 1 13 14 15 11 12 10 5 ▶ 2 6 7 4 8 3
No	Training/Training	5 10 14 11 15 12 13 1 ▶ 6 2 7 8 9 4 3
No	Validation/Validation	14 15 5 13 11 12 10 2 1 ▶ 6 9 7 4 8 3

Table 4.3: Order of eliminated variables for the *Augmented Two Spirals* data set after the SBS procedure for MLPs and different combinations of retraining/stopping criterion/loss measurement data set. Left variables were the first eliminated ones. Variables on the right are the most important variables considered by every configuration. The symbol ▶ indicates the point from which the variables have been selected.

there is no significant difference with regard to the stopping criterion (note that the Training/Validation configuration obtains similar results to the Validation/Validation one).

(b) Surprisingly, the evolution of the training set error is better using a validation set than without it (see figure 4.3).

4.5.3 Analysis

The order of variable elimination for every configuration can be seen in table 4.3. Ideally, variables x_{13}, x_{14} and x_{15} should never be considered as important, and variables from x_1 to x_7 should be more important than variables from x_8 to x_{12} (see appendix A).

The claim about x_{13}, x_{14} and x_{15} is satisfied by all the configurations. However, when retraining is not performed, we can see that several noisy variables (x_8 or x_9, for example) are considered as important. It does not happen when retraining is performed, and it was not observed for the *Augmented XOR* data set (see table 4.1). When retraining is performed, the redundancy of the variables allows to obtain good generalization results with feature subsets different from (although equivalent to) the two original variables. The Training/Training configuration seems to select, however, a wrong noise free subset.

The observed results can be explained by looking at the behavior of every configuration during the SBS procedure. We observed that:

1. When retraining is not performed, the temporary elimination of any variable leads to large errors in the training set when compared to the training error with the whole set of features. The third column in table 4.4 shows this fact for the first step of the SBS procedure in the *Augmented Two Spirals* data

Variable	*Augmented XOR*	*Augmented Two Spirals*
x_1	84.55%	60.56%
x_2	83.38%	59.58%
x_3	84.40%	51.34%
x_4	82.70%	52.39%
x_5	-	66.27%
x_6	90.60%	59.08%
x_7	88.97%	57.59%
x_8	98.30%	54.17%
x_9	97.90%	52.95%
x_{10}	-	65.12%
x_{11}	96.25%	61.40%
x_{12}	94.73%	59.86%
x_{13}	94.53%	54.94%
x_{14}	97.92%	61.87%
x_{15}	96.32%	61.23%
None	100.0%	100.0%

Table 4.4: Percentage of correct examples in the training set after the temporary elimination of every variable in the first step of the SBS procedure when retraining is not performed. The first column indicates the temporarily eliminated variable. The second and third columns are the percentages for the *Augmented XOR* and *Augmented Two Spirals* data sets respectively. Variables with large percentages can be interpreted as variables that are not very important in the obtained solution. Note that the whole set of features allows to fit perfectly the training set. Although the selection was made using the sum-of-squares error, we show the percentages of correct examples since they are easier to interpret.

set. This happens because the minimum obtained after the training process uses all the variables in a significant way. Therefore, the elimination of a feature without recalculating this minimum substantially modifies the learned function. In addition, it does not seem that noise free variables (from x_1 to x_7) are more used than the rest in order to learn the data set. The same behavior is observed in several subsequent steps, so that when retraining is not performed the permanent elimination of a variable is decided, for this data set, in a quite random way. For the *Augmented XOR* problem, in contrast, a line can be drawn which clearly separates variables x_1 to x_7 from the rest (see the second column in table 4.4). The reason of that different behavior is not clear, but it may be related to the difference between the percentages of correct examples in the test set during the first steps of the SBS procedure: the *Augmented Two Spirals* suffers a much more strong performance degradation than the *Augmented XOR* problem (see figures 4.2 and 4.3), indicating that irrelevant

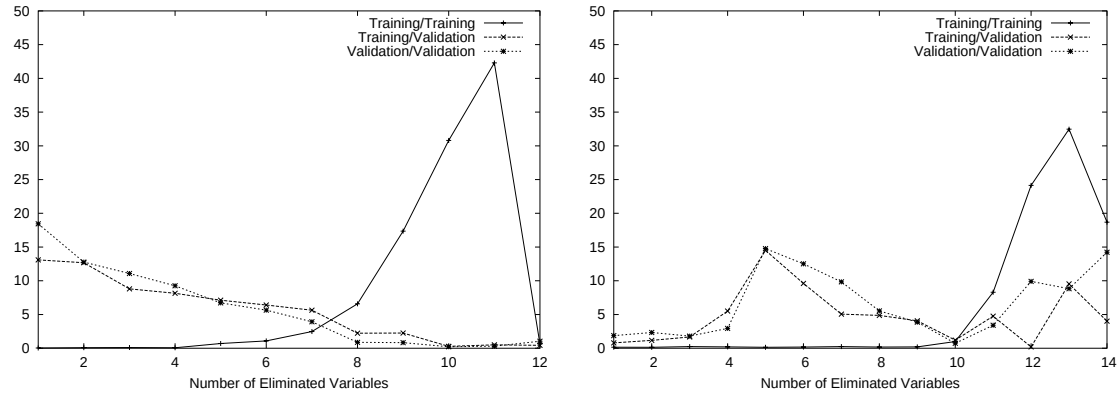

Figure 4.4: Differences, at every step of the SBS procedure, between the maximum and minimum percentages of correct examples in the respective loss measurement data sets after retraining with every variable temporarily removed. Left plot shows the results for the *Augmented XOR* data set. Right plot shows the results for the *Augmented Two Spirals* data set. As an example, suppose that (N, P) is plotted. It means that in the process of elimination of the Nth variable, and after retraining the network without every remaining feature temporarily removed, the maximum difference among the percentages of correct examples in the respective loss measurement data sets was P. Very small percentages indicate that there is hardly any difference among the variables. Although the selection was made using the sum-of-squares error, we show the percentages of correct examples since they are easier to interpret.

variables are not equally important in the respective obtained solutions.

2. When retraining is performed,

   (a) In the Training/Training configuration, we observed that, for the *Augmented Two Spirals* data set, the training set is learned almost perfectly independently on the temporarily eliminated variable, so that no significant difference can be stated among the variables. In our experiments, it was observed in the elimination of the first 9 or 10 variables in all the performed runs. The right plot in figure 4.4 shows clearly this behavior. Therefore, and similar to the case of absence of retraining, the permanent elimination of a variable is decided in a quite random way during (too) many steps. For the *Augmented XOR* data set, significant differences among the variables were observed from earlier steps of the procedure (see left plot in figure 4.4).

   (b) For the Training/Validation and Validation/Validation configurations, in contrast, the variables are much more clearly differentiated from the beginning of the SBS procedure (see figure 4.4). For the *Augmented Two*

Spirals data set, note that the minimum observed at point 10 in the X axis of the right plot of figure 4.4 corresponds to the point where all the noisy variables have already been eliminated (see table 4.3). The criterion to eliminate permanently a variable does not seem random. Although it does not mean that the order of eliminated variables is ideal (see table 4.3), it is reasonably acceptable, taking into account that there exist redundant variables.

A common point of the configurations that do not obtain satisfactory results is the fact that they consider variables x_3 and x_4 (the squared of the original variables) as important. Looking at the results, these variables do not seem the most promising ones for this problem: variables x_3 and x_4 allow, together with other ones, to fit the training set, but those feature subsets are not good for generalization purposes. Note that, when all the variables are present, variables x_3 and x_4 are significantly used to fit the training set in both problems (see table 4.4). Therefore, a similar behavior with respect to these variables could be expected for both data sets, but it was not observed. Although it is not clear the reason of this different behavior, we think that the number of examples in the data set can be a key issue. The *XOR* problem is easier than the *Two Spirals* one. The number of examples in the respective data sets could be enough for the *Augmented XOR* but not for the *Augmented Two Spirals* problem in order to select a proper subset of variables.

These observations allow to explain the results for the *Augmented Two Spirals* data set showed in table 4.2 and the differences with the results for the *Augmented XOR* data set in table 4.1.

4.6 Experiments on Benchmark Data Sets

In this section, the experiments on several benchmark data sets with the SBS procedure for MLPs described in figure 4.1 are shown. A brief description of the data sets used in the experiments can be found in appendix A.

The obtained results show that, although in a different scale, a similar behavior to that of the *Augmented Two Spirals* data set is observed.

4.6.1 Methodology

These are the specific parameters for these data sets in the general methodology described in section 4.4:

1. All experiments were performed for "linear" and non-linear MLPs, trained with standard BP. Linear MLPs had no hidden layers and hyperbolic tangents

Data Set	NHid	WRange	NEpochs	NEpochs+
Pima Indians Diabetes	15	[-0.05,0.05]	2,500	250
Wisconsin Breast Cancer	2	[-0.05,0.05]	1,500	150
Hepatitis	20	[-1.5,1.5]	300	20
Ionosphere	10	[-0.5,0.5]	300	20
Sonar	35	[-1.0,1.0]	125	10

Table 4.5: Learning parameters for the benchmark data sets used in our experiments. The column 'NHid' shows the number of hidden units, 'WRange' the initial range of random weights and the column 'NEpochs' the number of epochs. The column 'NEpochs+' indicates, when applied, the increment in epochs after the elimination of every variable.

in the output layer[4]. Non-linear MLPs had one hidden layer of sinusoidal units, as previously explained. The motivation for testing linear MLPs was to investigate whether it could be possible that after the elimination of several variables, the remaining data could be reasonably fitted by a linear model or not, and to study its performance in that case.

2. Table 4.5 shows the selected number of hidden units, range of weights and number of epochs for every data set in the first step. The increment in epochs after the elimination of every variable for the second step, when applied, is also shown.

4.6.2 Results

The best results among the different learning combinations for every configuration are shown in tables 4.6 to 4.15 (column 'Test') as the average percentage of correctly classified patterns on the respective test sets in the following trained networks:

1. For the Training/Training configuration, the networks with minimum squared test set error among the networks with minimum training set error after every variable is permanently eliminated.

2. For the Training/Validation and Validation/Validation configurations, the networks with minimum squared test set error among the networks with minimum validation set error after every variable is permanently eliminated.

The mean squared error on the test set (column 'Mse') and the number of variables (column 'NVar') which allowed to obtain these results are also shown. Results

[4]MLPs with no hidden layers and hyperbolic tangents in the output layer are not, strictly speaking, linear models. They are usually referred to as Single-Layer Perceptrons. Since the decision boundary for classification problems is a hyperplane, we will consider them as linear.

for linear models are in tables 4.6 to 4.10, whereas the results for non-linear ones are shown in tables 4.11 to 4.15. Figures 4.5 to 4.7 show, for every configuration, the evolution of the percentage of correct examples in the training and test sets with respect to the number of eliminated variables in the SBS procedure for the most interesting non-linear problems, namely *Hepatitis*, *Ionosphere* and *Sonar* (see below). The values in these tables and figures are computed as the mean over the different folds in the respective CV.

Several conclusions from these experiments can be summarized as follows:

1. Similar to the experiments for *SAOCIF* (see section 3.7), we can observe that there exist data sets, such as the *Pima Indians Diabetes* and *Wisconsin Breast Cancer*, where the results are very similar among all the parameter configurations. It is very difficult to obtain good conclusions from these experiments. Note that the learning parameters for these two data sets are quite different to the rest (see table 4.5).

2. Linear models have a different behavior from that of non-linear ones:

   (a) Linear models obtain consistently worse results than non-linear ones and the best resulting linear models have more variables, specially when the number of original variables is larger (see the results for the *Ionosphere* and *Sonar* data sets). For the *Hepatitis* data set, similar results are obtained for linear models with the same variables than non-linear ones (see below). It is worth noting that the *Ionosphere* and *Sonar* data sets do not have missing values, in contrast to the *Hepatitis* data set. It may happen that the missing information does not allow to observe differences between linear and non-linear models for this data set (for example, if the missing values are present in important variables).

   (b) In addition, linear models do not seem to take profit from measuring the loss function in a validation set (the best results are usually obtained with the Training/Training configuration, although the networks present overtraining), in contrast to non-linear models (see below).

   (c) Overall, linear models have no clear tendency. A deeper study should be necessary in order to explain their behavior.

3. Regarding non-linear models for the most interesting problems, namely *Hepatitis*, *Ionosphere* and *Sonar*, we can see that:

   (a) The best results are always obtained retraining the network with every feature temporarily removed previous to computing the saliency, as expected. As previously said, this is a non-common strategy (to the best of our knowledge, the only models that retrain the network at every step

with every feature temporarily removed/added are those in [Steppe et al. 1996; Onnia et al. 2001], and none of them is a pure SBS procedure). Therefore, the usefulness of the SBS procedure for MLPs retraining the network with every feature temporarily removed previous to computing the saliency is confirmed. We consider that the increase in the computational cost associated with this scheme is rewarded with a significant performance improvement, and deserves further research.

(b) With respect to the stopping criterion/loss measurement data set, it seems that the SBS procedure takes profit from measuring the loss function in a validation set, different to linear models, although it is not clear which is the best stopping criterion. For the *Sonar* problem the Validation/Validation strategy seems to work best. For the *Ionosphere* problem, in contrast, the Training/Validation strategy selects a better subset of variables[5]. For the *Hepatitis* problem both configurations can be considered as equivalent[6]. The goodness of the Validation/Validation configuration can be intuitively explained. However, forcing overtraining (in our scheme, training until a minimum of the training set) may help to improve performance, whenever a validation set is used to measure the goodness of a variable subset (the Training/Validation configuration). This is a non-intuitive result. Therefore, forcing the system to use all the available features as much as possible helps to detect irrelevant variables properly.

(c) Although in a different scale, a similar behavior to that of the *Augmented XOR* and *Augmented Two Spirals* data sets is observed in figures 4.5 to 4.7. First, note that the training set can be fitted with a much smaller subset of features than the original one. There exists a point where the remaining features do not allow to fit the training set. Regarding the test set, performance improves until a subset of features remains where the elimination of further variables results in performance degradation. This behavior seems to indicate that there exist irrelevant variables that the SBS procedure for MLPs has detected and eliminated. However, the differences among the results of the different configurations are clearly more similar to *Augmented Two Spirals* data set than to the *Augmented XOR* one. The reason for these differences could be, as in the *Augmented Two Spirals* data set, to the existence of several variables that allow to

[5]For the *Ionosphere* problem, the variables selected by the Training/Validation configuration were $\{x_2, x_4, x_5, x_7, x_{20}\}$. The rest of configurations with 5 features selected $\{x_2, x_4, x_7, x_{20}, x_{26}\}$.

[6]For the *Hepatitis* problem, the variables selected by the Training/Validation and Validation/Validation configuration were $\{x_2, x_5, x_{18}\}$. The rest of configurations with 3 features selected other subsets ($\{x_{13}, x_{15}, x_{18}\}$, $\{x_8, x_{13}, x_{18}\}$ or $\{x_{13}, x_{17}, x_{18}\}$).

fit the training set but are not good for generalization purposes. The number of available examples would not allow to filter these variables in some cases. In this sense, we think that the overall results could be further improved if more examples could be used.

(d) There exists an important improvement in the overall results with respect to learning with the whole set of variables (see table A.2 in appendix A and the results for *SAOCIF* in section 3.7 for MLP architectures, for example). Results in these experiments are also quite good when compared with other existing FS wrappers in the literature (see table A.3 in appendix A). Note that the number of selected variables for the best models is smaller than those in table A.3. The good results obtained with the previously described criteria are mainly due, in our opinion, to a proper detection of irrelevant variables.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
No	Training/Training	77.62%	0.63	7
Yes	Training/Training	77.41%	0.62	7
Yes	Validation/Validation	77.07%	0.62	5
No	Training/Validation	77.07%	0.63	8
Yes	Training/Validation	76.95%	0.63	7
No	Validation/Validation	76.95%	0.63	8

Table 4.6: Test set results for the *Pima Indians Diabetes* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
No	Training/Validation	96.89%	0.10	7
Yes	Validation/Validation	96.76%	0.10	7
Yes	Training/Validation	96.72%	0.10	7
No	Validation/Validation	96.64%	0.11	7
No	Training/Training	96.63%	0.12	5
Yes	Training/Training	96.52%	0.11	4

Table 4.7: Test set results for the *Wisconsin Breast Cancer* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
Yes	Training/Training	93.42%	0.25	3
Yes	Validation/Validation	93.10%	0.26	3
Yes	Training/Validation	92.71%	0.26	6
No	Training/Training	90.06%	0.36	1
No	Validation/Validation	89.94%	0.32	10
No	Training/Validation	88.90%	0.38	1

Table 4.8: Test set results for the *Hepatitis* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
Yes	Training/Training	92.06%	0.27	14
No	Training/Training	90.74%	0.33	17
Yes	Training/Validation	90.16%	0.33	16
Yes	Validation/Validation	90.04%	0.33	13
No	Validation/Validation	89.37%	0.36	12
No	Training/Validation	89.20%	0.37	12

Table 4.9: Test set results for the *Ionosphere* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
No	Training/Training	85.46%	0.51	11
Yes	Training/Training	85.07%	0.54	22
Yes	Training/Validation	83.88%	0.51	20
Yes	Validation/Validation	82.51%	0.58	18
No	Validation/Validation	80.93%	0.60	7
No	Training/Validation	79.37%	0.65	20

Table 4.10: Test set results for the *Sonar* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
No	Training/Training	77.91%	0.63	5
Yes	Validation/Validation	77.75%	0.62	4
No	Validation/Validation	77.70%	0.62	4
Yes	Training/Training	77.65%	0.62	5
No	Training/Validation	77.63%	0.62	4
Yes	Training/Validation	77.43%	0.62	4

Table 4.11: Test set results for the *Pima Indians Diabetes* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with non-linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
Yes	Validation/Validation	96.87%	0.11	6
No	Training/Training	96.86%	0.10	7
No	Validation/Validation	96.70%	0.11	8
Yes	Training/Training	96.69%	0.11	5
Yes	Training/Validation	96.68%	0.11	8
No	Training/Validation	96.64%	0.11	6

Table 4.12: Test set results for the *Wisconsin Breast Cancer* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with non-linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
Yes	Training/Validation	93.90%	0.24	3
Yes	Validation/Validation	93.77%	0.25	3
Yes	Training/Training	92.26%	0.25	3
No	Training/Training	92.13%	0.26	3
No	Validation/Validation	88.97%	0.40	1
No	Training/Validation	88.26%	0.36	3

Table 4.13: Test set results for the *Hepatitis* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with non-linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
Yes	Training/Validation	93.61%	0.22	5
No	Validation/Validation	92.77%	0.24	5
Yes	Validation/Validation	92.73%	0.24	5
No	Training/Validation	92.57%	0.24	5
No	Training/Training	92.40%	0.25	5
Yes	Training/Training	90.51%	0.33	3

Table 4.14: Test set results for the *Ionosphere* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with non-linear networks.

Retrain	Stopping Criterion/Loss Measurement	Test	Mse	NVar
Yes	Validation/Validation	89.73%	0.33	14
Yes	Training/Training	88.59%	0.37	7
Yes	Training/Validation	87.95%	0.36	11
No	Training/Training	87.41%	0.40	57
No	Training/Validation	85.02%	0.46	46
No	Validation/Validation	84.49%	0.47	50

Table 4.15: Test set results for the *Sonar* data set after the SBS procedure and different combinations of retraining/stopping criterion/loss measurement data set with non-linear networks.

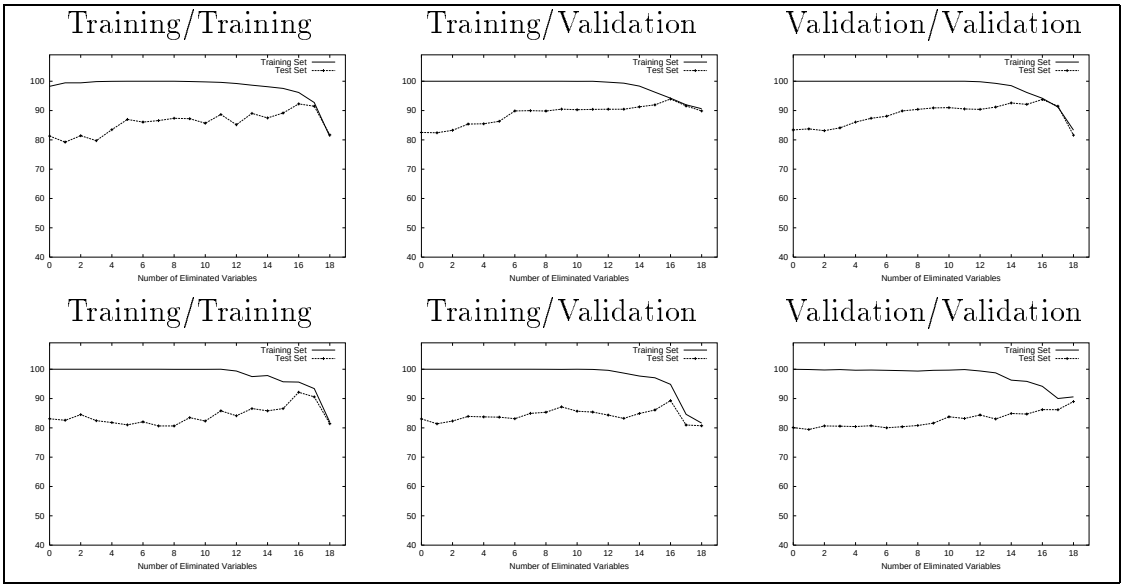

Figure 4.5: Percentage of correct examples for the *Hepatitis* data set in the training and test sets (non-linear models) with respect to the number of eliminated variables in the SBS procedure for MLPs with retraining (top) and without it (bottom). These values are computed as the mean over the different folds in the respective CV.

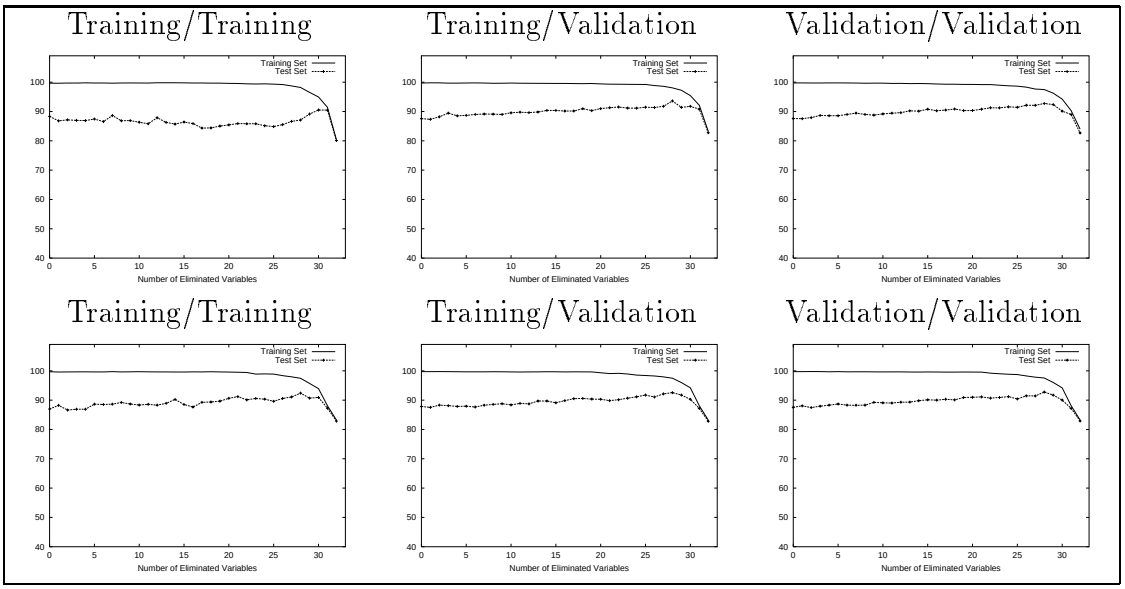

Figure 4.6: Percentage of correct examples for the *Ionosphere* data set in the training and test sets (non-linear models) with respect to the number of eliminated variables in the SBS for MLPs procedure with retraining (top) and without it (bottom). These values are computed as the mean over the different folds in the respective CV.

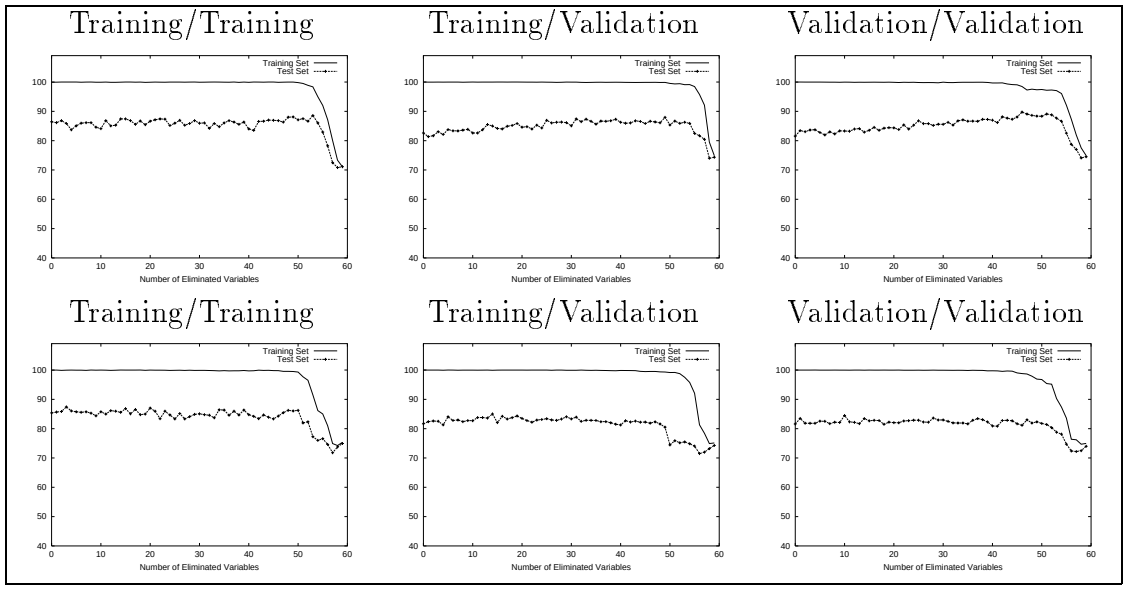

Figure 4.7: Percentage of correct examples for the *Sonar* data set in the training and test sets (non-linear models) with respect to the number of eliminated variables in the SBS procedure for MLPs with retraining (top) and without it (bottom). These values are computed as the mean over the different folds in the respective CV.

Chapter 5

WQL: Weighting the Quadratic Loss Function to Maximize the Margin

In this chapter we describe a modification of the quadratic loss function for classification problems inspired in Support Vector Machines (SVMs) and the AdaBoost algorithm. In the linearly separable case, the hyperplane that maximizes the normalized margin also minimizes asymptotically the loss function proposed. The *hardness* of the resulting solution can be controlled, as in SVMs, so that this model can also be used with Feed-forward Neural Networks (FNNs) for the non-linearly separable case. The modification proposed is tested with artificial and real-world data sets.

5.1 Introduction

As previously said (see chapter 2), FNNs and SVMs are two alternative Supervised Machine Learning (SML) frameworks for approaching classification and regression problems developed from very different starting points of view. The minimization of the sum-of-squares error and the maximization of the margin performed by FNNs and SVMs, respectively, lead to a very different inductive bias with very interesting properties [Bishop 1995a; Vapnik 1998a].

We will consider the classification task given by a data set D as in (2.1), where each instance x_i belongs to the input space $\mathbb{R}^I$, $y_i \in \{-1,+1\}^C$ and C is the number of classes[1]. Looking at the similarities and differences between FNNs and SVMs, it can be observed that the main difference between the sum-of-squares minimization problem of an FNN and the margin maximization problem of an (1-norm soft margin) SVM lies in the constraints related to the objective function. Since these

[1]For 2-class problems, usually $y_i \in \{-1,+1\}$.

constraints are responsible for the existence of the support vectors, their behavior will give the key to propose a new learning model for FNNs related to margin maximization.

A weighting of the sum-of-squares error function is proposed, that will be referred to as *Weighted Quadratic Loss* (*WQL*) function. During the learning process, the *WQL* function reinforces the contribution of misclassified points to the total error, and reduces the contribution of well classified ones. The reinforcement or reduction of every point depends on its margin, and it is inspired in the AdaBoost algorithm. The proposed error weighting forces the training procedure to pay more attention to the points with a smaller margin. In terms of the Bias/Variance decomposition, variance tries to be controlled by not attempting to overfit the points that are already well classified.

In the linearly separable case, the hyperplane that maximizes the normalized margin also minimizes asymptotically the *WQL* function proposed. The hardness of the resulting solution can be controlled, as in SVMs, so that this model can be used for the non-linearly separable case as well.

The classical FNN architecture of the new proposed scheme presents some advantages. On the one hand, the final solution is neither restricted to have an architecture with so many hidden units as points in the data set (or support vectors) nor to use kernel functions. The frequencies are not restricted to be a subset of the data set. On the other hand, it allows to deal with multiclass and multilabel problems in a natural way as FNNs usually do. This is a non-trivial problem for SVMs, since they are initially designed for binary classification problems.

Both theoretic and experimental results are shown confirming these claims. Several experiments have been conducted on artificial and two real-world problems from the Natural Language Processing domain, namely Word Sense Disambiguation and Text Categorization. Several comparisons among the new proposed model and SVMs have been made in order to see the agreement of the predictions made by the respective classifiers. The obtained results seem to confirm the hypothesis made about the new proposed scheme.

In section 5.2, the comparison between FNNs and SVMs is performed. The weighting of the sum-of-squares error function is described in section 5.3. Sections 5.4, 5.5 and 5.6 are devoted to the experimental work.

5.2 A Comparison Between FNNs and SVMs

In this section, a comparison between FNNs and (1-norm soft margin) SVMs is performed. After comparing the respective output and cost functions, it will be clear that the main difference lies in the presence or absence of constraints in the optimization problem.

5.2.1 Comparing the Output Functions

As pointed out elsewhere (see, for example, [Vapnik 1998b]), the output function f^o_{SVM} (2.28) of an SVM can be expressed with a fully connected FNN with one output unit and one hidden layer of units f^o_{FNN} (2.8) with the following identifications:

1. Number of hidden units: L (the number of examples in D)
2. Coefficients: $\lambda_k = y_k \alpha_k$
3. Frequencies: $\omega_k = x_k$
4. Biases: every b_k vanishes, and $b_0 = b$
5. Activation functions:
   (a) Hidden layer: $\varphi_k(x_k, x, b_k) = K(x_k, x)$
   (b) Output layer: φ_0 linear

As in SVMs, the only parameters to be learned in such an FNN would be the coefficients and the biases. So, the main differences between FNNs and SVMs rely on both the cost function to be optimized and the constraints. Specific learning algorithms are a consequence of the optimization problem to be solved.

5.2.2 Comparing the Cost Functions

Defining

$$K_L = \begin{pmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots & K(x_1, x_L) \\ K(x_2, x_1) & K(x_2, x_2) & \cdots & K(x_2, x_L) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_L, x_1) & K(x_L, x_2) & \cdots & K(x_L, x_L) \end{pmatrix}$$

$y = (y_1, \ldots, y_L)^T$, $y\alpha = (y_1\alpha_1, \ldots, y_L\alpha_L)^T$, neglecting the bias term b, and considering the identifications stated in section 5.2.1 we have that $f^o_{FNN}(x_j)$, which is equal to $f^o_{SVM}(x_j)$, is the j-th row of the vector $y\alpha^T \cdot K_L$. Therefore, we can express the

respective cost functions (2.9) and (2.29) as[2]:

$$\begin{aligned} E(D) &= \sum_{i=1}^{L} f^o_{FNN}(x_i)^2 - 2\sum_{i=1}^{L} y_i f^o_{FNN}(x_i) + \sum_{i=1}^{L} y_i^2 \\ &= y\alpha^T \cdot K_L \cdot K_L \cdot y\alpha - 2 \cdot y\alpha^T \cdot K_L \cdot y + L \\ M(D) &= -\frac{1}{2}\, y\alpha^T \cdot K_L \cdot y\alpha + y\alpha^T \cdot y \end{aligned}$$

Regardless of their apparent similarity, we may wonder whether there is any direct relationship between the minima of $E(D)$ and the maxima of $M(D)$ (or equivalently, the minima of $-M(D)$). The next result partially answers this question.

Proposition 2. Consider the identifications in section 5.2.1 without the bias term b. If K_L is non-singular, then the respective cost functions $E(D)$ and $-M(D)$ attain their unique minimum (without constraints) at the same point

$$(y_1\alpha_1^*, ..., y_L\alpha_L^*)^T = K_L^{-1} \cdot y.$$

Proof. As $E(D)$ and $-M(D)$ are convex functions, a necessary and sufficient condition for $y\alpha^*$ to be a global minimum is that their derivative with respect to $y\alpha$ vanishes:

$$\frac{\partial E(D)}{\partial y\alpha} = 2 \cdot K_L \cdot K_L \cdot y\alpha - 2 \cdot K_L \cdot y = 0$$

$$-\frac{\partial M(D)}{\partial y\alpha} = K_L \cdot y\alpha - y = 0$$

Since K_L is non-singular, both equations have the same solution.

□

If K_L is singular, there will be more than one point where the optimum value is attained, but all of them are equivalent. In addition, K_L has rows which are linearly dependent among them. This fact indicates that the information provided (via the inner product) by a point in the data set is redundant, since it is a linear combination of the information provided by other points. If the bias b is fixed *a priori*, the same result holds.

[2]For the sake of simplicity, in this chapter we will consider the empirical risk as $E(D) = \sum_{i=1}^{L} (y_i - f^o_{FNN}(x_i))^2$, neglecting the $\frac{1}{L}$ term.

5.2.3 Towards a common model

The optima of $E(D)$ and $M(D)$ can be very different depending on the absence or presence of the constraints. Therefore, it seems that the main difference between the respective optimization problems for FNNs and (1-norm soft margin) SVMs lies in the constraints.

At this point, at least two questions can be posed:

1. Which constraints could be added to the sum-of-squares error to imitate an SVM?

2. How can the sum-of-squares error be modified to simulate the behavior of an SVM?

To answer the first question it seems necessary to identify the arguments of the constraints. They could be the coefficients, but it is not clear, since they play a different role in FNNs than in SVMs. Another possibility could be to define constraints involving the outputs of the network in the points of the data set, but the resulting optimization problem would not have linear constraints. The next section gives an answer to the second question above.

5.3 An FNN that Maximizes the Margin

In this section we explore the effect of the constraints in the solution obtained by the SVMs approach. Since these constraints are responsible for the existence of the support vectors, their behavior will give the key to propose a weighting of the sum-of-squares error function, inspired in the AdaBoost algorithm.

5.3.1 Contribution of Every Point to the Cost Function

The existence of linear constraints in the optimization problem to be solved in SVMs has a very important consequence: only some of the α_i will be different from zero. These coefficients are associated with the so called support vectors. Thus, the remaining vectors can be omitted, both for optimizing $M(D)$ and computing the output (2.28). The problem is that we do not know them in advance. In the linearly separable case (either in the input space or in the feature space), with hard margin solutions, support vectors have margin 1 (i.e., $f_{SVM}(x_i) = y_i$), while the remaining points (that will be referred to as *super-classified* points) have a margin strictly greater than 1. Super-classified points can be omitted.

In contrast, for FNNs minimizing the sum-of-squares error function, every point makes a certain contribution to the total error. The greater is the squared error, the greater will be the contribution, independently on whether the point is well or

wrongly classified. With linear output units, there may be points (very) well classified with a (very) large squared error. Super-classified points are a clear example of this type. Sigmoidal output units can help to solve this problem, but they can also create new ones (in the linearly separable case, for example, the solution is not bounded). An alternative idea to sigmoidal output units could be to reduce the contribution of super-classified points and reinforce those of misclassified points, as explained in the next section.

5.3.2 Weighting the Contribution

Unfortunately, we do not know in advance which points will be finally super-classified or misclassified. But during the FNN learning process it is possible to treat every point in a different way depending on its error (or, equivalently, its margin). In order to simulate the behavior of an SVM, the learning process could be guided by the following heuristics:

1. Any well classified point contributes less to the error than any misclassified point.

2. Among well classified points, the contribution is larger for smaller errors in absolute value (or equivalently, smaller margins).

3. Among misclassified points, the contribution is larger for larger errors in absolute value (or equivalently, smaller margins).

These guidelines reinforce the contribution of misclassified points and reduces the contribution of well classified ones. Therefore, variance tries to be controlled by not attempting to overfit the points that are already well classified. As it can be seen, this is exactly the same idea as the distribution (2.30) for the AdaBoost algorithm. Similarly, the contribution of every point to the error can be modified simply by weighting it individually as a function of the margin with respect to the output function $f^o_{FNN}(x)$. We will call this error function a *Weighted Quadratic Loss* (*WQL*) function. In order to allow more flexibility to the model, two parameters $\alpha^+, \alpha^- \geqslant 0$ can be introduced into the weighting function as follows:

$$W(x_i, y_i, \alpha^+, \alpha^-) = \begin{cases} e^{-|\mathrm{mrg}|^{\alpha^+}} & \text{if } \mathrm{mrg} \geqslant 0 \\ e^{+|\mathrm{mrg}|^{\alpha^-}} & \text{if } \mathrm{mrg} < 0 \text{ and } \alpha^- \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.1}$$

where the margin $\mathrm{mrg} = mrg(x_i, y_i, f^o_{FNN}) = y_i f^o_{FNN}(x_i)$. Figure 5.1 shows several weighting functions corresponding to different values of α^+. As it can be observed, for very large values of α^+ it is very similar to a heavy-side function.

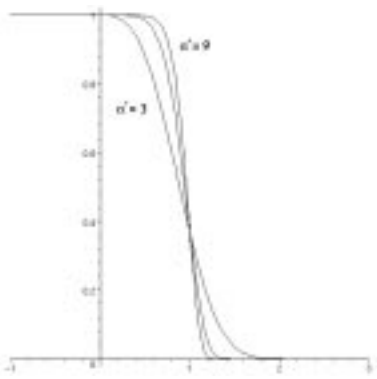

Figure 5.1: The weighting function (5.1) depending on α^+.

There are (at least) two different ways of obtaining the behavior previously described:

1. Weighting the sum-of-squares error:

$$E_p = (y_i - f^o_{FNN}(x_i))^2 \cdot W(x_i, y_i, \alpha^+, \alpha^-) \tag{5.2}$$

2. Weighting the sum-of-squares error derivative (when the derivative is involved in the learning process):

$$E_p \text{ such that } -\frac{\partial E_p}{\partial f^o_{FNN}} = (y_i - f^o_{FNN}(x_i)) \cdot W(x_i, y_i, \alpha^+, \alpha^-) \tag{5.3}$$

where $E_p = E_p(f^o_{FNN}(x_i), y_i, \alpha^+, \alpha^-)$. Graphically, the right branch of the squared error parabola is bended to a horizontal asymptote, as it can be seen in figure 5.2. Weighting the sum-of-squares error derivative also implies a kind of weighting the sum-of-squares error, although in a slightly different way[3].

The following result justifies that the previously suggested *WQL* functions (5.2) and (5.3) are well founded. In addition, it allows to construct new error functions with the same underlying ideas.

Theorem 2. Let $f \in \mathbb{R}$, $y \in \{-1, +1\}$, $\alpha^+, \alpha^- \geqslant 0$ and $E_p(f, y, \alpha^+, \alpha^-)$ an error function satisfying:

1. There exists a constant A such that for every f, y, α^+, α^- we have

$$E_p(f, y, \alpha^+, \alpha^-) \geqslant A.$$

[3]Note that (5.3) does not derive from (5.2). Abusing of notation, we indicate that the weighting can be made either at the error level or at the error derivative level.

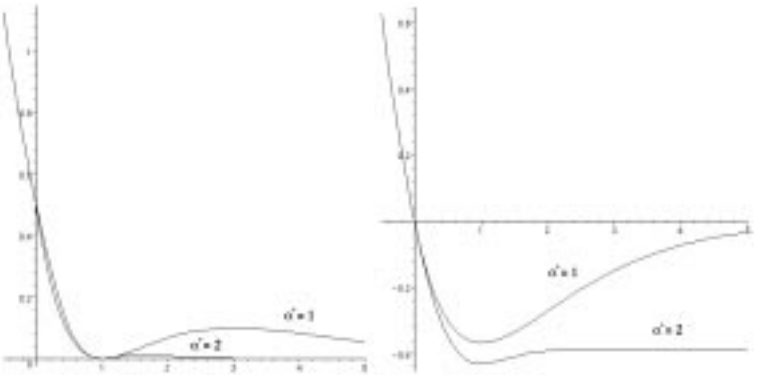

Figure 5.2: Individual error E_p for the *WQL* (5.2) (left) and the *WQL* derivative (5.3) (right) for $\alpha^+ = 1, 2$ (α^- is fixed to 0). The target value is $+1$ and the X axis indicates the output function.

2. For every α^- and every y, f satisfying $yf \geqslant 1$ we have

$$\lim_{\alpha^+ \to \infty} E_p(f, y, \alpha^+, \alpha^-) = A.$$

Then, if $D = \{(x_1, y_1), \ldots, (x_L, y_L)\}$ is a linearly separable data set, the hyperplane $h(x)$ that maximizes the normalized margin also minimizes asymptotically ($\alpha^+ \to \infty$)

$$E_P(f, D) = \sum_{i=1}^{L} E_p(f(x_i), y, \alpha^+, \alpha^-). \tag{5.4}$$

Proof. Since A is a lower bound for E_p, we have $E_P(f, D) \geqslant L \cdot A$ for any function $f(x)$. Since D is linearly separable, $h(x)$ satisfies that support vectors have margin $y_i h(x_i) = 1$, whereas $y_i h(x_i) > 1$ for non-support vectors. The second hypothesis implies that, for every $x_i \in D$, $E_p(h(x_i), y, \alpha^+, \alpha^-) = A$ asymptotically ($\alpha^+ \to \infty$). □

Remarks.

1. The theorem holds true independently on whether the data set D is linearly separable either in the input space or in the feature space.

2. The previously suggested weighted error functions (5.2) and (5.3) satisfy the hypothesis of the theorem, with the additional property that the absolute minimum of E_p is attained when the margin equals 1.

3. The reciprocal may not be necessarily true, since there can be many different hyperplanes which asymptotically minimize $E_P(f, D)$. However, the solution obtained by minimizing $E_P(f, D)$ is expected to have a similar behavior than the hyperplane of maximal margin. In particular, using (5.2) or (5.3):

   (a) It is expected that a large α^+ will be related to a hard margin.

   (b) For the linearly separable case, the expected margin for every support vector is 1.

   (c) For the non-linearly separable case, points with margin less or equal than 1 are expected to be support vectors.

Experimental results with several artificial and real-world problems suggest that these hypotheses seem to be well founded (see sections 5.5 and 5.6).

The relationship among α^+, the learning algorithm, and the hardness of the margin deserves special attention. Suppose that E_P is minimized with an iterative procedure, such as Back-Propagation (BP), and the data set is linearly separable. For large α^+, the contribution of super-classified points to E_P can be ignored. Far away from the minimum there exist points whose margin is smaller than $1 - \varepsilon$, for a certain small $\varepsilon > 0$. Very close to the minimum, in contrast, the margin value of every point is greater than $1 - \varepsilon$. Using the same terminology as in the SVMs approach, the number of bounded support vectors decreases as the number of iterations increases, leading to a solution without bounded support vectors. In other words, for linearly separable data sets and large α^+, the effect of an iterative procedure minimizing E_P is the increase of the hardness of the solution with regard to the number of iterations. For small α^+, in contrast, the contribution of super-classified points cannot be ignored, and the solutions obtained probably share more properties with the regression solution (i.e., the solution minimizing the standard quadratic loss function).

For non-linearly separable data sets, it seems that the behavior could be very similar to the linearly separable case (in the sense of the hardness of the solution). In this case, the existence of misclassified points will lead to solutions having a balanced combination of bounded and non-bounded support vectors. As a consequence, solutions close to the minimum may lead to overfitting, since they are being forced to concentrate on the hardest points to classify, which may be outliers or wrongly labeled points. The same sensitivity to noise was observed for the AdaBoost algorithm in [Dietterich 2000].

Since the contribution of every point to the error is different in both the WQL and regression models, the respective solutions may also be very different depending on the distribution of the points in the data set. For example, well classified outliers far away from the decision boundary have a great influence for standard quadratic

loss function and null influence for *WQL*. In contrast, outliers near the decision boundary affect *WQL* much more than if the contribution is not weighted.

5.3.3 Practical Considerations

Some benefits can be obtained by minimizing an error function as the one defined in (5.4), since there is no assumption about the existence of any predetermined architecture in the FNN:

1. In the final solution, there is no need to have as many hidden units as points in the data set (or any subset of them), nor the frequencies must be the points in the data set. Therefore, a more "classical" FNNs approach may be used.

2. There is no need to use kernel functions, since there is no inner product to compute in the feature space.

In addition, it allows to deal with multiclass and multilabel problems as FNNs usually do:

1. For C-class problems, an architecture with C output units may be constructed, so that the learning algorithm minimizes the multiclass *WQL*, defined as usual [Bishop 1995a]:

$$E_P^C(f_{FNN}^{o,c}, D) = \sum_{i=1}^{L} \sum_{c=1}^{C} E_p(f_{FNN}^{o,c}(x_i), y_i^c, \alpha^+, \alpha^-). \tag{5.5}$$

2. The error function defined in (5.5) also allows to deal with multilabel problems with the same architecture.

5.4 Experimental Motivation

We performed some experiments on both artificial (section 5.5) and real data (section 5.6) in order to test the validity of the new model and the predicted behavior explained in section 5.3.

Regarding to the new proposed FNNs training model, we were interested in testing:

1. Whether learning is possible or not with a standard FNN architecture when a *WQL* function $E_P(f, D)$ is minimized with standard methods.

2. Whether the use of non-kernel functions can lead to a similar behavior to that of kernel functions or not.

3. The effect of large α^+, together with the number of iterations, on the hardness of the margin.

4. The identification of the support vectors, simply by comparing their margin value with 1.

5. The behavior of the model in multiclass and multilabel problems minimizing the error function defined in (5.5).

6. The behavior of the model in both linearly and non-linearly separable cases, with linear and non-linear activation functions.

All *WQL* experiments were performed with Multi-Layer Perceptrons (MLPs) trained with standard BP weighting the sum-of-squares error derivative (5.3). The parameter α^- was set to 0 in all experiments, so that misclassified points had always a weight equal to 1 (i.e., no modification of their contribution). From now on, we will refer to this method as *BPW*. If not stated otherwise, every architecture had linear output units. Artificial problems were trained in batch mode, whereas real-world problems were trained in pattern mode.

In real-world problems, we made several comparisons among FNNs trained with *BPW* (for several activation functions and number of epochs) and SVMs with different kernels. Our main interest was to investigate the similarities among the partitions that every model induced on the input space. We can approximately do that by comparing the outputs of the different learned classifiers on a test set, containing points never used during the construction of the classifier.

5.5 Experiments on Artificial Data Sets

5.5.1 Two Linearly Separable Classes

The first experiment consisted in learning the maximal margin hyperplane of two linearly separable classes. We constructed two different linearly separable data sets ($L1$ and $R1$), shown in figure 5.3. Despite of their apparent simplicity, there is a big difference between the maximal margin hyperplane (dashed line) and the minimum sum-of-squares hyperplane (dotted line), used as the initial weights for *BPW* in an MLP without hidden layers.

Solid lines in figure 5.3 show the resulting hyperplanes after the training (until numerical convergence) for different values of α^+. As it can be observed, the maximal margin hyperplane was obtained for $\alpha^+ = 9$, so that the effect of large α^+ together with the number of iterations on the hardness of the resulting solution was confirmed. When looking at the output calculated by the network, we could see that every point had functional margin strictly greater than 1 except:

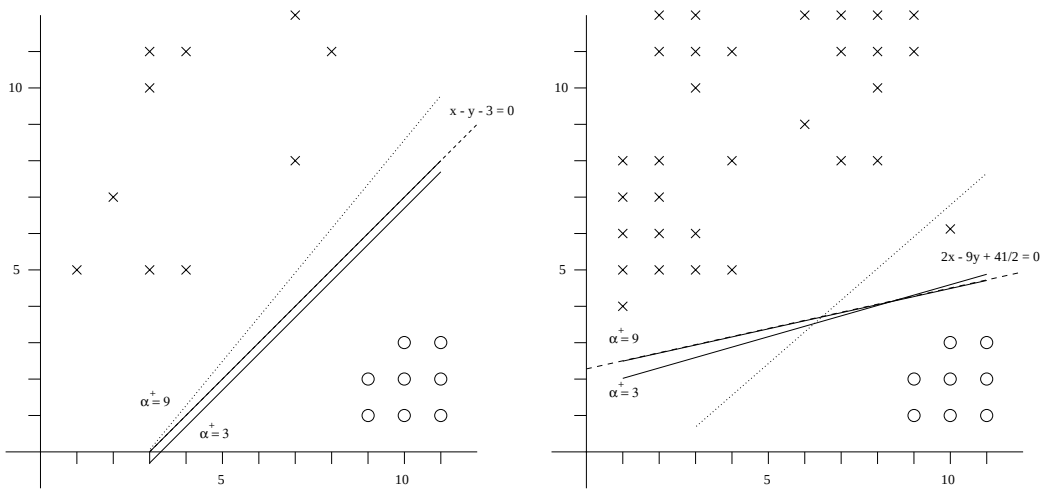

Figure 5.3: Separating hyperplanes (solid lines) for BPW and different values of α^+ in the two-class linearly separable problems $L1$ (left) and $R1$ (right). Dotted and dashed lines represent the minimum sum-of-squares and the maximal margin hyperplanes, respectively.

- for $L1$, points $\{(9,2),(10,3),(4,5),(7,8)\}$;
- for $R1$, points $\{(10,3),(1,4),(10,6)\}$;

which had margin very close to 1. These points are, respectively, the support vectors of the maximal margin hyperplane of the data sets, confirming the prediction about the support vectors just by looking at their margin value.

5.5.2 Three Linearly Separable Classes

The second experiment consisted in learning three linearly separable classes. As in the previous experiments, we constructed two different linearly separable data sets ($L2$ and $R2$), shown in figure 5.4. In this case, the constructed MLPs had three output units, and the training algorithm minimized (5.5). In the same conditions as in the previous section, solid lines in figure 5.4 show the resulting hyperplanes (the output function for every output unit) after the minimization of the weighted sum-of-squares for $\alpha^+ = 9$. We looked at the output calculated by the network for every point in the data set, in order to identify the support vectors. Splitting the resulting network into one network for every class, we observed that every output unit of every network, as in the two linearly separable case, had functional margin strictly greater than 1 for every point in the data set except

- for $L2$:
  - $\{(9,3),(3,3),(9,7)\}$ for the circled points class;

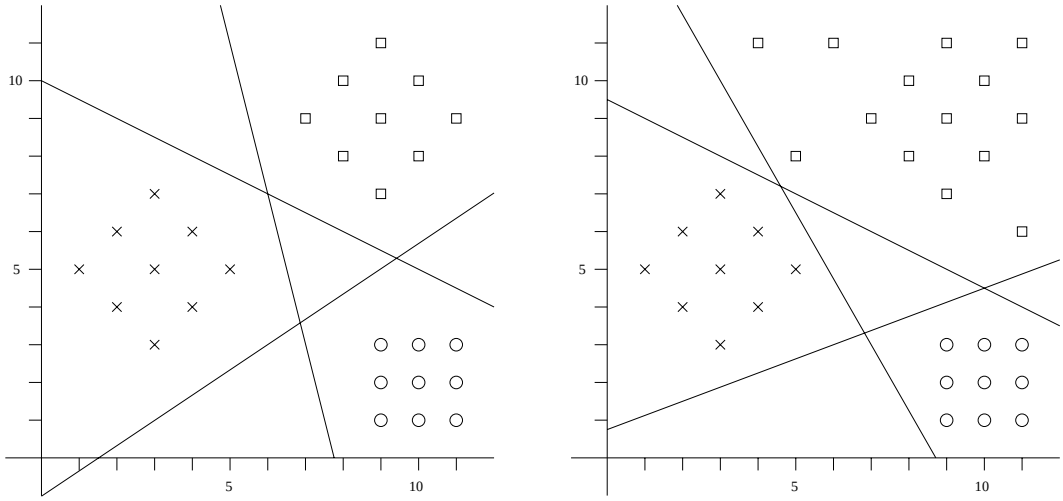

Figure 5.4: Separating hyperplanes for *BPW* multiclass (see (5.5)) for $\alpha^+ = 9$ in the three-class linearly separable problems $L2$ (left) and $R2$ (right).

  - $\{(9,1),(5,5),(7,9)\}$ for the crossed points class;
  - $\{(11,3),(3,7),(9,7)\}$ for the squared points class;

- for $R2$:
  - $\{(9,3),(3,3),(11,6)\}$ for the circled points class;
  - $\{(9,1),(5,5),(5,8)\}$ for the crossed points class;
  - $\{(11,3),(3,7),(5,8)\}$ for the squared points class;

which had margin very close to 1. These vectors are the support vectors obtained after solving the *one-vs-all* binarization of the problem. It confirms our hypothesis about the applicability of the model for multiclass problems.

5.5.3 The *Two Spirals* Data Set

A description of the *Two Spirals* data set can be found in appendix A. An SVM with Gaussian kernels and standard deviation 1 was constructed using the SVM^{light} software [Joachims 1999][4]. We did not obtain satisfactory results either for polynomial or sigmoidal kernels. The hard margin solution contained 176 support vectors (0 bounded). In order to make a comparison with an FNN with the same activation functions and frequencies, we constructed a network with 194 hidden Gaussian RBF units (also with standard deviation 1). The frequencies were fixed to be the points

[4] Available from `http://svmlight.joachims.org`.

Figure 5.5: Generalization obtained by SVM^{light} (left), *BPW* with Gaussian functions (center) and *BPW* with sine functions (right) for the *Two Spirals* data set. The results for *BPW* are the mean over 10 runs.

in the data set, and the initial range for the coefficients was 0.001. Since it is a separable problem with Gaussian kernels, we set $\alpha^+ = 9$. After 10 runs of a training with *BPW*, the mean of the number of points with functional margin less than 1.05 ("support vectors" in our model) was 168. These points were always a subset of the support vectors obtained with the SVM^{light} software. None of them had functional margin less than 0.95.

We also constructed an MLP with a hidden layer of 24 sinusoidal units, as in [Sopena et al. 1999a]. Initial frequencies for *BPW* were randomly assigned to an interval $[-3.5, 3.5]$, and the initial range for the coefficients was 0.0001. We set again $\alpha^+ = 9$. After 10 runs of a training with *BPW*, the mean of the number of points with functional margin less than 1.05 was 101.6, and none of them had functional margin less than 0.95. These results confirm that there is no need to use either an "SVM architecture" or kernel functions (the sine is not a kernel function) when minimizing the *WQL* proposed.

The good generalization obtained by these models can be seen in figure 5.5, where the corners are $(-6.5, -6.5)$ and $(+6.5, +6.5)$. It is worth noting that all the points in the training and test sets are radially equidistant inside a disk of radius 6.5. Therefore, while Gaussian functions are expected to have a good behavior for this problem (the radial equidistance should favor radial functions), it is not so clear *a priori* for sine functions.

5.6 Experiments on Real-World Data Sets

We have concentrated on two classic classification problems from the Natural Language Processing domain, namely Word Sense Disambiguation and Text Categorization, for carrying out the experimental evaluation with real data. These experiments

were possible due to the collaboration of the author with Dr. Lluís Màrquez and Xavier Carreras [Romero et al. 2004b].

5.6.1 Word Sense Disambiguation

Word Sense Disambiguation (WSD) or lexical ambiguity resolution is the problem of automatically determining the appropriate *meaning* or *sense* to each word in a text or discourse. Resolving the ambiguity of words is a central problem for Natural Language Processing applications and their associated tasks, including, for instance, natural language understanding, machine translation, and information retrieval and extraction [Ide and Véronis 1998]. Although far from obtaining satisfactory results [Kilgarriff and Rosenzweig 2000], the best WSD systems up to date are based on SML algorithms. SVM-based classifiers are among the top systems for this problem [Lee and Ng 2002]. A brief description of the WSD data set used in our experiments can be found in appendix A.

5.6.1.1 Methodology

Mainly due to the high number of features, the problem is linearly separable (note that this does not imply that the problem should be easy to resolve, and in fact it is not). This is why we have compared only linear models of *BPW* and SVMs in this experiment.

More specifically, we trained two linear FNN architectures for 200, 500, 1,000 and 2,000 epochs. Both models were trained using *BPW*. The first ones (*bpw-1*) used a value of $\alpha^+ = 1$, while the second ones (*bpw-7*) were trained with $\alpha^+ = 7$. The problem was not binarized, so that *BPW* was trained to minimize (5.5). Regarding SVMs, an extensive exploration of the C parameter was done for linear models in order to determine the ranges in which some differences in accuracy took place. We determined that a value of $C = 1$ corresponds to a hard-margin solution, while a value of C between 10 and 20 times lower can be considered a soft margin. Therefore, we compared three SVM linear models using C values of 0.05, 0.1, and 1 (from soft to hard). The SVM model was trained on a *one-vs-all* binarized version of the training corpus.

5.6.1.2 Results

Table 5.1 contains the basic information about the learning models and the global accuracy results obtained by each of them on the WSD problem. Note that the accuracy figures have been calculated by averaging over the 4 words under study (see appendix A), considering all the examples together (micro-average).

Identifier	**Algorithm**	**Act.F.**	**NEpochs**	**Accuracy**
bpw-1-200	*BPW*	lin	200	72.45%
bpw-1-500	*BPW*	lin	500	73.99%
bpw-1-1000	*BPW*	lin	1,000	73.37%
bpw-1-2000	*BPW*	lin	2,000	73.37%
bpw-7-200	*BPW*	lin	200	72.76%
bpw-7-500	*BPW*	lin	500	73.68%
bpw-7-1000	*BPW*	lin	1,000	74.30%
bpw-7-2000	*BPW*	lin	2,000	73.37%
Identifier	**Software**	**Kernel**	**C-value**	**Accuracy**
svm-C005	SVM^{light}	lin	0.05	73.37%
svm-C01	SVM^{light}	lin	0.1	73.99%
svm-C1	SVM^{light}	lin	1	72.45%

Table 5.1: Description and accuracy results of all models trained on the WSD problem.

It can be seen that all methods achieve accuracy rates that are significantly higher than the baseline 48.54% determined by the *most-frequent-sense classifier*. Additionally, all induced classifiers achieve comparable accuracy rates (ranging from 72.45% to 74.30%) confirming that both approaches are competitive in the WSD domain. Parenthetically, the best result, 74.30%, corresponds to the *bpw-7-1000* model. It should be noted that, although it could seem quite a low accuracy, the five best performing systems in the SensEval-2 competition (including Boosting-based and SVM-based classifiers) achieved a global accuracy between 60% and 65% on the whole set of 73 words. Our figures are mostly comparable to these systems when restricting to the 4 words treated in this study. It is also worth mentioning that a moderate overfitting to the training examples is observed by both methods. *BPW* models slightly overfit with the number of epochs and SVMs with the hardness of the margin (i.e., large values of the C parameter).

But the main goal in this experiments is to compare the similarities and differences among the induced classifiers rather than solely the accuracy values achieved. For that, we calculated the agreement ratio between each pair of models on the test set (i.e., the proportion of test examples in which the two classifiers agree in their predictions). Additionally, we have calculated the Kappa statistic[5]. Table 5.2 contains a subset of these comparisons which allows us to extract some interesting conclusions about the similarities and differences among the models learned. It can be observed that:

[5]The Kappa statistic (κ) [Cohen 1960] is a measure of inter-annotator agreement which reduces the effect of chance agreement. A Kappa value of 0 indicates that the agreement is purely due to chance, whereas a Kappa value of 1 indicates perfect agreement. A Kappa value of 0.8 and above is considered as indicating good agreement.

	svm-C005		*svm-C01*		*svm-C1*	
	Agreement	**Kappa**	**Agreement**	**Kappa**	**Agreement**	**Kappa**
bpw-1-200	94.74%	0.8964	93.48%	0.8787	91.64%	0.8481
bpw-1-500	96.28%	0.9286	95.98%	0.9279	93.50%	0.8855
bpw-1-1000	96.28%	0.9309	95.98%	0.9256	93.81%	0.8882
bpw-1-2000	94.43%	0.8972	95.36%	0.9136	93.50%	0.8827
bpw-7-200	96.90%	0.9397	95.35%	0.9164	93.18%	0.8810
bpw-7-500	96.28%	0.9305	97.21%	0.9527	95.67%	0.9291
bpw-7-1000	94.74%	0.8994	96.90%	0.9428	96.59%	0.9406
bpw-7-2000	93.19%	0.8717	95.67%	0.9199	96.28%	0.9341

Table 5.2: Agreement and Kappa values between linear *BPW* and linear SVM models trained on the WSD problem (test set).

1. All *BPW* models are fairly similar in their predictions to all SVM models. Note that the agreement rates are always over 91% and that the Kappa values are over 0.84, indicating very high agreement. This is specially remarkable, since the agreement (and Kappa) values among the three SVM linear models (information not included in table 5.2) range from 94.42% ($\kappa = 0.899$) and 97.21% ($\kappa = 0.947$).

2. Roughly speaking, the *BPW* models with $\alpha^+ = 7$ are more similar to SVMs than the ones with $\alpha^+ = 1$ (specially for large values of C), suggesting that large values for α^+ are more directly correlated with margin maximization.

3. Restricting to *bpw-7* models, another connection can be made between the number of epochs and the hardness of the margin. On the one hand, comparing to the *svm-C005* models (SVMs with a soft margin) the less number of epochs performed the higher agreement rates are achieved. On the other hand, this trend is inverted when comparing to the hard-margin model of SVMs (*svm-C1*). Note that there are two cells in the table that seems to contradict this statement, since the agreement between *bpw-7-2000* and *svm-C1* should not be lower than the agreement between *bpw-7-1000* and *svm-C1*, and this is not the case. However, note that the difference in agreement is due to the different classification of a unique example, and therefore they can be considered almost equivalent. Put in another way, restricting to the *bpw-7-2000* row, the more harder the SVMs margin is, the more similar the models are, whereas in the *bpw-7-200* row the tendency is completely the opposite.

4. Note that the behavior described in the last point is not so evident in the *bpw-1* model, and that in some cases it is even contradictory. We think that this fact is giving more evidence to the idea that large values of α^+ are required to

resemble the SVM model, and that the hardness of the resulting solution can be controlled with the number of iterations.

5.6.2 Text Categorization

Text Categorization (TC), or classification, is the problem of automatically assigning text documents to a set of pre-specified categories, based on their contents. Since the seminal works in the early 60's, TC has been used in a number of applications, including, among others: automatic indexing for information retrieval systems, document organization, text filtering, hierarchical categorization of Web pages, and topic detection and tracking [Sebastiani 2002]. From the 90's, many statistical and SML algorithms have been successfully applied to the TC task. There is a general agreement in that SVMs are among the top-notch performance systems. A brief description of the TC data set and the evaluation measures used in our experiments can be found in appendix A.

5.6.2.1 Methodology

The description of the models tested can be seen in table 5.3, together with the F_1 results[6] obtained on the test corpus and micro-averaged (i.e., considering all the examples together) over the 10 categories. As in the WSD data set, the problem was not binarized for FNNs. Additionally, in this data set we had the opportunity of testing the new model in a multilabel problem. Several MLP architectures were trained with *BPW* minimizing (5.5), combining activation functions, number of hidden units and number of epochs. We set $\alpha^+ = 7$, so that the effect of super-classified points can be ignored. The non-linear activation functions used were hyperbolic tangent (*tnh*) and sine (*sin*). We also trained a linear (*lin*) architecture with standard BP. The F_1 results for FNNs are the average-output committee of the resulting networks for 5 different runs. As usual, the problem was binarized for SVMs. We used the LIBSVM software [Chang and Lin 2002][7] to test several models with linear, Gaussian (*gau*) and sigmoidal (hyperbolic tangent) kernels, and different values of the parameter C. We can observe the different scale of the hardness of the margin with regard to WSD, with values ranging from $C = 20$ to $C = 200$. We used LIBSVM instead of SVM^{light}due to some convergence problems of the latter in this data set. Both for FNNs and SVMs, every non-linear activation function tested obtained a satisfactory performance. For SVMs with sigmoidal kernels, the best parameters made the sigmoidal function work very similar to a linear function, leading to models almost identical to linear SVMs. We have not included these results in the tables.

[6]The F_1 measure is the harmonic mean of precision and recall: $F_1(P, R) = 2 \cdot P \cdot R/(P+R)$ (see appendix A).

[7]Available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Identifier	Algorithm	Act.F.	NEpochs	F_1
bp-lin-500	BP	lin	500	84.09
bpw-lin-50	*BPW*	lin	50	88.84
bpw-lin-200	*BPW*	lin	200	89.12
bpw-lin-500	*BPW*	lin	500	88.81
bpw-tnh-lin-50	*BPW*	tnh-lin (35H)	50	89.93
bpw-tnh-lin-200	*BPW*	tnh-lin (35H)	200	89.77
bpw-tnh-lin-500	*BPW*	tnh-lin (35H)	500	89.41
bpw-sin-lin-50	*BPW*	sin-lin (20H)	50	89.87
bpw-sin-lin-200	*BPW*	sin-lin (20H)	200	88.93
bpw-sin-lin-500	*BPW*	sin-lin (20H)	500	88.30
Identifier	**Software**	**Kernel**	**C-value**	F_1
svm-lin-C20	LIBSVM	linear	20	88.20
svm-lin-C50	LIBSVM	linear	50	88.85
svm-lin-C200	LIBSVM	linear	200	89.09
svm-gau-C20	LIBSVM	Gaussian	20	89.14
svm-gau-C50	LIBSVM	Gaussian	50	89.62
svm-gau-C200	LIBSVM	Gaussian	200	89.02

Table 5.3: Description of the different models for the comparison on the TC problem. For BP and *BPW*, the "Act.F." column indicates the activation function of every layer and the number of hidden units.

In contrast, when we trained an FNN with *BPW* and hyperbolic tangents as activation functions, the results were very satisfactory (see table 5.3) and the resulting models were different from linear SVMs (see table 5.4).

5.6.2.2 Results

We can see that the linear FNN trained with standard BP obtained a poor performance, whereas the other linear classifiers (*BPW* and SVMs) clearly outperformed this model (see table 5.3). Therefore, it seems that the inductive bias provided by the maximization of the margin has a positive effect in this problem, when linear classifiers are used. In contrast, for non-linear functions this effect was not observed (we also trained several architectures with standard BP and non-linear activation functions, leading to similar results to those of non-linear models of table 5.3). As in WSD, a slight overfitting was present in the non-linear models tested: *BPW* with regard to the number of epochs and SVMs with regard to the hardness of the margin.

Table 5.4 shows the comparison of the predictions in the test set (agreement[8]

[8]Due to the vast majority of negatives in all the (*document, category*) binary decision of the TC problem, the *negative-negative* predictions have not been taken into account to compute agreement ratios between classifiers.

and Kappa values) among several SVM models with linear kernels and the solutions obtained with *BPW*:

1. Looking only at linear *BPW* models, a strong correlation between the number of epochs and the hardness of the margin can be observed. It can be checked by simply looking at the table by rows: *BPW* models with 50 epochs tend to be more different as the hardness of the margin increases, whereas *BPW* models with 500 epochs tend to be more similar to hard margin SVM models. This confirms again the relationship between the hardness of the margin and the number of iterations, provided α^+ has a large value. For *BPW* with non-linear activation functions this behavior is not so clear, although there also exist similar tendencies in some cases (see, for example, the rows of the models with 500 epochs).

2. Looking at the table by columns, the tendency of the SVM model with $C = 20$ is to be more similar to the *BPW* models with 50 epochs. However, the SVM model with the hardest margin ($C200$) significantly tends to be more similar to models with many epochs only for linear *BPW* activation functions. For non-linear functions, the tendency is the opposite. Looking at the most similar models between SVMs and *BPW*, we can see that, as expected, the most similar models to *svm-lin* are those of *bpw-lin*, with very significant differences over the non-linear ones. The differences decrease as the margin becomes harder.

The comparison of *BPW* models with SVMs with Gaussian kernels can be seen in table 5.5. A similar behavior can be observed, but with some differences:

1. The tendency of linear *BPW* models changes: the less similar model to those with 200 and 500 epochs is now *svm-gau-C200*, the model with hardest margin. In contrast, non-linear *BPW* models have the same tendency previously shown, leading to a situation where the agreement rates between *svm-gau-C200* and any other model is low. This may be indicating that the harder the margin (either with a larger C in SVMs or more epochs in *BPW*), the highest the importance of the kernel is.

2. Surprisingly, there exists a strong similarity between SVMs with Gaussian kernels and linear *BPW*, specially for non-hard margin SVM models. For non-linear activation functions, *BPW* models with few epochs are also quite similar to these SVM models. This may be indicating that soft margin models are quite similar among them, regardless of the kernel function. To confirm this hypothesis, we also looked at the agreement among the different SVM models (table 5.6). As it can be observed, the differences grow up as the hardness for the Gaussian kernel takes more extreme values.

	svm-lin-C20		*svm-lin-C50*		*svm-lin-C200*	
	Agreement	**Kappa**	**Agreement**	**Kappa**	**Agreement**	**Kappa**
bpw-lin-50	96.28%	0.86	95.70%	0.83	92.31%	0.70
bpw-lin-200	93.88%	0.76	95.62%	0.82	95.24%	0.80
bpw-lin-500	91.89%	0.69	94.13%	0.76	95.26%	0.80
bpw-tnh-lin-50	93.81%	0.75	94.56%	0.77	93.60%	0.73
bpw-tnh-lin-200	87.88%	0.53	89.66%	0.58	91.61%	0.64
bpw-tnh-lin-500	86.44%	0.48	87.77%	0.51	89.34%	0.56
bpw-sin-lin-50	92.52%	0.70	93.53%	0.73	93.43%	0.72
bpw-sin-lin-200	86.81%	0.51	87.88%	0.53	89.57%	0.58
bpw-sin-lin-500	84.06%	0.43	85.06%	0.44	86.63%	0.48

Table 5.4: Agreement and Kappa values between *BPW* and linear SVM models on the TC problem (test set).

	svm-gau-C20		*svm-gau-C50*		*svm-gau-C200*	
	Agreement	**Kappa**	**Agreement**	**Kappa**	**Agreement**	**Kappa**
bpw-lin-50	95.69%	0.83	93.60%	0.74	89.16%	0.58
bpw-lin-200	95.81%	0.82	95.95%	0.82	92.32%	0.68
bpw-lin-500	94.12%	0.76	95.05%	0.79	92.91%	0.71
bpw-tnh-lin-50	95.08%	0.79	94.64%	0.77	91.27%	0.64
bpw-tnh-lin-200	89.96%	0.59	92.05%	0.65	92.84%	0.69
bpw-tnh-lin-500	88.07%	0.52	89.63%	0.56	90.25%	0.59
bpw-sin-lin-50	93.98%	0.75	94.20%	0.75	92.08%	0.67
bpw-sin-lin-200	88.37%	0.55	89.68%	0.58	90.60%	0.62
bpw-sin-lin-500	85.60%	0.46	86.85%	0.48	87.97%	0.53

Table 5.5: Agreement and Kappa values between *BPW* and Gaussian SVM models on the TC problem (test set).

	svm-gau-C20		*svm-gau-C50*		*svm-gau-C200*	
	Agreement	**Kappa**	**Agreement**	**Kappa**	**Agreement**	**Kappa**
svm-lin-C20	96.44%	0.86	93.16%	0.73	88.85%	0.58
svm-lin-C50	98.52%	0.94	95.87%	0.83	90.94%	0.64
svm-lin-C200	94.63%	0.78	96.45%	0.87	94.07%	0.75

Table 5.6: Agreement and Kappa values between linear SVMs and Gaussian SVM models on the TC problem (test set).

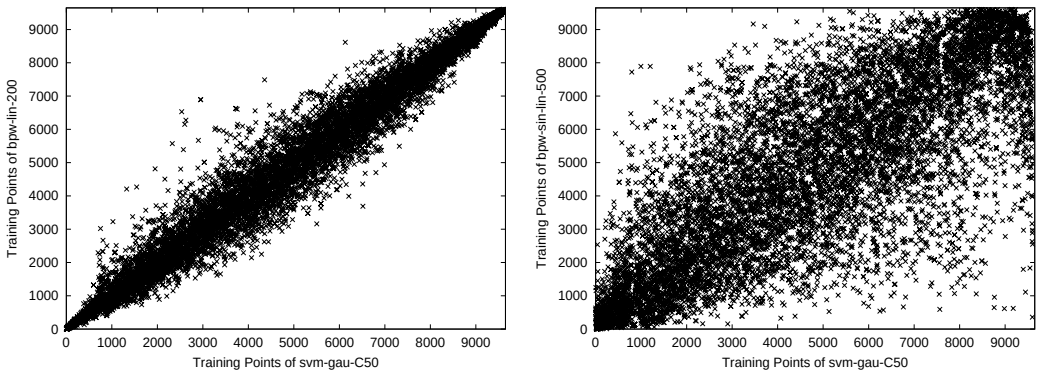

Figure 5.6: Comparison of training vectors between *svm-gau-C50* (X axis) and *bpw-lin-200* (left) or *bpw-sin-lin-500* (right).

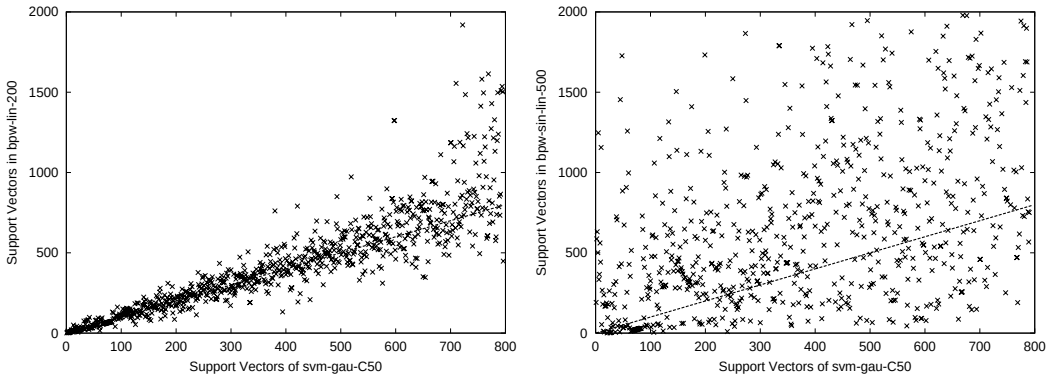

Figure 5.7: Comparison of support vectors between *svm-gau-C50* (X axis) and *bpw-lin-200* (left) or *bpw-sin-lin-500* (right).

Figures 5.6 and 5.7 show another interesting comparison regarding the training vectors. This experiment aims at investigating the relationship between the margin of training vectors in the SVM model and those of the *BPW* model. The SVM model chosen for the comparison was *svm-gau-C50*. We selected two different *BPW* models, one very similar in agreement to *svm-gau-C50* and another very different, *bpw-lin-200* and *bpw-sin-lin-500*, respectively (see table 5.5). In the X axis of the plots in figure 5.6 we have the $9,603$ training vectors of the *svm-gau-C50* model for the binarized problem of class `earn` (the most frequent category), ordered by its margin value (i.e., the first points in the left of each plot are those training points with a lower margin value). In the Y axis it is shown the position that every vector would occupy if the respective model had been ordered following the same criterion (left for *bpw-lin-200* and right for *bpw-sin-lin-500*). In figure 5.7 we have the same plot only for the 796 support vectors of the *svm-gau-C50* model[9]. A straight line

[9]Although, theoretically, non-bounded support vectors have margin 1, the computationally

indicates the (ideal) exact coincidence between the two models. It can be clearly seen that there exists a very strong correlation for *bpw-lin-200*, whereas the correlation with *bpw-sin-lin-500* is much weaker. Therefore, these models not only are similar or different (see table 5.5) in their predictions, but also in the importance that both give to the points in the training set, in particular to the support vectors. A similar behavior to that in figures 5.6 and 5.7 was also observed for the remaining nine categories of the problem.

obtained margin may not be exactly 1. We have ordered the vectors by its computational margin. The percentage of support vectors of the *svm-gau-C50* model which occupy a position inferior to 796 are 88.69% for the *bpw-lin-200* model and 59.93% for *bpw-sin-lin-500*.

Chapter 6

Conclusions and Future Research

In this chapter we summarize the contributions of the thesis. We also draw some concluding remarks about the work and discuss some extensions and directions for future research.

6.1 Conclusions

Three schemes related to the control of the Bias/Variance decomposition for Feed-forward Neural Networks (FNNs) with the (sometimes modified) quadratic loss function have been presented. These schemes deal with different components of the learning process with FNNs: the network architecture, the input dimension and the loss function.

An extensive experimentation has been carried out with artificial data sets, benchmark data sets from several well-known Machine Learning repositories and two real-world problems from the Natural Language Processing domain. The overall results can be considered as very satisfactory.

In addition to widely used activation functions, such as the hyperbolic tangent or the Gaussian function, other activation functions have been tested. In particular, the sine and cosine functions showed a very good behavior.

6.1.1 *SAOCIF*

Chapter 3 has been devoted to describe an algorithm for sequential approximation with FNNs, referred to as *Sequential Approximation with Optimal Coefficients and Interacting Frequencies* (*SAOCIF*). The main ideas of *SAOCIF* can be summarized as follows:

1. The frequencies (the non-linear weights) are selected at every step taking into account the interactions with the previously selected terms.

2. The interactions are discovered by means of their optimal coefficients (the linear weights).

A number of candidate frequencies are obtained at every step using different heuristics. Every candidate frequency is installed in the network, and the whole set of coefficients are optimized, in order to test the real contribution of the frequency to the approximation to the target vector. The candidate frequency that, together (interacting) with the previously selected frequencies, allows a better approximation of the target vector is finally selected.

The importance of the interacting frequencies lies in the hypothesis that they allow to find better partial approximations, with the same number of hidden units, than frequencies selected just to match the residue as best as possible. Likewise, the same level of approximation may be achieved with less hidden units. In terms of the Bias/Variance decomposition, it will be possible to obtain simpler models with the same bias.

The proposed algorithm can be seen as an extension and generalization of the Orthogonal Least Squares Learning algorithm [Chen et al. 1991a], the learning algorithm for Multi-Layer Perceptrons described in [Zhang and Morris 1998] and Kernel Matching Pursuit with *pre-fitting* [Vincent and Bengio 2002] in several ways. First, it is not restricted to select the candidate frequencies from the points in the data set. In this sense, a number of different heuristics can be used. Second, it is possible to further tune the selected frequency.

The idea behind *SAOCIF* can be extended to approximations in Hilbert spaces, maintaining orthogonal-like properties. The theoretical results obtained prove that, under reasonable conditions, the residue of the approximation is (in the limit) the best one that can be obtained with any subset of a given set of vectors. In this case, the importance of the interacting frequencies lies in the expectation of increasing the rate of approximation.

Experimental results on artificial problems show that the selection of frequencies performed by *SAOCIF* allows to obtain better solutions than the idea of matching the residue, both for approximation and generalization purposes. As an example of approximation capability, the *Two Spirals* data set could be adequately learned by *SAOCIF* with sigmoidal units and a suitable strategy. For benchmark data sets, the results obtained with *SAOCIF* are competitive with other results found in the literature for the same problems.

Regarding the activation functions, it is worth noting that non-linear activation functions such as sines or cosines, different from the classical sigmoidal and Gaussian ones, may be satisfactorily used. Linear hidden units have, in some cases, a positive influence on the results when combined with non-linear ones. The use of linear activation functions may help to limit the complexity of the resulting output function. In addition, it seems that there are activation functions which are better

suited for some problems. Although theoretical results of universal approximation apply to many function families, only a few have been used in practice. We think that the choice of the activation function may help to improve the performance of FNNs. This issue is clearly related to the choice of the kernel function for Support Vector Machines.

Regarding the strategy to select the frequencies, both the Breeder Genetic Algorithm (BGA) and the Input strategies appear to be superior to the Random selection. For the benchmark data sets tested, the Input strategy obtains similar (and sometimes superior) results than the BGA strategy, with several additional advantages. First, the Input strategy has a computational cost much smaller than the BGA one. In addition, the Input strategy is deterministic. Finally, selecting the frequencies from the points in the data set seems well suited not only for Radial Basis Function Networks (RBFNs), as commonly used, but also for Multi-Layer Perceptrons (MLPs). In our opinion, the Input strategy should be preferred to the BGA strategy for future experiments. The resulting model shares with Support Vector Machines the property that their frequencies are a subset of the data set. From this point of view, models obtained with *SAOCIF* and the Input strategy are usually quite sparse.

Tuning the selected frequency sometimes helps to improve the overall results, but it does not do it either significantly or consistently. Most of the times, solutions obtained with tuning have less hidden units than without it. However, this fact is not always translated into a better performance.

Parenthetically, it was observed that there exist data sets, such as the *Pima Indians Diabetes* where the results are very similar among a variety of (linear and non-linear) systems. Results found in the literature are also very similar. In our opinion, it is quite difficult to obtain conclusions from this kind of data sets.

6.1.2 SBS for MLPs

In chapter 4, a study and comparison of different criteria to perform Feature Selection (FS) with MLPs and the Sequential Backward Selection (SBS) procedure within the *wrapper* approach has been carried out. In terms of the Bias/Variance decomposition, FS procedures may reduce the variance term by eliminating irrelevant variables. Several critical points have been studied and compared:

1. The stopping criterion of the network training.

2. The data set where the value of the loss function is measured.

3. The network retraining with respect to the computation of the saliency: first train the network and then remove temporarily the feature in order to compute

the saliency or first remove temporarily the feature and then train the network previous to computing the saliency.

For artificial data sets, it was observed that the addition of irrelevant features affects very negatively to the performance of sinusoidal MLPs, even if we try to control the overfitting. The information needed to learn the problem is present, but the system is not able to use it in a proper way. The reason for this fact may be the relatively small number of examples in the data set. As far as irrelevant variables are eliminated, performance improves. We also observed this behavior in several preliminary experiments with the hyperbolic tangent function.

The behavior of non-linear (sinusoidal) models was different from that of linear ones, with respect to both the selected configurations and the generalization results. For the benchmark problems tested, sinusoidal activation functions allow to obtain better results, after FS with the SBS procedure, than linear FNNs. As for *SAOCIF*, the behavior of sinusoidal MLPs was very satisfactory.

Experimental results suggest that the increase in the computational cost associated with retraining the network with every feature temporarily removed previous to computing the saliency can be rewarded with a significant performance improvement, specially if non-linear models are used. Although this idea could be thought as very intuitive, it has been hardly used in practice.

Regarding the data set where the value of the loss function is measured, it seems clear that the SBS procedure for MLPs takes profit from measuring the loss function in a validation set. Again, this is a quite intuitive idea, although many models in the literature do not take this approach. A somewhat non-intuitive conclusion is drawn looking at the stopping criterion, where it is suggested that forcing overtraining may be as useful as early stopping.

In some benchmark data sets, we obtained an important improvement in the overall results with respect to learning with the whole set of variables and compared with other existing FS wrappers in the literature. Although the model can be further improved, the good results obtained are mainly due, in our opinion, to a proper detection of irrelevant variables.

6.1.3 *WQL*

Chapter 5 describes the *Weighted Quadratic Loss* (*WQL*) function, a modification of the quadratic loss function for classification problems inspired in Support Vector Machines (SVMs) [Boser et al. 1992; Cortes and Vapnik 1995] and the AdaBoost algorithm [Schapire and Singer 1999]. The definition of *WQL* was suggested by the behavior of the support vectors and it depends on the margin. In terms of the Bias/Variance decomposition, variance tries to be controlled by not attempting to overfit the points that are already well classified. A theoretical result justifies that

the proposed *WQL* is well founded: in the linearly separable case, the hyperplane that maximizes the normalized margin also minimizes asymptotically the weighted sum-of-squares error function proposed. As in SVMs, the *hardness* of the resulting solution can be controlled, so that this model can also be used for the non-linearly separable case. The final solution is neither restricted to have an architecture with as many hidden units as examples in the data set (or any subset of them) nor to use kernel functions. In addition, it allows to deal with multiclass and multilabel problems as FNNs usually do.

Experiments on artificial data sets show that models equivalent to hard margin SVMs can be obtained by training FNNs with *WQL* in linearly separable cases, both for two-class and multiclass problems. In addition, models similar to non-linear hard SVMs can be obtained without an "SVM architecture" (i.e., with so many hidden units as points in the data set or support vectors) and without kernel functions, whenever the *WQL* is minimized.

In the real-world problems tested, several soft margin SVMs were compared with different FNN models obtained by minimizing *WQL*. A consistent correlation was observed between SVM models (with different hardness of the margin) and FNN ones (minimizing *WQL* with adequate parameters). Both linear and non-linear models (with and without kernel functions) showed this behavior, in two-class and multiclass problems. In particular, the sine function (which is not a kernel function) showed again a very interesting behavior.

Regarding the performance of the induced classifiers, models trained with *WQL* obtained similar and sometimes superior results than SVM models, which are competitive in the respective domains.

6.2 Future Research

6.2.1 *SAOCIF*

The described particular algorithm for *SAOCIF* has several points to study and improve:

1. The computational cost of *SAOCIF* with the Input strategy can be improved simply by selecting the new frequency from a subset of the training set rather than from the whole data set. The size of this subset can be computed so that there is a large probability that at least one frequency is useful (see [Smola and Schölkopf 2000]). In addition, the Input strategy can take profit from, for example, several optimizations described in the literature for the Orthogonal Least Squares Learning algorithm, as in [Chen and Wigger 1995].

2. The candidate frequencies can be selected with heuristics different from current

ones. In principle, a more intelligent selection could lead to better approximations. For example, the data could be preprocessed in order to obtain a set of promising candidate frequencies. Principal Component Analysis [Fukunaga and Koontz 1970; Guterman 1994], or some clustering and mapping procedures [Kohonen 1995; Bishop et al. 1998; Jain et al. 1999] could be useful to that end. In the same way, the selection of the activation function for the new hidden unit admits any number of heuristics.

3. We are not particularly satisfied with the results obtained with the tuning procedure. However, we still think that tuning the frequencies may allow to obtain better approximations with less hidden units. For a gradient-based technique, the main problem lies in the automatic adjustment of the parameters. Therefore, the particular procedure used in this work may be substituted by another adaptive learning algorithm (see, for example, [Jacobs 1989; Fahlman 1988; Tollenaere 1990; Riedmiller and Braun 1993]).

4. In some cases (for example, when the number of hidden units is very large), the linear equations system may degenerate leading to numerical problems. Several techniques can be used in these cases to avoid these problems, such as Singular Value Decomposition (see, for example, [Press et al. 1992]). As an alternative, however, a new hidden layer can be constructed when the number of hidden units is too large or the rate of decrease of the error after the addition of several consecutive hidden units is small, for example. The selection of the frequencies in the new hidden layer can be made following the same ideas than those in *SAOCIF*, whenever the frequencies in the previous layers are kept fixed.

Since the most promising results have been obtained with the Input strategy, some new experiments can be performed in order to compare the inductive bias of FNNs and SVMs:

1. A comparison between *SAOCIF* with the Input strategy and SVMs (specially in data sets where significant differences between SVMs and standard FNNs have been observed). The sparseness of the model and the overall results should be compared, in addition to the final subsets of frequencies in both models (the support vectors in the SVMs terminology).

2. Taking profit from the design of new kernel functions for specific problems, and using them as activation functions for *SAOCIF*. If no tuning of the selected frequency is done, it can be easily carried out.

A different point of view can be introduced in *SAOCIF* if we reconsider the previously selected frequencies, similar to the *back-fitting* scheme in Projection Pursuit

Regression [Friedman and Stuetzle 1981]. Comparing the addition of hidden units in *SAOCIF* with the selection of features in the Sequential Forward Selection procedure for FS, we can see that they share the same general ideas, although applied to different objects. Whereas *SAOCIF* applies the forward selection to the hidden units, Sequential Forward Selection applies it to the features. Therefore, other FS search procedures can be applied to the construction of FNNs. In particular, floating methods [Pudil et al. 1994] may help to obtain more compact networks. Note that the criterion to remove a previously selected hidden unit, needed in these new approaches, cannot be based in the approximation of a previous residue. In contrast, the same idea used by *SAOCIF* to add a new term can be used to remove an old one: the frequency such that, when removed, allows to obtain the smallest error (after computing the optimal coefficients), is the candidate to be eliminated.

SAOCIF can also be applied to other models of FNNs. In [Belanche 2000], for example, Heterogeneous Neural Networks allow to deal with heterogeneous information, such as continuous variables, discrete (ordinal or nominal) ones and fuzzy quantities, based on similarity functions. Missing data are also explicitly considered. The only condition needed to be able to construct an Heterogeneous Neural Network with *SAOCIF* is that the output of the activation function of the hidden units (the similarity) must be a real number, and that condition holds for the proposed models in [Belanche 2000].

Finally, the interpretation of the approximation may be of great interest, especially if we are dealing with real-world problems. In principle, as any sequential method, it can give more information than a non-sequential one, since we can study the resulting approximation after every term has been added. Different sources of information may be, for example, the frequencies, the projection of the data onto the frequencies or the activation functions. It is expected that the first hidden units added will give more information (and probably more important) than the last ones. This may be more interesting if the Input strategy is used.

6.2.2 SBS for MLPs

The experimental results were obtained with the parameters adjusted for the whole set of variables. A revision of these parameters within the SBS procedure could, hopefully, improve the results. For example, learning rates were not readjusted after the elimination of every feature. Although the initial learning rates allowed to fit the training set during the elimination of several variables and the number of epochs was increased as far as variables were eliminated, they may not be adequate when only a few number of variables remain. In this sense, an adaptive learning algorithm may be used, similar to the alternatives to the tuning procedure for *SAOCIF*. The number of hidden units in the architecture may also be revised when the number of features decreases.

In addition, a certain non-determinism was observed in the resulting set of variables. This fact may be inherent to the problem (for example, if there are many subsets of variables that are approximately equivalent), although it would be desirable to be controlled. It is worth noting, however, that the most important variables (i.e., the variables eliminated at the last steps) are very similar among the different runs of the same configuration.

But the main drawback of the SBS procedure for MLPs described in this work is its computational cost. Previous ideas described in the literature with the aim to alleviate this problem, such as the elimination of several features at every step or the approximation of the importance of a feature in a heuristic manner without retraining the network, may work in some cases but have several problems, as previously explained. Training algorithms faster than Back-Propagation (BP) may obviously be used, but BP was not the main source of the computational cost in our experiments. The algorithm is quadratic with respect to the number of variables, and the first steps of the algorithm (i.e., when there are probably many irrelevant variables) take most of the computational time. Several heuristics could be designed in order to eliminate the most clearly irrelevant variables with a relatively low computational cost. Then, when a reasonable number of features remains, the whole procedure would start. In addition, the elimination of a feature can be parallelized, training the networks with every feature temporarily removed in an independent way.

The results shown in chapter 4 were obtained with sinusoidal MLPs minimizing the quadratic loss function. We would like to study whether the resulting selected variables are dependent on the activation function of the MLPs or not. Preliminary studies seem to indicate that there is a certain dependence on it. In the same way, note that the basic scheme described in this work can be tested within any other framework which can be adjusted to the required specifications. For instance, the SBS procedure can be performed with SVMs using some function of the margin as the saliency. The stopping criteria may be related to different hardness of the margin. The fast available SVM implementations make this framework suitable to that end. Those experiments could, in addition, answer to a question that can be posed after this work: Which is the most important element of the process? The SBS procedure? The use of MLPs? The proper combination of both?

Obviously, other search procedures different from the SBS may be tested. Floating methods [Pudil et al. 1994] are good candidates, due to their promising results [Jain and Zongker 1997]. The critical decision points to test do not vary.

6.2.3 *WQL*

The weighted functions proposed in this work are only a first proposal to weight the sum-of-squares error function. We think that this issue deserves further research,

defining weighting functions that may be more robust to overfitting. Several works in this line can be found in [Freund 2001; Rätsch et al. 2001] for the AdaBoost algorithm.

The theoretical work could be extended by looking for conditions (or modifying the weighted functions proposed) such that the reciprocal of Theorem 2 holds. That result would help to shed light on the relationships between the respective inductive bias of FNNs and SVMs.

Although in this work we have only considered classification problems, the same idea can be applied to regression problems, just by changing the condition of the weighting function (5.1) from $mrg(x_i, y_i, f^o_{FNN}) \geqslant 0$ to $|f^o_{FNN}(x_i) - y_i| \leqslant \varepsilon$, where ε is a new parameter that controls the resolution at which we want to look at the data, similar to the ε-insensitive cost function proposed in [Vapnik 1995].

6.2.4 Combination of the Proposed Schemes

The three schemes proposed in this work can be combined among them, since they deal with complementary aspects of the whole learning process:

1. The SBS procedure may be performed with *SAOCIF*.
2. *SAOCIF* can be modified so as to deal with the *WQL* function.
3. The SBS procedure can also be performed trying to minimize the *WQL* function.

It is expected that some of these combinations could, hopefully, take profit from the good properties of each individual scheme.

Appendix A

Data Sets Used in the Experiments

A.1 Artificial Data Sets

A.1.1 HEA Data Sets

In [Hwang et al. 1994] five non-linear functions $g_i : [0,1]^2 \to \mathbb{R}$ are used to generate different data sets:

1. Simple Interaction Function:

$$g_1(x_1, x_2) = 10.391\,((x_1 - 0.4)(x_2 - 0.6) + 0.36)\,.$$

2. Radial Function:

$$g_2(x_1, x_2) = 24.234\left(r^2(0.75 - r^2)\right),$$

where $r^2 = (x_1 - 0.5)^2 + (x_2 - 0.5)^2$.

3. Harmonic Function:

$$g_3(x_1, x_2) = 42.659\left((2 + x_1)/20 + Re(z^5)\right),$$

where $z = x_1 + ix_2 - 0.5(1 + i)$.

4. Additive Function:

$$g_4(x_1, x_2) = 1.3356(\ 1.5(1 - x_1) + e^{2x_1 - 1}\sin(3\pi(x_1 - 0.6)^2) + e^{3(x_2 - 0.5)}\sin(4\pi(x_2 - 0.9)^2)\ ).$$

5. Complicated Interaction Function:

$$g_5(x_1, x_2) = 1.9\left(1.35 + e^{x_1}\sin(13(x_1 - 0.6)^2)e^{-x_2}\sin(7x_2)\right).$$

225 pairs (x_1, x_2) of values were generated from the uniform distribution in $[0, 1]^2$. These data were used for all five functions in order to generate five noise free training sets:

$$\text{HEA}n\text{-NF} = \{(x_1^i, x_2^i, g_n(x_1^i, x_2^i))\}_{i=1,\cdots,255}$$

where $n \in \{1, 2, 3, 4, 5\}$. In addition, another five training data sets were generated adding independent and identically distributed Gaussian noise:

$$\text{HEA}n\text{-WN} = \{(x_1^i, x_2^i, g_n(x_1^i, x_2^i) + 0.25\varepsilon^i)\}_{i=1,\cdots,255}$$

where $\varepsilon^i \sim \mathcal{N}(0, 1)$. The test set was built sampling every function on a regularly spaced grid on $[0, 1]^2$ with $10,000$ points. In summary, 10 training sets (5 noise free and 5 noisy versions) and 5 test sets were generated in [Hwang et al. 1994] for the 5 aforementioned functions. These data sets were used in [Maechler et al. 1990; Hwang et al. 1991, 1992a,b, 1994, 1996; Lay et al. 1994; You et al. 1994; Kwok and Yeung 1995a,b, 1996b,a, 1997b; Treadgold and Gedeon 1997a,b, 1998, 1999a,b; Ma and Khorasani 2003, 2004].

In our experiments, we constructed 10 training sets for every function, each containing 225 points, changing the initial seed of the random function for the uniform distribution (the noise free data sets). Similar to [Hwang et al. 1994], 10 noisy training data sets were generated in the same way. For every function, the test set in [Hwang et al. 1994] was used as a validation set for the adjustment of the parameters. For the final results, a new test set was constructed, with an offset of 0.0025 with respect to the input points of the original test set in [Hwang et al. 1994].

In summary, 100 training sets (50 noise free and 50 noisy versions), 5 validation sets and 5 test sets were generated for the 5 aforementioned functions.

A.1.2 The *Two Spirals* Data Set

The well-known *Two Spirals* problem consists in identifying the points of two interlocking spirals that go around the origin three times. The training, validation and test sets comprise 194 two-dimensional points with balanced classes. The C source code to generate these data sets can be found in the Carnegie Mellon University Artificial Intelligence Repository [Kantrowitz 1993]. The training set is symmetric with respect to the target (i.e., if (x, y) belongs to a class, the training set also contains the point $(-x, -y)$, which belongs to the other class). The validation and test sets are not symmetric, and they are obtained adding an offset to the points in the training set. Figure A.1 shows the training set for this problem.

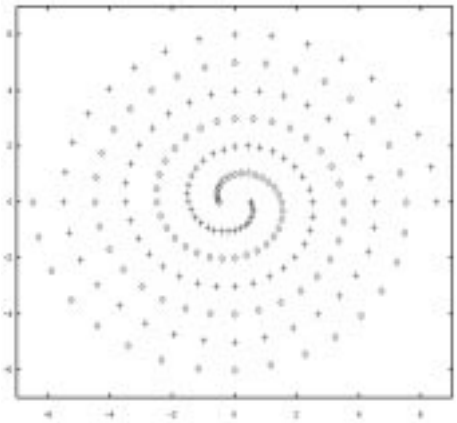

Figure A.1: The *Two Spirals* training set. The points in every spiral are indicated with different symbols.

As it is well known, this is an extremely hard problem for architectures with sigmoidal activation functions because of its intrinsic high non-linearity and radial symmetry [Lang and Witbrock 1988; Fahlman and Lebiere 1990].

A.1.3 The *Augmented Two Spirals* Data Set for FS

We created the *Augmented Two Spirals* data set to perform Feature Selection (FS) experiments by artificially adding 13 new features to the two original variables of the *Two Spirals* data set. These new features were defined to be redundant or independent on the original variables (and therefore irrelevant). Some of them were noisy. The whole set of variables is defined as follows:

1. x_1: The first feature in the original data set.
2. x_2: The second feature in the original data set.
3. x_3: x_1^2.
4. x_4: x_2^2.
5. x_5: $x_1 \cdot x_2$.
6. x_6: $x_1 + x_2$.
7. x_7: $x_1 - x_2$.
8. x_8: $x_1^2 + \mathcal{N}(0,1)$.
9. x_9: $x_2^2 + \mathcal{N}(0,1)$.

10. x_{10}: $x_1 \cdot x_2 + \mathcal{N}(0,1)$.
11. x_{11}: $x_1 + x_2 + \mathcal{N}(0,1)$.
12. x_{12}: $x_1 - x_2 + \mathcal{N}(0,1)$.
13. x_{13}: Values according to a uniform distribution in $[0,1]$.
14. x_{14}: Values according to a $\mathcal{N}(0,1)$ distribution.
15. x_{15}: Values according to a $\mathcal{N}(0,5)$ distribution.

The values of the input variables were linearly scaled in $[-6.5,+6.5]$ when they were not in this range. The target value was that of (x_1, x_2) in the original data set.

A.1.4 The *Augmented XOR* Data Set for FS

We created the *Augmented XOR* data set to perform FS experiments in the following way:

1. First, a symmetric data set was created as an extension of the *XOR* data set to the hypercube $[-1,+1]^2$. 150 points for each class were created. Figure A.2 shows the data set for this problem, generated as

$$\{(z_i + \delta(z_i, c), z_j + \delta(z_j, c)) \mid \; i,j = 0, \cdots, 9 \quad c = 0,1,2\} \tag{A.1}$$

   where

$$z_k = -\frac{9}{10} + \left(\frac{2 \cdot k}{10}\right)$$

   and

$$\delta(z,c) = -\frac{c \cdot \text{sign}(z)}{30}.$$

   The target only depends on the quadrant where the point is placed (+1 for points in the first and third quadrant, and −1 otherwise). Equivalently, it is the sign of the product of the components.

2. Second, we artificially added 11 new features to the two original variables, similar to the *Augmented Two Spirals* data set. These new features were defined to be redundant or independent on the original variables (and therefore irrelevant). Some of them were noisy.

The whole set of variables is defined as follows:

1. x_1: The first feature defined in (A.1).

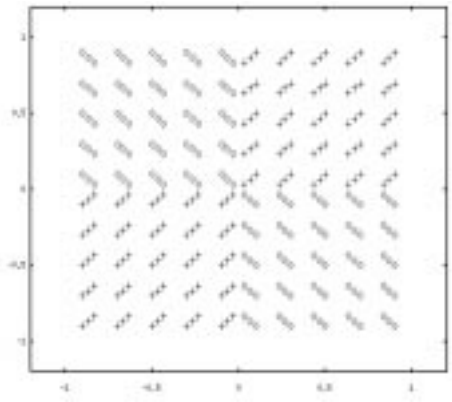

Figure A.2: The extension of the *XOR* data set to the hypercube $[-1,+1]^2$. The points in every class are indicated with different symbols.

2. x_2: The second feature defined in (A.1).
3. x_3: x_1^2.
4. x_4: x_2^2.
5. x_6: $x_1 + x_2$.
6. x_7: $x_1 - x_2$.
7. x_8: $x_1^2 + \mathcal{N}(0,1)$.
8. x_9: $x_2^2 + \mathcal{N}(0,1)$.
9. x_{11}: $x_1 + x_2 + \mathcal{N}(0,1)$.
10. x_{12}: $x_1 - x_2 + \mathcal{N}(0,1)$.
11. x_{13}: Values according to a uniform distribution in $[0,1]$.
12. x_{14}: Values according to a $\mathcal{N}(0,1)$ distribution.
13. x_{15}: Values according to a $\mathcal{N}(0,5)$ distribution.

The values of the input variables were linearly scaled in $[-1,+1]$ when they were not in this range. The target value was the sign of $x_1 \cdot x_2$.

Initially we had also defined x_5 and x_{10} as in the *Augmented Two Spirals* data set (see above). But in this case the problem can be learned with only one of these variables (perfectly with x_5 and almost perfectly with x_{10}), so that it could not allow to see the differences among the configurations, if any. Therefore, we decided to remove these variables from the data set, but maintaining the rest of variables with the same name as in the *Augmented Two Spirals* data set.

Data Set	NVar	NCla	NExamples	Missing
Pima Indians Diabetes	8	2	768 (65.10%/34.90%)	yes
Wisconsin Breast Cancer	9	2	699 (65.52%/34.48%)	no
Hepatitis	19	2	155 (79.35%/20.65%)	yes
Ionosphere	33	2	351 (64.10%/35.90%)	no
Sonar	60	2	208 (53.37%/46.63%)	no

Table A.1: Description of the benchmark data sets. The column 'NVar' shows the number of variables, the column 'NCla' the number of classes and the column 'NExamples' the number of examples. Figures in brackets are the percentage of examples for every class. The column 'Missing' indicates whether the data have missing values or not.

A.2 Benchmark Data Sets

We selected several data sets for classification problems from several well-known Machine Learning repositories: University of California, Irvine repository [Blake and Merz 1998], Statlog [Michie et al. 1994] project and Carnegie Mellon University repository [Kantrowitz 1993]. A brief description of these benchmarks can be seen in table A.1. The data were not preprocessed for any data set except for the Hepatitis one, which was scaled in $[0, 1]$.

The *Pima Indians Diabetes* contains the data of at least 21 years old females Pima Indians living near Phoenix, Arizona, USA. This data set was originally donated by V. Sigillito from the Johns Hopkins University. The task is to decide whether a patient shows signs of diabetes according to the World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). Several constraints were placed on the selection of these instances from a larger database held by the National Institute of Diabetes and Digestive and Kidney Diseases.

The *Wisconsin Breast Cancer* data set was originally obtained at the University of Wisconsin Hospitals, Madison, from W. H. Wolberg. The purpose of the data set is to classify a tumor as either benign or malignant based on cell description gathered by microscopic examination.

The problem posed in the *Hepatitis* data set is to predict whether the patients will die or not as a consequence of their hepatitis. The source of this data set is unknown, and it was donated by G. Gong (Carnegie Mellon University).

The *Ionosphere* data set contains radar data collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not (their signals pass through the ionosphere). The task is to decide whether a radar return

is "good" or "bad" as the previous definition. This data set was originally donated by V. Sigillito from the Johns Hopkins University.

The *Sonar* data set is the data set used in [Gorman and Sejnowski 1988] in their study of the classification of sonar signals using neural networks. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. This data set was donated by T. J. Sejnowski, and was developed in collaboration with R. P. Gorman.

As a brief reference, several results found in the literature for the benchmark data sets used in this work are shown in table A.2. There exist many references in the literature that use these data sets in their experiments. Our purpose was to include in this table those ones with a similar experimental setting to that performed in this work. In some cases, however, the experimental settings cannot be considered as very similar[1]. For the sake of completeness, they are also included. Results for Multi-Layer Perceptrons (MLPs) were obtained with sigmoidal hidden units trained with Back-Propagation (BP). Support Vector Machines (SVMs) results were obtained with Gaussian kernels. DistAl is a constructive neural network learning algorithm specific for classification [Yang and Honavar 1998]. The key idea behind DistAl is to add hyper-spherical hidden units based on a greedy strategy which ensures that the new hidden unit correctly classifies a maximal subset of training patterns belonging to the same class. Correctly classified examples can then be discarded when a new unit is to be added.

Additionally, table A.3 shows several results found in the literature after the application of an FS procedure on these data sets. In this case the comparison is even more difficult than with the whole set of features, since the search procedure and the evaluation criterion may also vary among different methods. Wrappers, in addition, may use different ML learning schemes. Unfortunately, we did not found more references (for wrapper approaches and with a similar experimental setting to that performed in this work) than the results in [Yang and Honavar 1998] showed in the table. The search procedure was a genetic algorithm where the fitness function for a given feature subset was computed as the mean percentage of correctly classified patterns on the test sets of a 10-fold CV trained with DistAl. However, looking at the good results of DistAl with the whole set of features when compared with MLPs (see table A.2) and the good behavior of genetic algorithms for FS [Kudo and Sklansky 2000], we consider the results in table A.3 as a useful reference.

[1]Results for SVMs with the *Hepatitis* data set were obtained omitting patterns with missing values in [Anguita et al. 2000] and discretizing continuous attributes in [Huang et al. 2003].

Data Set	ML Alg.	Test	Sampling	Source
Pima Indians	MLP+BP	76.4%	12-fold CV	[Michie et al. 1994]
Diabetes	DistAl	76.3%	10-fold CV	[Yang and Honavar 1998]
	SVM	75.9%	50 splits TVT	[Vincent and Bengio 2002]
Wisconsin	MLP+BP	96.7%	10-fold CV	[Michie et al. 1994]
Breast Cancer	DistAl	97.8%	10-fold CV	[Yang and Honavar 1998]
	SVM	96.6%	50 splits TVT	[Vincent and Bengio 2002]
Hepatitis	MLP+BP	82.1%	10-fold CV	[Michie et al. 1994]
	DistAl	84.7%	10-fold CV	[Yang and Honavar 1998]
	SVM	85.0%	1,000 Bootstrap	[Anguita et al. 2000]
	SVM	85.8%	10-fold CV	[Huang et al. 2003]
Ionosphere	MLP+BP	90.3%	10-fold CV	[Opitz and Maclin 1999]
	DistAl	94.3%	10-fold CV	[Yang and Honavar 1998]
	SVM	93.4%	1,000 Bootstrap	[Anguita et al. 2000]
	SVM	94.2%	50 splits TVT	[Vincent and Bengio 2002]
Sonar	MLP+BP	83.4%	10-fold CV	[Opitz and Maclin 1999]
	DistAl	83.0%	10-fold CV	[Yang and Honavar 1998]
	SVM	87.4%	1,000 Bootstrap	[Anguita et al. 2000]
	SVM	79.4%	50 splits TVT	[Vincent and Bengio 2002]

Table A.2: Several results found in the literature for the benchmark data sets used in this work with the whole set of features. Cross-Validation is indicated as CV. TVT means "Training/Validation/Test" (the validation set was used to tune the parameters). Column 'ML Alg.' indicates the ML algorithm used.

Data Set	Search	ML Alg.	Test	NVar	Sampling
Pima Indians Diabetes	Genetic	DistAl	76.8%	2	10-fold CV
Wisconsin Breast Cancer	Genetic	DistAl	98.6%	8	10-fold CV
Hepatitis	Genetic	DistAl	88.7%	10	10-fold CV
Ionosphere	Genetic	DistAl	96.0%	13	10-fold CV
Sonar	Genetic	DistAl	85.5%	28	10-fold CV

Table A.3: Results in [Yang and Honavar 1998] for the benchmark data sets used in this work after the application of a wrapper FS procedure. Column 'ML Alg.' indicates the ML algorithm used. The final number of variables selected can be seen in column 'NVar'.

A.3 Other Real-World Data Sets

A.3.1 Word Sense Disambiguation

For the Word Sense Disambiguation (WSD) data set we used a part of the English SensEval-2 corpus[2], consisting of a set of annotated examples for 4 words (one

[2] A complete information about the SensEval initiative, including the corpus, can be found at `http://www.cs.unt.edu/~rada/senseval`.

Word	PoS	NTraining	NTest	NSenses	NFeatures
bar	noun	249	65	10	3,222
begin	verb	667	170	9	6,144
natural	adjective	196	53	9	2,777
train	verb	153	35	9	1,710

Table A.4: Description of the WSD data set.

adjective, one noun, and two verbs), divided into a training set and a test set for each word. Each example is provided with a context of several sentences around the word to be disambiguated. Each word is treated as an independent problem.

Table A.4 contains information about the concrete words, the number of training and test examples, and the number of senses (classes) per word. It can be observed that the number of training examples is quite small, whereas the number of classes is high. The high polysemy of the words is partly due to the sense repository used for annotating the corpus. The sense definitions were extracted from the WordNet lexico-semantic database [Fellbaum 1998], which is known to be very fine grained. These facts significantly contribute to the difficulty of the data set.

Three kinds of information have been used to describe the examples and to train the classifiers. These features refer to *local* and *topical* contexts, and *domain labels*. Let "$\ldots w_{-3}\ w_{-2}\ w_{-1}\ w\ w_{+1}\ w_{+2}\ w_{+3}\ldots$" be the context of consecutive words around the word w to be disambiguated, and $p_{\pm i}$ ($-3 \leq i \leq 3$) be the part-of-speech tag of word $w_{\pm i}$. Feature patterns referring to local context are the following 13: p_{-3}, p_{-2}, p_{-1}, p_{+1}, p_{+2}, p_{+3}, w_{-2}, w_{-1}, w_{+1}, w_{+2}, (w_{-2}, w_{-1}), (w_{-1}, w_{+1}), and (w_{+1}, w_{+2}), where the last three correspond to collocations of two consecutive words. The topical context is formed by $\{c_1, \ldots, c_m\}$, which stand for the unordered set of open class words appearing in a medium-size 21-word window centered around the target word. This basic set of features has been enriched by adding semantic information in the form of domain labels. These domain labels are computed during a preprocessing step using the 164 domain labels linked to the nominal part of WordNet 1.6 [Magnini and Cavaglia 2000]. See [Escudero et al. 2001] for details about the preprocessing of the data set and about the attribute extraction.

Table A.4 also shows the number of binary features per data set. Note that the number of actual features is much higher (over ten times) than the number of training examples for each word.

A.3.2 Text Categorization

We used the publicly available Reuters-21578 collection of documents[3], which can be considered the most important benchmark corpus for the Text Categorization

[3]The Reuters-21578 collection and other variants are freely available from the following Web site: `http://www.daviddlewis.com/resources/testcollections`.

earn	acq	money	grain	crude	trade	interest	wheat	ship	corn	None
2,877	1,650	538	433	389	369	347	212	197	181	3,113
1,087	719	179	149	189	117	131	71	89	56	754

Table A.5: Number of examples for the 10 most frequent categories in the TC problem for the training set (first row) and test set (second row).

(TC) task. This corpus contains 12,902 documents of an average length of about 200 words, and it is divided (according to the "ModApte" split) into a training set of 9,603 examples and a test set of 3,299 examples. The corpus is labeled using 118 different categories and has a ratio of 1.2 categories per document. However, the frequency distribution of these categories is very extreme (the 10 most frequent categories covers 75% of the training corpus, and there are 31 categories with only one or two examples). For that reason, we have considered, as in many other works, only the 10 most frequent categories of the corpus. In this way our training corpus contains 3,113 documents with no category and a ratio of 1.11 categories per document in the rest. Table A.5 shows the number of examples for every category.

Regarding the representation of the documents, we have used the simple *bag of words* model, in which each feature corresponds to a single word, and all features are binary valued indicating the presence or absence of the words in the documents. We discarded using more complex document representations since the main goal of this paper is not to achieve the best results on the TC task, but to make comparisons among several models, and because a quite limited utility has been observed by considering these extensions. The attributes have been filtered out by selecting the 50 most relevant for each of the ten classes and merging them all in a unique feature set, containing 387 features. The relevance measure used for ranking attributes is the RLM entropy-based distance function [López de Mántaras 1991].

Regarding the evaluation measures, note that TC is a multiclass multilabel classification problem, since each document may be assigned a set of categories (which may be empty). Thus, one may think that a *yes/no* decision must be taken for each pair (document, category), in order to assign categories to the documents. The most standard way of evaluating TC systems is in terms of *precision* (P), *recall* (R), and a combination of both (e.g., the F_1 measure). Precision is defined as the ratio between the number of correctly assigned categories and the total number of categories assigned by the system. Recall is defined as the ratio between the number of correctly assigned categories and the total number of real categories assigned to examples. The F_1 measure is the harmonic mean of precision and recall: $F_1(P,R) = 2PR/(P+R)$.

Appendix B

Related Publications

A list of up-to-date author's publications related to the contents of this thesis follows below:

1. International journals:

   - [Romero and Alquézar 2004] Romero, E. and Alquézar, R. (2004). A Sequential Algorithm for Feed-forward Neural Networks with Optimal Coefficients and Interacting Frequencies. Submitted.
   - [Romero and Sopena 2004] Romero, E. and Sopena, J. M. (2004). Critical Decision Points for Feature Selection with Multi-Layer Perceptrons. Submitted.
   - [Romero et al. 2004b] Romero, E., Màrquez, L., and Carreras, X. (2004). Margin Maximization with Feed-forward Neural Networks: A Comparative Study with SVM and AdaBoost. *Neurocomputing*, 57:313–344.
   - [Selva et al. 2003] Selva, A., Romero, E., Sopena, J. M., Mijares, T., Solans, R., Labrador, M., and Vilardell, M. (2003). Reply to Linder et al. (2003). *Arthritis and Rheumatism*, 48(4):1169–1170.
   - [Selva et al. 2002] Selva, A., Mijares, T., Solans, R., Labrador, M., Romero, E., Sopena, J. M., and Vilardell, M. (2002). The Neural Network as a Predictor of Cancer in Patients with Inflammatory Myopathies. *Arthritis and Rheumatism*, 46(9):2547–2548.

2. International conferences:

   - [Romero et al. 2004a] Romero, E., Carreras, X., and Màrquez, L. (2004). Exploiting Diversity of Margin-based Classifiers. In *International Joint Conference on Neural Networks*, volume 1, pages 419–424.

- [Romero et al. 2003] Romero, E., Sopena, J. M., Navarrete, G., and Alquézar, R. (2003). Feature Selection Forcing Overtraining May Help to Improve Performance. In *International Joint Conference on Neural Networks*, volume 3, pages 2181–2186.
- [Romero and Alquézar 2002a] Romero, E. and Alquézar, R. (2002). A New Incremental Method for Function Approximation using Feed-forward Neural Networks. In *International Joint Conference on Neural Networks*, volume 2, pages 1968–1973.
- [Romero and Alquézar 2002b] Romero, E. and Alquézar, R. (2002). Maximizing the Margin with Feed-forward Neural Networks. In *International Joint Conference on Neural Networks*, volume 1, pages 743–748.
- [Sopena et al. 1999a] Sopena, J. M., Romero, E., and Alquézar, R. (1999). Neural Networks with Periodic and Monotonic Activation Functions: A Comparative Study in Classification Problems. In *9th International Conference on Artificial Neural Networks*, volume 1, pages 323–328.

3. National journals:

- [Sopena and Romero 2004] Sopena, J. M. and Romero, E. (2004). Redes Neuronales y Métodos Estadísticos Clásicos en el Diagnóstico Médico: La Importancia de las Variables Irrelevantes (in Spanish). *Medicina Clínica*, 122(9):336–338. Editorial.
- [Armadans et al., 2003] Armadans, I., Pol, E., Sopena, J. M., and Romero, E. (2003). Actividad de Ocio Turístico y Personas Mayores: Análisis de Diferencias Psicosociales entre "Viajeros" y "No Viajeros" (in Spanish). *Encuentros en Psicología Social*, 1(3):33–39.

4. National conferences:

- [Mijares et al. 2001] Mijares, T., Selva, A., Romero, E., Sopena, J. M., Solans, R., Labrador, M., Bosch, J. A., and Vilardell, M. (2001). Predicción de Neoplasia y Muerte en Pacientes con Miopatía Inflamatoria Idiopática mediante Redes Neuronales (in Spanish). In *9è Congrés Català-Balear de Medicina Interna*, pages 60–60.
- [Sopena et al. 1999b] Sopena, J. M., Sanz, E., Ramos, P., Romero, E., and Alquézar, R. (1999). ¿Puede el Aleteo de una Mariposa Causar que una Pareja se Divorcie? Un Estudio del Riesgo de Divorcio mediante Redes Neuronales y Selección de Atributos (in Spanish). In *Segon Congrés Català d'Intel·ligència Artificial*, pages 86–92.

Bibliography

Achieser, N. I. (1956). *Theory of Approximation*. Frederick Ungar Pub. Co., New York.

Alexandridis, A., Sarimveis, H., and Bafas, G. (2003). A New Algorithm for Online Structure and Parameter Adaptation of RBF Networks. *Neural Networks*, 16(7):1003–1017.

András, P. (2002). The Equivalence of Support Vector Machine and Regularization Neural Networks. *Neural Processing Letters*, 15(2):97–104.

Anguita, D., Boni, A., and Ridella, S. (2000). Evaluating the Generalization Ability of Support Vector Machines Through the Bootstrap. *Neural Processing Letters*, 11(1):51–58.

Anthony, M. and Bartlett, P. L. (1999). *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, UK.

Armadans, I., Pol, E., Sopena, J. M., and Romero, E. (2003). Actividad de Ocio Turístico y Personas Mayores: Análisis de Diferencias Psicosociales entre "Viajeros" y "No Viajeros" (in Spanish). *Encuentros en Psicología Social*, 1(3):33–39.

Ash, T. (1989). Dynamic Node Creation in Backpropagation Networks. *Connection Science*, 1(4):365–375.

Attik, M., Bougrain, L., and Alexandre, F. (2004). Optimal Brain Surgeon Variants for Feature Selection. In *International Joint Conference on Neural Networks*, volume 2, pages 1371–1374.

Avnimelech, R. and Intrator, N. (1999). Boosted Mixture of Experts: An Ensemble Learning Scheme. *Neural Computation*, 11(2):483–497.

Azimi-Sadjadi, M. R., Sheedvash, S., and Trujillo, F. O. (1993). Recursive Dynamic Node Creation in Multilayer Neural Networks. *IEEE Transactions on Neural Networks*, 4(2):242–256.

Baesens, B., Viaene, S., Vanthienen, J., and Dedene, G. (2000). Wrapped Feature Selection by means of Guided Neural Network Optimisation. In *International Conference on Pattern Recognition*, volume 2, pages 113–116.

Bahbah, A. G. and Girgis, A. A. (1999). Input Feature Selection for Real-Time Transient Stability Assessment for Artificial Neural Network (ANN) using ANN Sensitivity Analysis. In *IEEE International Conference on Power Industry Computer Applications*, pages 295–300.

Barron, A. R. (1990). Complexity Regularization with Application to Artificial Neural Networks. In Roussas, G., editor, *Nonparametric Functional Estimation and Related Topics*, pages 561–576. Kluwer Academic Publishers.

Barron, A. R. (1993). Universal Approximation Bounds for Superposition of a Sigmoidal Function. *IEEE Transactions on Information Theory*, 39(3):930–945.

Barron, A. R. (1994). Approximation and Estimation Bounds for Artificial Neural Networks. *Machine Learning*, 14(1):115–133.

Barron, A. R. and Barron, R. L. (1988). Statistical Learning Networks: A Unifying View. In *Computing Science and Statistics: 20th Symposium on the Interface*, pages 192–203.

Bartlett, E. B. (1994). Dynamic Node Architecture Learning: An Information Theoretic Approach. *Neural Networks*, 7(1):129–140.

Bartlett, P. L. (1998). The Sample Complexity of Pattern Classification with Neural Networks: The Size of the Weights Is More Important than the Size of the Network. *IEEE Transactions on Information Theory*, 44(2):525–536.

Bartlett, P. L. and Maass, W. (2003). Vapnik-Chervonenkis Dimension of Neural Nets. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 1188–1192. MIT Press, 2nd edition.

Bartlett, P. L. and Williamson, R. C. (1996). The VC-Dimension and Pseudodimension of Two-Layer Neural Networks with Discrete Inputs. *Neural Computation*, 8(3):625–628.

Bastian, A. (1994). An Effective Way to Generate Neural Network Structures for Function Approximation. *Mathware and Soft Computing*, 1(2):139–161.

Battiti, R. (1989). Accelerated Backpropagation Learning: Two Optimization Methods. *Complex Systems*, 3:331–342.

Bauer, E. and Kohavi, R. (1999). An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting and Variants. *Machine Learning*, 36(1-2):105–139.

Bauer, K. W., Alsing, S. G., and Greene, K. A. (2000). Feature Screening using Signal-to-Noise Ratios. *Neurocomputing*, 31(1-4):29–44.

Baum, E. B. and Haussler, D. (1989). What Size Net Gives Valid Generalization? *Neural Computation*, 1(1):151–160.

Belanche, L. (1999). A Study in Function Optimization with the Breeder Genetic Algorithm. Technical Report LSI-99-36-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain.

Belanche, L. (2000). *Heterogeneous Neural Networks: Theory and Applications.* PhD thesis, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain.

Beliczyński, B. (1996a). An Almost Analytical Design of Incremental Discrete Functions Approximation by One-Hidden-Layer Neural Networks. In *World Congress on Neural Networks*, pages 988–991.

Beliczyński, B. (1996b). Incremental Approximation by One-Hidden-Layer Neural Networks: Discrete Functions Rapprochements. In *IEEE International Symposium on Industrial Electronics*, volume 1, pages 392–397.

Beliczyński, B. and Kúrková, V. (1997). Incremental Orthogonal Projection Learning of Radial-Basis-Function Networks. In *Symposium on Methods and Models in Automation and Robotics*, volume 2, pages 717–722.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour.* Princeton University Press, NJ.

Bello, M. G. (1992). Enhanced Training Algorithms, and Integrated Training/Architecture Selection for Multilayer Perceptron Networks. *IEEE Transactions on Neural Networks*, 3(6):864–875.

Belue, L. M. and Bauer, K. W. (1995). Determining Input Features for Multilayer Perceptrons. *Neurocomputing*, 7(2):111–122.

Berberian, S. K. (1961). *Introduction to Hilbert Space.* Oxford University Press, Inc.

Billings, S. A., Korenberg, M. J., and Chen, S. (1988). Identification of Non-linear Output-affine Systems using an Orthogonal Least Squares Algorithm. *International Journal of Systems Science*, 19(8):1559–1568.

Bishop, C. M. (1995a). *Neural Networks for Pattern Recognition.* Oxford University Press Inc., New York.

Bishop, C. M. (1995b). Training with Noise Is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998). GTM: The Generative Topographic Mapping. *Neural Computation*, 10(1):215–234.

Blake, C. L. and Merz, C. J. (1998). UCI Repository of Machine Learning Databases. University of California, Irvine, Department of Information and Computer Science. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Blum, A. L. and Langley, P. (1997). Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, 97(1-2):245–271. Special Issue on Relevance.

Boger, Z. (2003). Artificial Neural Networks Methods for Identification of the Most Relevant Genes from Gene Expression Array Data. In *International Joint Conference on Neural Networks*, volume 4, pages 3095–3100.

Boger, Z. and Guterman, H. (1997). Knowledge Extraction from Artificial Neural Networks Models. In *International Conference on Systems, Man and Cybernetics*, volume 4, pages 3030–3035.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. In *5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152.

Breiman, L. (1996a). Bagging Predictors. *Machine Learning*, 26(2):123–140.

Breiman, L. (1996b). Bias, Variance and Arcing Classifiers. Technical Report 460, Department of Statistics, University of California, Berkeley, CA.

Breiman, L. (1998). Arcing Classifiers. *The Annals of Statistics*, 26(3):801–849.

Brown, M. P. S., Grundy, W. P., Lin, D., Cristianini, N., Sugnet, C. W., Furey, T. S., Ares, M., and Haussler, D. (2000). Knowledge-based Analysis of Microarray Gene Expression Data by using Support Vector Machines. *Proceedings of the National Academy of Sciences of the USA*, 97(1):262–267.

Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.

Caruana, R., Lawrence, S., and Giles, C. L. (2001). Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. In *Advances in Neural Information Processing Systems*, volume 13, pages 402–408. MIT Press.

Castellano, G. and Fanelli, A. M. (2000). Variable Selection using Neural-Networks Models. *Neurocomputing*, 31(1-4):1–13.

Chang, C. C. and Lin, C. J. (2002). LIBSVM: A Library for Support Vector Machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chapelle, O., Haffner, P., and Vapnik, V. N. (1999). Support Vector Machines for Histogram-based Image Classification. *IEEE Transactions on Neural Networks*, 10(5):1055–1064.

Cheang, G. H. L. and Barron, A. R. (1999). Estimation with Two Hidden Layer Neural Nets. In *IEEE International Conference on Neural Networks*, volume 1, pages 375–378.

Chen, S. (1995). Regularised OLS Algorithm with Fast Implementation for Training Multi-Output Radial Basis Function Networks. In *International Conference on Artificial Neural Networks*, pages 290–294.

Chen, S., Billings, S. A., and Luo, W. (1989). Orthogonal Least Squares Methods and their Applications to Non-linear System Identification. *International Journal of Control*, 50(5):1873–1896.

Chen, S., Chng, E. S., and Alkadhimi, K. (1996). Regularized Orthogonal Least Squares Algorithm for Constructing Radial Basis Function Networks. *International Journal of Control*, 64(5):829–837.

Chen, S., Cowan, C. F. N., and Grant, P. M. (1991a). Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*, 2(2):302–309.

Chen, S., Grant, P. M., and Cowan, C. F. N. (1991b). Orthogonal Least Squares Algorithm for Training Multi-Output Radial Basis Function Networks. In *International Conference on Artificial Neural Networks*, pages 336–339.

Chen, S., Grant, P. M., and Cowan, C. F. N. (1992). Orthogonal Least-Squares Algorithm for Training Multioutput Radial Basis Function Networks. *Proceedings of Radar and Signal Processing*, 139(6):378–384.

Chen, S. and Wigger, J. (1995). Fast Orthogonal Least Squares Algorithm for Efficient Subset Model Selection. *IEEE Transactions on Signal Processing*, 43(7):1713–1715.

Chen, S., Wu, Y., and Alkadhimi, K. (1995). A Two-Layer Learning Method for Radial Basis Function Networks using Combined Genetic and Regularised Algorithms. In *International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 245–249.

Chen, S., Wu, Y., and Luk, B. L. (1999). Combined Genetic Algorithm and Regularized Orthogonal Least Squares Learning for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*, 10(5):1239–1243.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (1998). Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61.

Chng, E. S., Chen, S., and Mulgrew, B. (1994). Reducing the Computational Requirement of the Orthogonal Least Squares Algorithm. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 529–532.

Chng, E. S., Chen, S., and Mulgrew, B. (1995). Efficient Computational Schemes for the Orthogonal Least Squares Algorithm. *IEEE Transactions on Signal Processing*, 43(1):373–376.

Chung, F. L. and Lee, T. (1995). Network-Growth Approach to Design of Feedforward Neural Networks. *IEE Proceedings Control Theory and Applications*, 142(5):486–492.

Cibas, T., Soulié, F. F., Gallinari, P., and Raudys, Š. (1994a). Variable Selection with Neural Networks. In *International Conference on Artificial Neural Networks*, volume 2, pages 1464–1469.

Cibas, T., Soulié, F. F., Gallinari, P., and Raudys, Š. (1994b). Variable Selection with Optimal Cell Damage. In *International Conference on Artificial Neural Networks*, volume 1, pages 727–730.

Cibas, T., Soulié, F. F., Gallinari, P., and Raudys, Š. (1996). Variable Selection with Neural Networks. *Neurocomputing*, 12(2-3):223–248.

Cid-Sueiro, J. and Sancho-Gómez, J. L. (2001). Saturated Perceptrons for Maximum Margin and Minimum Misclassification Error. *Neural Processing Letters*, 14(3):217–226.

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Journal of Educational and Psychological Measurement*, 20:37–46.

Cortes, C. and Vapnik, V. N. (1995). Support-Vector Networks. *Machine Learning*, 20(3):273–297.

Courrieu, P. (1993). A Convergent Generator of Neural Networks. *Neural Networks*, 6(6):835–844.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines*. Cambridge University Press, UK.

Darken, C., Donahue, M., Gurvits, L., and Sontag, E. (1993). Rate of Approximation Results Motivated by Robust Neural Network Learning. In *6th Annual ACM Conference on Computational Learning Theory*, pages 303–309.

Daubechies, I. (1992). *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, Philadelphia.

De, R. K., Pal, N. R., and Pal, S. K. (1997). Feature Analysis: Neural Networks and Fuzzy Set Theoretic Approaches. *Pattern Recognition*, 30(10):1579–1590.

Deco, G. and Ebmeyer, J. (1993). Coarse Coding Resource-Allocating Network. *Neural Computation*, 5(1):105–114.

DeVore, R. A. and Temlyakov, V. N. (1996). Some Remarks on Greedy Algorithms. *Advances in Computational Mathematics*, 5(2-3):173–187.

Devroye, L. and Lugosi, G. (1995). Lower Bounds in Pattern Recognition and Learning. *Pattern Recognition*, 28(7):1011–1018.

Diaconis, P. and Freedman, D. (1984). Asymptotics of Graphical Projection Pursuit. *The Annals of Statistics*, 12(3):793–815.

Diaconis, P. and Shahshahani, M. (1984). On Nonlinear Functions of Linear Combinations. *SIAM Journal on Scientific and Statistical Computing*, 5(1):175–191.

Dietterich, T. G. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157.

Dingankar, A. and Sandberg, I. W. (1996). A Note on Error Bounds for Approximation in Inner Product Spaces. *Circuits, Systems and Signal Processing*, 15(4):519–522.

Domingos, P. (2000a). A Unified Bias-Variance Decomposition and its Applications. In *17th International Conference on Machine Learning*, pages 231–238.

Domingos, P. (2000b). A Unified Bias-Variance Decomposition for Zero-One and Squared Loss. In *17th National Conference on Artificial Intelligence AAAI-2000*, pages 564–569.

Draelos, T. and Hush, D. (1996). A Constructive Neural Network Algorithm for Function Approximation. In *IEEE International Conference on Neural Networks*, volume 1, pages 50–55.

Drucker, H., Wu, D., and Vapnik, V. N. (1999). Support Vector Machines for Spam Categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* John Wiley, NY.

Dudley, R. M. (1984). A Course on Empirical Processes. In *Lecture Notes in Mathematics*, volume 1097, pages 2–142. Springer-Verlag, Berlin.

Dunkin, N., Shawe-Taylor, J., and Koiran, P. (1997). A New Incremental Learning Technique. In *8th Italian Workshop on Neural Nets*, pages 112–118.

Egmont-Petersen, M., Talmon, J. L., Hasman, A., and Ambergen, A. W. (1998). Assessing the Importance of Features for Multi-Layer Perceptrons. *Neural Networks*, 11(4):623–635.

El-Deredy, W. and Branston, N. M. (1995). Identification of Relevant Features in ^{1}HMR Tumour Spectra using Neural Networks. In *IEEE International Conference on Neural Networks*, volume 1, pages 454–458.

Engelbrecht, A. P. and Cloete, I. (1996). A Sensitivity Analysis Algorithm for Pruning Feedforward Neural Networks. In *IEEE International Conference on Neural Networks*, volume 2, pages 1274–1277.

Escudero, G., Màrquez, L., and Rigau, G. (2001). Using LazyBoosting for Word Sense Disambiguation. In *Senseval Workshop on the Evaluation of WSD Systems*, pages 71–74.

Evgeniou, T., Pontil, M., and Poggio, T. (2000). Regularization Networks and Support Vector Machines. *Advances in Computational Mathematics*, 13(1):1–50.

Faddeeva, V. N. (1959). *Computational Methods of Linear Algebra*. Dover Publications Inc., New York.

Fahlman, S. E. (1988). An Empirical Study of Learning Speed in Back-Propagation Networks. Technical Report CMU-CS-88-162, Carnegie Mellon University.

Fahlman, S. E. and Lebiere, C. (1990). The Cascade-Correlation Learning Architecture. In *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann.

Fang, W. and Lacher, R. C. (1994). Network Complexity and Learning Efficiency of Constructive Learning Algorithms. In *IEEE International Conference on Neural Networks*, volume 1, pages 366–369.

Faragó, A. and Lugosi, G. (1993). Strong Universal Consistency of Neural Networks Classifiers. *IEEE Transactions on Information Theory*, 39(4):1146–1151.

Fellbaum, C., editor (1998). *WordNet. An Electronic Lexical Database*. Language, Speech, and Communication Series, MIT Press.

Fernández, M. and Hernández, C. (1999). Input Selection by Multilayer Trained Networks. In *International Joint Conference on Neural Networks*, volume 3, pages 1834–1839.

Fine, T. L. (1999). *Feedforward Neural Network Methodology*. Springer-Verlag, New York.

Flexer, A. (1996). Statistical Evaluation of Neural Network Experiments: Minimum Requirements and Current Practice. In *13th European Meeting on Cybernetics and System Research*, volume 2, pages 1005–1008.

Flick, T. E., Jones, L. K., Priest, R. G., and Herman, C. (1990). Pattern Classification using Projection Pursuit. *Pattern Recognition*, 23(2):1367–1376.

Freund, Y. (1990). Boosting a Weak Learning Algorithm by Majority. In *3rd Annual Workshop on Computational Learning Theory*, pages 202–216.

Freund, Y. (2001). An Adaptive Version of the Boost by Majority Algorithm. *Machine Learning*, 43(3):293–318.

Freund, Y. and Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. In *13th International Conference on Machine Learning*, pages 148–156.

Freund, Y. and Schapire, R. E. (1997). A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, J. H. (1985). Classification and Multiple Regression Through Projection Pursuit. Technical Report 12, Department of Statistics, Stanford University.

Friedman, J. H. (1997). On Bias, Variance, 0/1-Loss and the Curse-of-Dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77.

Friedman, J. H. and Stuetzle, W. (1981). Projection Pursuit Regression. *Journal of the American Statistical Association*, 76:817–823.

Friedman, J. H. and Stuetzle, W. (1984). Projection Pursuit Density Estimation. *Journal of the American Statistical Association*, 79:599–608.

Friedman, J. H. and Tukey, J. W. (1974). A Projection Pursuit Algorithm for Exploratory Data Analysis. *IEEE Transactions on Computers*, C-23(9):881–889.

Fritzke, B. (1994a). Fast Learning with Incremental RBF Networks. *Neural Processing Letters*, 1(1):2–5.

Fritzke, B. (1994b). Growing Cell Structures - A Self-Organizing Network for Unsupervised and Supervised Learning. *Neural Networks*, 7(9):1441–1460.

Fukunaga, K. and Koontz, W. (1970). Application of Karhunen-Loeve Expansion to Feature Selection and Ordering. *IEEE Transactions on Computers*, C-19(4):311–318.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58.

Girosi, F. (1998). An Equivalence Between Sparse Approximation and Support Vector Machines. *Neural Computation*, 10(6):1455–1480.

Girosi, F., Jones, M., and Poggio, T. (1995). Regularization Theory and Neural Networks Architectures. *Neural Computation*, 7(2):219–269.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. John Hopkins University Press, Baltimore. Third Edition.

Gorman, R. P. and Sejnowski, T. J. (1988). Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets. *Neural Networks*, 1(1):75–89.

Graepel, T., Herbrich, R., and Shawe-Taylor, J. (2000). Generalisation Error Bounds for Sparse Linear Classifiers. In *13th Annual Conference on Computational Learning Theory*, pages 298–303.

Grandvalet, Y. (2000). Anisotropic Noise Injection for Input Variables Relevance Determination. *IEEE Transactions on Neural Networks*, 11(6):1201–1212.

Grandvalet, Y., Canu, S., and Boucheron, S. (1997). Noise Injection: Theoretical Prospects. *Neural Computation*, 9(5):1093–1108.

Guterman, H. (1994). Application of Principal Component Analysis to the Design of Neural Networks. *Neural, Parallel & Scientific Computation*, 2(1):43–54.

Guyon, I. and Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182. Special Issue on Variable and Feature Selection.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. N. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46(1-3):389–422.

Hansen, L. K. and Salamon, P. (1990). Neural Networks Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

Hanson, S. J. (1990). Meiosis Networks. In *Advances in Neural Information Processing Systems*, volume 2, pages 533–541. Morgan Kaufmann.

Hassibi, B. and Stork, D. G. (1993). Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems*, volume 5, pages 164–171. Morgan Kaufmann.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The Elements of Statistical Learning*. Springer-Verlag, New York.

Haussler, D. (1989). Generalizing the PAC Model: Sample Size Bounds from Metric Dimension-based Uniform Convergence Results. In *30th Annual Symposium on Foundations of Computer Science*, pages 40–45.

Haussler, D. (1992). Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications. *Information and Computation*, 100(1):78–150.

Haykin, S. (1994). *Neural Networks. A Comprehensive Foundation*. Prentice Hall International, NJ.

Herbrich, R. (2002). *Learning Kernel Classifiers*. MIT Press.

Herbrich, R., Graepel, T., and Shawe-Taylor, J. (2000). Sparsity vs. Large Margins for Linear Classifiers. In *13th Annual Conference on Computational Learning Theory*, pages 304–308.

Hernández, C. and Fernández, M. (2002). On the Design of Constructive Training Algorithms for Multilayer Feedforward. In *International Joint Conference on Neural Networks*, volume 1, pages 890–895.

Hinton, G. E. (1987). Learning Translation Invariant Recognition in a Massively Parallel Network. In *PARLE: Parallel Architectures and Languages Europe*, volume 1, pages 1–13.

Hirose, Y., Yamashita, K., and Hijiya, S. (1991). Back-Propagation Algorithm Which Varies the Number of Hidden Units. *Neural Networks*, 4(1):61–66.

Hlaváčková, K. and Fischer, M. M. (2000). An Incremental Algorithm for Parallel Training of the Size and the Weights in a Feedforward Neural Network. *Neural Processing Letters*, 11(2):131–138.

Hlaváčková, K. and Sanguinetti, M. (1998). Algorithm of Incremental Approximation using Variation of a Function with Respect to a Subset. In *International ICSC/IFAC Symposium on Neural Computation*, pages 896–899.

Horst, R. and Tuy, H. (1993). *Global Optimization: Deterministic Approaches*. Springer-Verlag, Berlin.

Hsu, C. N., Huang, H. J., and Schuschel, D. (2002). The ANNIGMA-Wrapper Approach to Fast Feature Selection for Neural Nets. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 32(2):207–212.

Hua, S. and Sun, Z. (2001). A Novel Method of Protein Secondary Structure Prediction with High Segment Overlap Measure: Support Vector Approach. *Journal of Molecular Biology*, 308(2):397–407.

Huang, J., Lu, J., and Ling, C. X. (2003). Comparing Naive Bayes, Decision Trees, and SVM with AUC and Accuracy. In *International Conference on Data Mining*, pages 553–556.

Huber, P. J. (1985). Projection Pursuit. *The Annals of Statistics*, 13(2):435–475.

Hwang, J. N., Li, H., Maechler, M., Martin, D., and Schimert, J. (1992a). A Comparison of Projection Pursuit and Neural Network Regression Modelling. In *Advances in Neural Information Processing Systems*, volume 4, pages 1159–1166. Morgan Kaufmann.

Hwang, J. N., Li, H., Maechler, M., Martin, D., and Schimert, J. (1992b). Projection Pursuit Learning Networks for Regression. *Engineering Applications of Artificial Intelligence*, 5(3):193–204.

Hwang, J. N., Li, H., Martin, D., and Schimert, J. (1991). The Learning Parsimony of Projection Pursuit and Back-Propagation Networks. In *25th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 491–495.

Hwang, J. N., Ray, S. R., Maechler, M., Martin, D., and Schimert, J. (1994). Regression Modelling in Back-Propagation and Projection Pursuit Learning. *IEEE Transactions on Neural Networks*, 5(3):342–353.

Hwang, J. N., You, S. S., Lay, S. R., and Jou, I. C. (1996). The Cascade-Correlation Learning: A Projection Pursuit Learning Perspective. *IEEE Transactions on Neural Networks*, 7(2):278–289.

Ide, N. and Véronis, J. (1998). Introduction to the Special Issue on Word Sense Disambiguation: The State of the Art. *Computational Linguistics*, 24(1):1–40.

Intrator, N. (1993). Combining Exploratory Projection Pursuit and Projection Pursuit Regression with Application to Neural Networks. *Neural Computation*, 5(3):443–455.

Intrator, N. (1999). Robust Prediction in Many-Parameters Models: Specific Control of Variance and Bias. In Kay, J. W. and Titterington, D. M., editors, *Statistics and Neural Networks: Advances at the Interface*, pages 97–128. Oxford University Press.

Islam, M. M. and Murase, K. (2001). A New Algorithm to Design Compact Two-hidden-layer Artificial Neural Networks. *Neural Networks*, 14(9):1265–1278.

Ivakhnenko, A. G. (1971). Polynomial Theory of Complex Systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-1(4):364–378.

Ivakhnenko, A. G. and Ivakhnenko, G. A. (1995). The Review of Problems Solvable by Algorithms of the Group Method of Data Handling (GMDH). *Pattern Recognition and Image Analysis*, 5(4):527–535.

Jacobs, R. A. (1989). Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks*, 1(4):295–307.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive Mixture of Local Experts. *Neural Computation*, 3(1):79–87.

Jain, A. and Zongker, D. (1997). Feature Selection: Evaluation, Application, and Small Sample Performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–158.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323.

Jankowski, N. and Kadirkamanathan, V. (1997). Statistical Controls of RBF-like Networks for Classification. In *International Conference on Artificial Neural Networks*, pages 385–390.

Joachims, T. (1999). Making Large-Scale SVM Learning Practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*, pages 169–184. MIT Press.

John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant Features and the Subset Selection Problem. In *11th International Conference on Machine Learning*, pages 121–129.

Jones, L. K. (1987). On a Conjecture of Huber Concerning the Convergence of Projection Pursuit Regression. *The Annals of Statistics*, 15(2):880–882.

Jones, L. K. (1992). A Simple Lemma on Greedy Approximation in Hilbert Space and Convergence Rates for Projection Pursuit Regression and Neural Network Training. *The Annals of Statistics*, 20(1):608–613.

Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical Mixture of Experts and the EM Algorithm. *Neural Computation*, 6(2):181–214.

Jutten, C. and Chentouf, R. (1995). A New Scheme for Incremental Learning. *Neural Processing Letters*, 2(1):1–4.

Kadirkamanathan, V. (1994). A Statistical Inference Based Growth Criterion for the RBF Network. In *IEEE Workshop on Neural Networks for Signal Processing*, pages 12–21.

Kadirkamanathan, V. and Niranjan, M. (1993). A Function Estimation Approach to Sequential Learning with Neural Networks. *Neural Computation*, 5(6):954–975.

Kainen, P. C. (1998). Recent Results and Mathematical Methods for Functional Approximation by Neural Networks. In Karny, M., Warwick, K., and Kúrková, V., editors, *Dealing With Complexity: A Neural Network Approach*, pages 220–237. Springer-Verlag, London.

Kantrowitz, M. (1993). CMU Artificial Intelligence Repository. University of Carnegie Mellon, School of Computer Science. `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/-neural/bench/cmu`.

Kearns, M. J. and Schapire, R. E. (1990). Efficient Distribution-free Learning of Probabilistic Concepts. In *31st Annual Symposium on Foundations of Computer Science*, volume 1, pages 382–391.

Kearns, M. J. and Schapire, R. E. (1994). Efficient Distribution-free Learning of Probabilistic Concepts. *Journal of Computer and System Sciences*, 48(3):464–497.

Khorasani, K. and Weng, W. (1994). Structure Adaptation in Feed-forward Neural Networks. In *IEEE International Conference on Neural Networks*, volume 3, pages 1403–1408.

Kilgarriff, A. and Rosenzweig, J. (2000). English SENSEVAL: Report and Results. In *2nd International Conference on Language Resources and Evaluation*, volume 3, pages 1239–1243.

Kira, K. and Rendell, L. (1992). A Practical Approach to Feature Selection. In *9th International Conference on Machine Learning*, pages 249–256.

Kittler, J. (1978). Feature Set Search Algorithms. In Chen, C. H., editor, *Pattern Recognition and Signal Processing*, pages 41–60. Sijthoff & Noordhoff Int. Pub.

Kittler, J. (1986). Feature Selection and Extraction. In Young, T. and Fu, K. S., editors, *Handbook of Pattern Recognition and Image Processing*, pages 59–83. Academic Press, New York.

Kohavi, R. (1995). A Study of Cross-Validation and Booststrap for Accuracy Estimation and Model Selection. In *International Joint Conference on Artificial Intelligence*, volume 2, pages 1137–1143.

Kohavi, R. and John, G. H. (1997). Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1-2):273–324.

Kohavi, R. and Wolpert, D. H. (1996). Bias Plus Variance Decomposition for Zero-One Loss Functions. In *13th International Conference on Machine Learning*, pages 275–283.

Kohonen, T. (1995). *Self-Organizing Maps*. Springer-Verlag, Berlin.

Koiran, P. (1994). Efficient Learning of Continuous Neural Networks. In *7th Annual ACM Conference on Computational Learning Theory*, pages 348–355.

Kolmogorov, A. N. and Fomin, S. V. (1975). *Elements of the Theory of Functions and Functional Analysis*. Mir, Moscow.

Kong, E. B. and Dietterich, T. G. (1995). Error-Correcting Output Coding Corrects Bias and Variance. In *12th International Conference on Machine Learning*, pages 313–321.

Krogh, A. and Hertz, J. A. (1992). A Simple Weight Decay Can Improve Generalization. In *Advances in Neural Information Processing Systems*, volume 4, pages 950–957. Morgan Kaufmann.

Kudo, M. and Sklansky, J. (2000). Comparison of Algorithms that Select Features for Pattern Classifiers. *Pattern Recognition*, 33(1):25–41.

Kúrková, V. (1998). Incremental Approximation by Neural Networks. In Karny, M., Warwick, K., and Kúrková, V., editors, *Dealing With Complexity: A Neural Network Approach*, pages 177–188. Springer-Verlag, London.

Kúrková, V. and Beliczyński, B. (1995a). An Incremental Learning Algorithm for Gaussian Radial-Basis-Function Approximation. In *International Symposium on Methods and Models in Automation and Robotics*, volume 2, pages 675–680.

Kúrková, V. and Beliczyński, B. (1995b). Incremental Approximation by One-Hidden-Layer Neural Networks. In *International Conference on Artificial Neural Networks*, volume 1, pages 505–510.

Kwok, T. Y. and Yeung, D. Y. (1993). Experimental Analysis of Input Weight Freezing in Constructive Neural Networks. In *IEEE International Conference on Neural Networks*, pages 511–516.

Kwok, T. Y. and Yeung, D. Y. (1995a). Improving the Approximation and Convergence Capabilities of Projection Pursuit Learning. In *International Conference on Artificial Neural Networks*, volume 1, pages 197–202.

Kwok, T. Y. and Yeung, D. Y. (1995b). Improving the Approximation and Convergence Capabilities of Projection Pursuit Learning. *Neural Processing Letters*, 2(3):20–25.

Kwok, T. Y. and Yeung, D. Y. (1996a). Bayesian Regularization in Constructive Neural Networks. In *International Conference on Artificial Neural Networks*, pages 557–562.

Kwok, T. Y. and Yeung, D. Y. (1996b). Use of Bias Term in Projection Pursuit Learning Improves Approximation and Convergence Properties. *IEEE Transactions on Neural Networks*, 7(5):1168–1183.

Kwok, T. Y. and Yeung, D. Y. (1997a). Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. *IEEE Transactions on Neural Networks*, 8(3):630–645.

Kwok, T. Y. and Yeung, D. Y. (1997b). Objective Functions for Training New Hidden Units in Constructive Neural Networks. *IEEE Transactions on Neural Networks*, 8(5):1131–1148.

Lahnajärvi, J. J. T., Lehtokangas, M. I., and Saarinen, J. P. P. (1999). Fixed Cascade Error - A Novel Constructive Neural Network for Structure Learning. In *International Conference on Artificial Neural Networks in Engineering*, pages 25–30.

Lahnajärvi, J. J. T., Lehtokangas, M. I., and Saarinen, J. P. P. (2002). Evaluation of Constructive Neural Networks with Cascaded Architectures. *Neurocomputing*, 48(1-4):573–607.

Lang, K. J. and Witbrock, M. J. (1988). Learning to Tell Two Spirals Apart. In *1988 Connectionist Models Summer School*, pages 52–59,. Morgan Kaufmann.

Lang, S. (1989). *Undergraduate Analysis*. Springer-Verlag, New York.

Lawrence, S., Giles, C. L., and Tsoi, A. C. (1996). What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation. Technical Report UMIACS-TR-96-22, Institute for Advanced Computer Studies, Maryland University.

Lawrence, S., Giles, C. L., and Tsoi, A. C. (1997). Lessons in Neural Network Training: Overfitting May Be Harder than Expected. In *14th National Conference on Artificial Intelligence AAAI-97*, pages 540–545.

Lay, S. R., Hwang, J. N., and You, S. S. (1994). Extensions to Projection Pursuit Learning Networks with Parametric Smoothers. In *IEEE International Conference on Neural Networks*, volume 3, pages 1325–1330.

Le Cun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal Brain Damage. In *Advances in Neural Information Processing Systems*, volume 2, pages 598–605. Morgan Kaufmann.

Lee, H., Mehrotra, K., Mohan, C., and Ranka, S. (1993). Selection Procedures for Redundant Inputs in Neural Networks. In *INNS World Congress on Neural Networks*, volume 1, pages 300–303.

Lee, K.-M. and Street, W. N. (2003). An Adaptive Resource-Allocating Network for Automated Detection, Segmentation, and Classification of Breast Cancer Nuclei Topic Area: Image Processing and Recognition. *IEEE Transactions on Neural Networks*, 14(3):680–687.

Lee, S. and Kil, R. M. (1991). A Gaussian Potential Function Network with Hierarchically Self-Organizing Learning. *Neural Networks*, 4(2):207–224.

Lee, W. S., Bartlett, P. L., and Williamson, R. C. (1996). Efficient Agnostic Learning of Neural Networks with Bounded Fan-in. *IEEE Transactions on Information Theory*, 42(6):2118–2132.

Lee, Y. K. and Ng, H. T. (2002). An Empirical Evaluation of Knowledge Sources and Learning Algorithms for Word Sense Disambiguation. In *7th Conference on Empirical Methods in Natural Language Processing*, pages 41–48.

Leerink, L. R., Giles, C. L., Horne, B. G., and Jabri, M. A. (1995). Learning with Product Units. In *Advances in Neural Information Processing Systems*, volume 7, pages 537–544. MIT Press.

Lehtokangas, M. I. (1999). Modelling with Constructive Backpropagation. *Neural Networks*, 12(4-5):707–716.

Lehtokangas, M. I. (2000). Modified Constructive Backpropagation for Regression. *Neurocomputing*, 35(1-4):113–122.

Leray, P. and Gallinari, P. (1999). Feature Selection with Neural Networks. *Behaviormetrika*, 26(1):145–166.

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function. *Neural Networks*, 6(6):861–867.

Liang, H. and Dai, G. (1998). Improvement of Cascade Correlation Learning Algorithm with an Evolutionary Initialization. *Information Sciences*, 112(1-4):1–6.

Liang, X. and Ma, L. (2004). A Study of Removing Hidden Neurons in Cascade-Correlation Neural Networks. In *International Joint Conference on Neural Networks*, volume 2, pages 1015–1020.

Liao, Y., Fang, S. C., and Nuttle, H. L. W. (2003). Relaxed Conditions for Radial-Basis Function Networks to Be Universal Approximators. *Neural Networks*, 16(7):1019–1028.

Lin, W. M., Yang, C. D., Lin, J. H., and Tsay, M. T. (2001). A Fault Classification Method by RBF Neural Network with OLS Learning Procedure. *IEEE Transactions on Power Delivery*, 16(4):473–477.

Littmann, E. and Ritter, H. (1992a). Cascade LLM Networks. In *International Conference on Artificial Neural Networks*, volume 1, pages 253–257.

Littmann, E. and Ritter, H. (1992b). Cascade Network Architectures. In *International Joint Conference on Neural Networks*, volume 2, pages 398–404.

Liu, G. P., Kadirkamanathan, V., and Billings, S. A. (1996). Stable Sequential Identification of Continuous Nonlinear Dynamical Systems by Growing Radial Basis Function Networks. *International Journal of Control*, 65(1):53–69.

Liu, G. P., Kadirkamanathan, V., and Billings, S. A. (1998). On-line Identification of Nonlinear Systems using Volterra Polynomial Basis Function Neural Networks. *Neural Networks*, 11(9):1645–1657.

Liu, G. P., Kadirkamanathan, V., and Billings, S. A. (1999). Variable Neural Networks for Adaptive Control of Nonlinear Systems. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 29(1):34–43.

Liu, H. and Motoda, H. (1998). *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers.

Liu, H. and Setiono, R. (1996). Feature Selection and Classification: A Probabilistic Wrapper Approach. In *International Conference on Industrial and Engineering Applications on Artificial Intelligence and Expert Systems*, pages 419–424.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and C., W. (2002). Text Categorization using String Kernels. *Journal of Machine Learning Research*, 2:419–444.

López de Mántaras, R. (1991). A Distance-based Attribute Selection Measure for Decision Tree Induction. *Machine Learning*, 6(1):81–92.

Lorentz, G. G. (1966). *Approximation of Functions*. Chelsea Pub. Co., New York.

Lugosi, G. and Pintér, M. (1996). A Data-dependent Skeleton Estimate for Learning. In *9th Annual Conference on Computational Learning Theory*, pages 51–56.

Ma, L. and Khorasani, K. (2000). Input-Side Training in Constructive Neural Networks Based on Error Scaling and Pruning. In *International Joint Conference on Neural Networks*, volume 6, pages 455–460.

Ma, L. and Khorasani, K. (2002). Application of Adaptive Constructive Neural Networks to Image Compression. *IEEE Transactions on Neural Networks*, 13(5):1112–1126.

Ma, L. and Khorasani, K. (2003). A New Strategy for Adaptively Constructing Multilayer Feedforward Neural Networks. *Neurocomputing*, 51:361–385.

Ma, L. and Khorasani, K. (2004). New Training Strategies for Constructive Neural Networks with Application to Regression Problems. *Neural Networks*, 17(4):589–609.

Maass, W. (1995). Vapnik-Chervonenkis Dimension of Neural Nets. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 1000–1003. MIT Press.

MacKay, D. J. C. (1992). Bayesian Interpolation. *Neural Computation*, 4(3):415–447.

Maechler, M., Martin, D., Schimert, J., Csoppenszky, M., and Hwang, J. N. (1990). Projection Pursuit Learning Networks for Regression. In *IEEE International Conference on Tools for Artificial Intelligence*, pages 350–358.

Magnini, B. and Cavaglia, G. (2000). Integrating Subject Field Codes into WordNet. In *2nd International Conference on Language Resources and Evaluation*, pages 1413–1418.

Mallat, S. G. (1998). *A Wavelet Tour of Signal Processing*. Academic Press, New York.

Mallat, S. G. and Zhang, Z. (1993). Matching Pursuits with Time-Frequency Dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415.

Mao, J., Mohiuddin, K., and Jain, A. K. (1994). Parsimonious Network Design and Feature Selection Through Node Pruning. In *International Conference on Pattern Recognition*, volume 2, pages 622–624.

Martin, G. L. and Pittman, J. A. (1991). Recognizing Hand-Printed Letters and Digits using Backpropagation Learning. *Neural Computation*, 3(2):258–267.

McQueen, J. (1967). Some Methods of Classification and Analysis of Multivariate Observations. In *5th Berkeley Symposium on Mathematical Statistics and Probabilities*, pages 281–297.

Messer, K. and Kittler, J. (1998). Choosing an Optimal Neural Network Size to Aid a Search Through a Large Image Database. In *British Machine Vision Conference*, volume 1, pages 235–244.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification.* Ellin Horwood. Results available at `http://www.phys.uni.torun.pl/kmk/projects/datasets-stat.html`.

Mijares, T., Selva, A., Romero, E., Sopena, J. M., Solans, R., Labrador, M., Bosch, J. A., and Vilardell, M. (2001). Predicción de Neoplasia y Muerte en Pacientes con Miopatía Inflamatoria Idiopática mediante Redes Neuronales (in Spanish). In *9è Congrés Català-Balear de Medicina Interna*, pages 60–60.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill, New York.

Mohraz, K. and Protzel, P. (1996). FlexNet: A Flexible Neural Network Construction Algorithm. In *European Symposium on Artificial Neural Networks*, pages 111–116.

Molina, L. C., Belanche, L., and Nebot, A. (2002). Feature Selection Algorithms: A Survey and Experimental Evaluation. In *International Conference on Data Mining*, pages 306–313.

Moody, J. (1991). Note on Generalization, Regularization and Architecture Selection in Nonlinear Learning Systems. In *IEEE Workshop on Neural Networks for Signal Processing*, pages 1–10.

Moody, J. (1992). The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems. In *Advances in Neural Information Processing Systems*, volume 4, pages 847–854. Morgan Kaufmann.

Moody, J. (1994). Prediction Risk and Architecture Selection for Neural Networks. In Cherkassky, V., Friedman, J. H., and Wechsler, H., editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, pages 147–165. Springer-Verlag, London.

Moody, J. and Utans, J. (1992). Principled Architecture Selection for Neural Networks: Application to Corporate Bond Rating Prediction. In *Advances in Neural Information Processing Systems*, volume 4, pages 683–690. Morgan Kaufmann.

Morgan, N. and Bourlard, H. (1990). Generalization and Parameter Estimation in Feedforward Nets: Some Experiments. In *Advances in Neural Information Processing Systems*, volume 2, pages 630–637. Morgan Kaufmann.

Mozer, M. C. and Smolensky, P. (1989). Skeletonization: A Technique for Trimming the Fat from a Network Via Relevance Assessment. In *Advances in Neural Information Processing Systems*, volume 1, pages 107–115. Morgan Kaufmann.

Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization. *Evolutionary Computation*, 1(1):25–49.

Müller, K. R., Mika, S., Rätsch, G., Tsuda, K., and Schölkopf, B. (2001). An Introduction to Kernel-based Learning Algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–202.

Nabhan, T. M. and Zomaya, A. Y. (1994). Toward Generating Neural Network Structures for Function Approximation. *Neural Networks*, 7(1):89–99.

Narendra, P. M. and Fukunaga, K. (1977). A Branch and Bound Algorithm for Feature Subset Selection. *IEEE Transactions on Computers*, C-26(9):917–922.

Nikolaev, N. Y. and Iba, H. (2003). Polynomial Harmonic GMDH Learning Networks for Time Series Modeling. *Neural Networks*, 16(10):1527–1540.

Niyogi, P. and Girosi, F. (1994). On the Relationship Between Generalization Error, Hypothesis Complexity and Sample Complexity for Radial Basis Functions. Technical Report A. I. Memo 1467, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Niyogi, P. and Girosi, F. (1999). Generalization Bounds for Function Approximation from Scattered Noisy Data. *Advances in Computational Mathematics*, 10(1):51–80.

Nowlan, S. J. and Hinton, G. E. (1992). Simplifying Neural Networks by Soft Weight Sharing. *Neural Computation*, 4(4):473–493.

Onnia, V., Tico, M., and Saarinen, J. (2001). Feature Selection Method using Neural Network. In *International Conference on Image Processing*, volume 1, pages 513–516.

Opitz, D. and Maclin, R. (1999). Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, 11:169–198.

Orr, M. J. L. (1995). Regularisation in the Selection of Radial Basis Function Centers. *Neural Computation*, 7(3):606–623.

Ortega, J. M. (1972). *Numerical Analysis: A Second Course*. Society for Industrial and Applied Mathematics, Philadelphia.

Parekh, R., Yang, J., and Honavar, V. (1997). Constructive Neural-Network Learning Algorithms for Multi-Category Real-Valued Pattern Classification. Technical Report TR#97-06, Department of Computer Science, Iowa State University.

Parekh, R., Yang, J., and Honavar, V. (2000). Constructive Neural-Network Learning Algorithms for Pattern Classification. *IEEE Transactions on Neural Networks*, 11(2):436–451.

Parker, R. E. and Tummala, M. (1992). Identification of Volterra Systems with a Polynomial Neural Network. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages 561–564.

Pati, Y. C., Rezaiifar, R., and Krishnaprasad, P. S. (1993). Orthogonal Matching Pursuit: Recursive Function Approximation with Application to Wavelet Decomposition. In *27th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 40–44.

Pedersen, M. W., Hansen, L. K., and Larsen, J. (1996). Pruning with Generalization Based Weight Saliencies: γOBD, γOBS. In *Advances in Neural Information Processing Systems*, volume 8, pages 521–527. MIT Press.

Perrone, M. P. and Cooper, L. N. (1993). When Networks Disagree: Ensemble Methods for Hybrid Neural Networks. In Mammone, R. J., editor, *Neural Networks for Speech and Image Processing*, pages 126–142. Chapman-Hall.

Phatak, D. S. and Koren, I. (1994). Connectivity and Performance Tradeoffs in the Cascade Correlation Learning Architecture. *IEEE Transactions on Neural Networks*, 5(6):930–935.

Platt, J. (1991). A Resource-Allocating Network for Function Interpolation. *Neural Computation*, 3(2):213–225.

Poggio, T. and Girosi, F. (1990). Networks for Approximation and Learning. *Proceedings of the IEEE*, 78(9):1481–1497.

Poggio, T. and Girosi, F. (1998). A Sparse Representation for Function Approximation. *Neural Computation*, 10(6):1445–1454.

Pollard, D. (1984). *Convergence of Stochastic Processes*. Springer-Verlag, Berlin.

Pontil, M. and Verri, A. (1998). Support Vector Machines for 3D Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646.

Prechelt, L. (1994). PROBEN1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe.

Prechelt, L. (1997). Investigation of the CasCor Family of Learning Algorithms. *Neural Networks*, 10(5):885–896.

Prechelt, L. (1998). Automatic Early Stopping using Cross Validation: Quantifying the Criteria. *Neural Networks*, 11(4):761–767.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, UK.

Priddy, K. L., Rogers, S. E., Ruck, D. W., and Tarr, G. L. (1993). Bayesian Selection of Important Features for Feedforward Neural Networks. *Neurocomputing*, 5(2-3):91–103.

Pudil, P., Novovičová, J., and Kittler, J. (1994). Floating Search Methods in Feature Selection. *Pattern Recognition Letters*, 15(11):1119–1125.

Qian, S. and Chen, D. (1992). Signal Approximation via Data-Adaptive Normalized Gaussian Functions and its Applications for Speech Processing. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 141–144.

Qian, S. and Chen, D. (1994). Signal Representation using Adaptive Normalized Gaussian Functions. *Signal Processing*, 36(1):1–11.

Ramaswamy, S., Tamayo, P., Rifkin, R., Mukherjee, S., Yeang, C. H., Angelo, M., Ladd, C., Reich, M., Latulippe, E., Mesirov, J. P., Poggio, T., Gerald, W., Loda, M., Lander, E. S., and Golub, T. R. (2001). Multiclass Cancer Diagnosis using Tumor Gene Expression Signatures. *Proceedings of the National Academy of Sciences of the USA*, 98(26):15149–15154.

Rätsch, G., Mika, S., Schölkopf, B., and Müller, K. R. (2002). Constructing Boosting Algorithms from SVMs: An Application to One-Class Classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1184–1199.

Rätsch, G., Onoda, T., and Müller, K. R. (1999). Regularizing AdaBoost. In *Advances in Neural Information Processing Systems*, volume 11, pages 564–570. MIT Press.

Rätsch, G., Onoda, T., and Müller, K. R. (2001). Soft Margins for AdaBoost. *Machine Learning*, 42(3):287–320.

Raudys, Š. (1998a). Evolution and Generalization of a Single Neuron: I. Single-Layer Perceptron as Seven Statistical Classifiers. *Neural Networks*, 11(2):283–296.

Raudys, Š. (1998b). Evolution and Generalization of a Single Neuron: II. Complexity of Statistical Classifiers and Sample Size Considerations. *Neural Networks*, 11(2):297–313.

Raudys, Š. (2000). How Good Are Support Vector Machines? *Neural Networks*, 13(1):17–19.

Raudys, Š. J. and Jain, A. K. (1991). Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practicioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):252–264.

Raviv, Y. and Intrator, N. (1996). Bootstrapping with Noise: An Effective Regularization Technique. *Connection Science*, 8(3-4):356–372. Special Issue on Combining Estimators.

Reddy, B. D. (1998). *Introductory Functional Analysis with Applications to Boundary Value Problems and Finite Elements*. Springer-Verlag, New York.

Reed, R. (1993). Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, 4(5):740–747.

Richard, M. D. and Lippmann, R. P. (1991). Neural Networks Classifiers Estimate Bayesian a Posteriori Probabilities. *Neural Computation*, 3(4):461–483.

Riedmiller, M. and Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *IEEE International Conference on Neural Networks*, volume 1, pages 586–591.

Ripley, B. D. (1995). Statistical Ideas for Selecting Network Architectures. In Kappen, B. and Gielen, S., editors, *Neural Networks: Artificial Intelligence and Industrial Applications*, pages 183–190. Springer-Verlag, London.

Rivals, I. and Personnaz, L. (2003a). MLPs (Mono-Layer Polynomials ans Multi-Layer Perceptrons) for Nonlinear Modelling. *Journal of Machine Learning Research*, 3:1383–1398. Special Issue on Variable and Feature Selection.

Rivals, I. and Personnaz, L. (2003b). Neural-Network Construction and Selection in Nonlinear Modelling. *IEEE Transactions on Neural Networks*, 14(4):804–819.

Romero, E. and Alquézar, R. (2002a). A New Incremental Method for Function Approximation using Feed-forward Neural Networks. In *International Joint Conference on Neural Networks*, volume 2, pages 1968–1973.

Romero, E. and Alquézar, R. (2002b). Maximizing the Margin with Feed-forward Neural Networks. In *International Joint Conference on Neural Networks*, volume 1, pages 743–748.

Romero, E. and Alquézar, R. (2004). A Sequential Algorithm for Feed-forward Neural Networks with Optimal Coefficients and Interacting Frequencies. Submitted.

Romero, E., Carreras, X., and Màrquez, L. (2004a). Exploiting Diversity of Margin-based Classifiers. In *International Joint Conference on Neural Networks*, volume 1, pages 419–424.

Romero, E., Màrquez, L., and Carreras, X. (2004b). Margin Maximization with Feed-forward Neural Networks: A Comparative Study with SVM and AdaBoost. *Neurocomputing*, 57:313–344.

Romero, E. and Sopena, J. M. (2004). Critical Decision Points for Feature Selection with Multi-Layer Perceptrons. Submitted.

Romero, E., Sopena, J. M., Navarrete, G., and Alquézar, R. (2003). Feature Selection Forcing Overtraining May Help to Improve Performance. In *International Joint Conference on Neural Networks*, volume 3, pages 2181–2186.

Roosen, C. B. and Hastie, T. J. (1994). Automatic Smoothing Spline Projection Pursuit. *Journal of Computational and Graphical Statistics*, 3(3):235–248.

Rosipal, R., Koska, M., and Farkaš, I. (1998). Prediction of Chaotic Time-Series with a Resource-Allocating RBF Network. *Neural Processing Letters*, 7(3):185–197.

Ruck, D. W., Rogers, S. K., and Kabrisky, M. (1990a). Feature Selection using a Multilayer Perceptron. *Journal of Neural Network Computing*, 2(2):40–48.

Ruck, D. W., Rogers, S. K., Kabrisky, M., Oxley, M. E., and Suter, B. W. (1990b). The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function. *IEEE Transactions on Neural Networks*, 1(4):296–298.

Rudin, W. (1987). *Real and Complex Analysis*. McGraw-Hill, New York.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition (vol. 1)*, pages 318–362. MIT Press.

Saha, A., Wu, C. L., and Tang, D. S. (1993). Approximation, Dimension Reduction, and Nonconvex Optimization using Linear Superpositions of Gaussians. *IEEE Transactions on Computers*, 42(10):1222–1232.

Salmerón, M., Ortega, J., Puntonet, C. G., and Prieto, A. (2001). Improved RAN Sequential Prediction using Orthogonal Techniques. *Neurocomputing*, 41(1-4):153–172.

Sanger, T. D. (1991a). A Tree-Structured Adaptive Network for Function Approximation in High-Dimensional Spaces. *IEEE Transactions on Neural Networks*, 2(2):285–293.

Sanger, T. D. (1991b). A Tree-Structured Algorithm for Reducing Computation in Networks with Separable Basis Functions. *Neural Computation*, 3(1):67–78.

Sano, H., Nada, A., Iwahori, Y., and Ishii, N. (1993). A Method for Analyzing Information Represented in Neural Networks. In *International Joint Conference on Neural Networks*, volume 3, pages 2719–2722.

Sargent, D. J. (2001). Comparison of Artificial Neural Networks with Other Statistical Approaches: Results from Medical Data Sets. *Cancer*, 91(8):1636–1642.

Scarselli, F. and Tsoi, A. C. (1998). Universal Approximation using Feedforward Neural Networks: A Survey of Some Existing Methods and Some New Results. *Neural Networks*, 11(1):15–37.

Schapire, R. (1990). The Strength of Weak Learnability. *Machine Learning*, 5(2):197–227.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. (1998). Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. *The Annals of Statistics*, 26(5):1651–1686.

Schapire, R. E. and Singer, Y. (1999). Improved Boosting Algorithms using Confidence-Rated Predictions. *Machine Learning*, 37(3):297–336.

Schölkopf, B., Burges, C., and Smola, A. J., editors (1999). *Advances in Kernel Methods - Support Vector Learning*. MIT Press.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels*. MIT Press.

Schölkopf, B., Sung, K. K., Burges, C. J. C., Girosi, F., Niyogi, P., Poggio, T., and Vapnik, V. (1997). Comparing Support Vector Machines with Gaussian Kernels to Radial Basis Function Classifiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765.

Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1–47.

Selva, A., Mijares, T., Solans, R., Labrador, M., Romero, E., Sopena, J. M., and Vilardell, M. (2002). The Neural Network as a Predictor of Cancer in Patients with Inflammatory Myopathies. *Arthritis and Rheumatism*, 46(9):2547–2548.

Selva, A., Romero, E., Sopena, J. M., Mijares, T., Solans, R., Labrador, M., and Vilardell, M. (2003). Reply to Linder et al. (2003). *Arthritis and Rheumatism*, 48(4):1169–1170.

Setiono, R. and Hui, L. C. K. (1995). Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm. *IEEE Transactions on Neural Networks*, 6(1):273–277.

Setiono, R. and Liu, H. (1997). Neural-Network Feature Selector. *IEEE Transactions on Neural Networks*, 8(3):654–662.

Shertinsky, A. and Picard, R. W. (1996). On the Efficiency of the Orthogonal Least Squares Training Algorithm Method for Radial Basis Function Networks. *IEEE Transactions on Neural Networks*, 7(1):195–200.

Shin, Y. and Ghosh, J. (1995). Ridge Polynomial Networks. *IEEE Transactions on Neural Networks*, 6(3):610–622.

Siedlecki, W. and Sklansky, J. (1988). On Automatic Feature Selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(2):197–220.

Sietsma, J. and Dow, R. J. F. (1991). Creating Artificial Neural Networks that Generalize. *Neural Networks*, 4(1):67–79.

Sindhwani, V., Rakshit, S., Deodhare, D., Erdogmus, D., Principe, J. C., and Niyogi, P. (2004). Feature Selection in MLPs and SVMs Based on Maximum Output Information. *IEEE Transactions on Neural Networks*, 15(4):937–948.

Singer, Y. (2000). Leveraged Vector Machines. In *Advances in Neural Information Processing Systems*, volume 12, pages 610–617. MIT Press.

Sjøgaard, S. (1992). Generalization in Cascade-Correlation Networks. In *IEEE-SP Workshop on Neural Networks for Signal Processing*, pages 59–68.

Smola, A. J. and Bartlett, P. (2001). Sparse Greedy Gaussian Process Regression. In *Advances in Neural Information Processing Systems*, volume 13, pages 619–625. MIT Press.

Smola, A. J., Bartlett, P., Schölkopf, B., and Schuurmans, D., editors (2000). *Advances in Large Margin Classifiers*. MIT Press.

Smola, A. J. and Schölkopf, B. (2000). Sparse Greedy Matrix Approximation for Machine Learning. In *International Conference on Machine Learning*, pages 911–918.

Smola, A. J., Schölkopf, B., and Müller, K. R. (1998). The Connection between Regularization Operators and Support Vector Kernels. *Neural Networks*, 11(4):637–650.

Smotroff, I. G., Friedman, D. H., and Connolly, D. (1991). Self Organizing Modular Neural Networks. In *International Joint Conference on Neural Networks*, volume 2, pages 187–192.

Sontag, E. D. (1998). VC Dimension of Neural Networks. In Bishop, C., editor, *Neural Networks and Machine Learning*, pages 69–95. Springer-Verlag, Berlin.

Sopena, J. M. and Romero, E. (2004). Redes Neuronales y Métodos Estadísticos Clásicos en el Diagnóstico Médico: La Importancia de las Variables Irrelevantes (in Spanish). *Medicina Clínica*, 122(9):336–338. Editorial.

Sopena, J. M., Romero, E., and Alquézar, R. (1999a). Neural Networks with Periodic and Monotonic Activation Functions: A Comparative Study in Classification Problems. In *9th International Conference on Artificial Neural Networks*, volume 1, pages 323–328.

Sopena, J. M., Sanz, E., Ramos, P., Romero, E., and Alquézar, R. (1999b). ¿Puede el Aleteo de una Mariposa Causar que una Pareja se Divorcie? Un Estudio del Riesgo de Divorcio mediante Redes Neuronales y Selección de Atributos (in Spanish). In *Segon Congrés Català d'Intel·ligència Artificial*, pages 86–92.

Stahlberger, A. and Riedmiller, M. (1997). Fast Network Pruning and Feature Extraction using the Unit-OBS Algorithm. In *Advances in Neural Information Processing Systems*, volume 9, pages 655–661. MIT Press.

Steppe, J. M. and Bauer, K. W. (1996). Improved Feature Screening in Feedforward Neural Networks. *Neurocomputing*, 13(1):47–58.

Steppe, J. M., Bauer, K. W., and Rogers, S. K. (1996). Integrated Feature and Architecture Selection. *IEEE Transactions on Neural Networks*, 7(4):1007–1013.

Stone, M. (1974). Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society*, B36:111–147.

Suykens, J. A. K. and Vandewalle, J. (1999a). Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300.

Suykens, J. A. K. and Vandewalle, J. (1999b). Training Multilayer Perceptron Classifiers Based on a Modified Support Vector Method. *IEEE Transactions on Neural Networks*, 10(4):907–911.

Suzuki, S. (1998). Constructive Function-Approximation by Three-Layer Artificial Neural Networks. *Neural Networks*, 11(6):1049–1058.

Tenorio, M. F. and Lee, W. T. (1990). Self-Organizing Network for Optimum Supervised Learning. *IEEE Transactions on Neural Networks*, 1(1):100–110.

Tetko, I. V., Tanchuk, V. Y., , and Luik, A. I. (1994). Simple Heuristic Methods for Input Parameters Estimation in Neural Networks. In *IEEE International Conference on Neural Networks*, volume 1, pages 376–380.

Tetko, I. V., Villa, A. E. P., , and Livingstone, D. J. (1996). Neural Network Studies 2. Variable Selection. *Journal of Chemical Information and Computer Sciences*, 36(4):794–803.

Tibshirani, R. (1996). Bias, Variance and Prediction Error for Classification Rules. Technical report, Department of Statistics, University of Toronto, Toronto, Canada.

Tikhonov, A. and Arsenin, V. Y. (1977). *Solutions of Ill-Posed Problems*. W.H. Winston, Washington D.C.

Tollenaere, T. (1990). SuperSAB: Fast Adaptive Back Propagation with Good Scaling Properties. *Neural Networks*, 3(5):561–573.

Treadgold, N. K. and Gedeon, T. D. (1997a). A Cascade Network Algorithm Employing Progressive RPROP. In *International Work-Conference on Artificial and Natural Neural Networks*, pages 733–742.

Treadgold, N. K. and Gedeon, T. D. (1997b). Extending CasPer: A Regression Survey. In *International Conference on Neural Information Processing and Intelligent Information Systems*, volume 1, pages 310–313.

Treadgold, N. K. and Gedeon, T. D. (1998). Exploring Architecture Variations in Constructive Cascade Networks. In *International Joint Conference on Neural Networks*, volume 1, pages 4–9.

Treadgold, N. K. and Gedeon, T. D. (1999a). A Constructive Cascade Network with Adaptive Regularisation. In *International Work-Conference on Artificial and Natural Neural Networks*, volume 2, pages 40–49.

Treadgold, N. K. and Gedeon, T. D. (1999b). Exploring Constructive Cascade Networks. *IEEE Transactions on Neural Networks*, 10(6):1335–1350.

Van de Laar, P. and Heskes, T. (2000). Input Selection Based on an Ensemble. *Neurocomputing*, 34(1-4):227–238.

Van de Laar, P., Heskes, T., and Gielen, S. (1999). Partial Retraining: A New Approach to Input Relevance Determination. *International Journal of Neural Systems*, 9(1):75–85.

Van Gestel, T., Suykens, J. A. K. Baesens, B., Viane, S., Vanthienen, J., Dedene, G., De Moor, B., and Vandewalle, J. (2004). Benchmarking Least Squares Support Vector Machine Classifiers. *Machine Learning*, 54(1):5–32.

Vapnik, V. N. (1982). *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, NY.

Vapnik, V. N. (1992). Principles of Risk Minimization for Learning Theory. In *Advances in Neural Information Processing Systems*, volume 4, pages 831–838. Morgan Kaufmann.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory.* Springer-Verlag, NY.

Vapnik, V. N. (1998a). *Statistical Learning Theory.* John Wiley & Sons, NY.

Vapnik, V. N. (1998b). The Support Vector Method of Function Estimation. In Bishop, C., editor, *Neural Networks and Machine Learning*, pages 239–268. Springer-Verlag, Berlin.

Vapnik, V. N. (1999). An Overview of Statistical Learning Theory. *IEEE Transactions on Neural Networks*, 10(5):988–999.

Vapnik, V. N. and Chervonenkis, A. J. (1971). On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. *Theory of Probability and Its Applications*, 16(2):264–280.

Verikas, A. and Bacauskiene, M. (2002). Feature Selection with Neural Networks. *Pattern Recognition Letters*, 23(11):1323–1335.

Verkooijen, W. and Daniels, H. (1994). Connectionist Projection Pursuit Regression. *Computational Economics*, 7(3):155–161.

Vincent, P. and Bengio, Y. (2000). A Neural Support Vector Architecture with Adaptive Kernels. In *International Joint Conference on Neural Networks*, volume 5, pages 187–192.

Vincent, P. and Bengio, Y. (2002). Kernel Matching Pursuit. *Machine Learning*, 48(1-3):165–187. Special Issue on New Methods for Model Combination and Model Selection.

Vinod, V. V. and Ghose, S. (1996). Growing Nonuniform Feedforward Networks for Continuous Mappings. *Neurocomputing*, 10(1):55–69.

Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., and Alkon, D. L. (1988). Accelerating the Convergence of the Back-Propagation Method. *Biological Cybernetics*, 59:257–263.

Vyšniauskas, V., Groen, F. C. A., Ben, J. A., and Kröse, B. J. A. (1995). Orthogonal Incremental Learning of a Feedforward Network. In *International Conference on Artificial Neural Networks*, volume 1, pages 311–316.

Wang, Z., Di Massimo, C., Tham, M. T., and Morris, J. (1994). A Procedure for Determining the Topology of Multilayer Feedforward Neural Networks. *Neural Networks*, 7(2):291–300.

Weng, W. and Khorasani, K. (1996). An Adaptive Structure Neural Networks with Applications to EEG Automatic Seizure Detection. *Neural Networks*, 9(7):1223–1240.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Boston, MA.

Wikel, J. H. and Dow, E. R. (1993). The Use of Neural Networks for Variable Selection in QSAR. *Bioorganic & Medicinal Chemistry Letters*, 3(4):645–651.

Wynne-Jones, M. (1992). Node Splitting: A Constructive Algorithm for Feed-forward Neural Networks. In *Advances in Neural Information Processing Systems*, volume 4, pages 1072–1079. Morgan Kaufmann.

Xu, J., Zhang, X., and Li, Y. (2001). Kernel MSE Algorithm: A Unified Framework for KFD, LS-SVM and KRR. In *International Joint Conference on Neural Networks*, volume 2, pages 1486–1491.

Yang, J. and Honavar, V. (1998). Feature Subset Selection using a Genetic Algorithm. In Liu, H. and Motoda, H., editors, *Feature Extraction, Construction and Selection: A Data Mining Perspective*, pages 117–136. Kluwer Academic Publishers.

Yeung, D. Y. (1991). A Neural Network Approach to Constructive Induction. In *8th International Workshop on Machine Learning*, pages 228–232.

Yeung, D. Y. (1993). Constructive Neural Networks as Estimators of Bayesian Discriminant Functions. *Pattern Recognition*, 26(1):189–204.

Yingwei, L., Sundararajan, N., and Saratchandran, P. (1997a). A Sequential Learning Scheme for Function Approximation using Minimal Radial Basis Function Neural Networks. *Neural Computation*, 9(2):461–478.

Yingwei, L., Sundararajan, N., and Saratchandran, P. (1997b). Identification of Time-varying Nonlinear Systems using Minimal Radial Basis Function Neural Networks. *IEE Proceedings on Control Theory and Applications*, 144(2):202–208.

Yosida, K. (1965). *Functional Analysis*. Springer-Verlag, New York.

You, S. S., Hwang, J. N., Jou, I. C., and Lay, S. R. (1994). A New Cascaded Projection Pursuit Network for Nonlinear Regression. In *International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 585–588.

Young, R. M. (1980). *An Introduction to Nonharmonic Fourier Series*. Academic Press, New York.

Zhang, B. T. (1994). An Incremental Learning Algorithm that Optimizes Network Size and Sample Size in One Trial. In *IEEE International Conference on Neural Networks*, volume 1, pages 215–220.

Zhang, J. and Morris, A. J. (1998). A Sequential Learning Approach for Single Hidden Layer Neural Networks. *Neural Networks*, 11(1):65–80.

Zhang, T. (2002). A General Greedy Approximation Algorithm with Applications. In *Advances in Neural Information Processing Systems*, volume 14, pages 1065–1072. MIT Press.

Zhao, Y. and Atkeson, C. G. (1991). Projection Pursuit Learning. In *International Joint Conference on Neural Networks*, volume 1, pages 869–874.

Zhao, Y. and Atkeson, C. G. (1996). Implementing Projection Pursuit Learning. *IEEE Transactions on Neural Networks*, 7(2):362–372.

Zurada, J. M., Malinowski, A., and Usui, S. (1997). Perturbation Method for Deleting Redundant Inputs of Perceptron Networks. *Neurocomputing*, 14(2):177–193.