Neural Networks in Economics: Background, Applications and New Developments

Ralf Herbrich
Technical University of Berlin
Statistics and Mathematics
Franklinstr. 28/29, 10587 Berlin

Max Keilbach
Technical University of Berlin
Econometrics Group
Franklinstr. 28/29, 10587 Berlin

Thore Graepel
Technical University of Berlin
Neural Information Processing
Franklinstr. 28/29, 10587 Berlin

Peter Bollmann–Sdorra
Technical University of Berlin
Statistics and Mathematics
Franklinstr. 28/29, 10587 Berlin

Klaus Obermayer
Technical University of Berlin
Neural Information Processing
Franklinstr. 28/29, 10587 Berlin

Department of Computer Science
Technical University of Berlin, Franklinstr. 28/29, 10587 Berlin
(for more infomation look at http://stat.cs.tu-berlin.de/publications)

Abstract

Neural Networks were developed in the sixties as devices for classification and regression. The approach was originally inspired from Neuroscience. Its attractiveness lies in the ability to “learn”, i.e. to generalize to as yet unseen observations. One aim of this paper is to give an introduction to the technique of Neural Networks and an overview of the most popular architectures. We start from statistical learning theory to introduce the basics of learning. Then, we give an overview of the general principles of neural networks and of their use in the field of Economics. A second purpose is to introduce a recently developed Neural Network Learning technique, so called Support Vector Network Learning, which is an application of ideas from statistical learning theory. This approach has shown very promising results on problems with a limited amount of training examples. Moreover, utilizing a technique that is known as the kernel trick, Support Vector Networks can easily be adapted to nonlinear models. Finally, we present an economic application of this approach from the field of preference learning.

1 Introduction

Neural Networks – originally inspired from Neuroscience – provide powerful models for statistical data analysis. Their most prominent feature is their ability to “learn” dependencies based on a finite number of observations. In the context of Neural Networks the term “learning” means that the knowledge acquired from the samples can be generalized to as yet unseen observations. In this sense, a Neural Network is often called a *Learning Machine*. As such, Neural Networks might be considered as a metaphor for an agent who *learns* dependencies of his environment and thus infers strategies of behavior based on a limited number of observations. In this contribution, however, we want to abstract from the biological origins of Neural Networks and rather present them as a purely mathematical models.

The paper is organized as follows: In Section 2 we will summarize the main results of Statistical Learning Theory which provide a basis for understanding the generalization properties of existing Neural Network learning algorithms. In Section 3 we will introduce basic concepts and techniques of Neural Network Learning. Then, in Section 4, we will give an overview of existing applications of Neural Networks in Economics. Recently, however, the ideas from Statistical Learning Theory, as introduced in Section 2, have lead the way to the so called Support Vector learning, which will be described in Section 5 for the task of classification. Finally, in Section 6 we leave the review track and present a state–of–the–art application of this technique to the problem of learning the preference structure from a given set of observations. In contrast to cardinal utility theory our approach is based on ordinal utilities and requires as input only objects together with their respective relation (i.e. preference ranking). Throughout the paper, we will support the theoretical arguments by examples.

2 Statistical Learning Theory

Before studying the principles of Neural Network learning, we will give some results from Statistical Learning Theory (Vapnik 1982; Pollard 1984; Vapnik 1998). These results will provide insights into the problems encountered when trying to learn from finite samples. To make the following definitions more transparent, we start with a simple example regarding the classification of bank customers (Feng and Michie 1994).

Example 1 (Classification). A bank is confronted with the task of judging its customers according to their reliability of paying back a given loan. To this end the bank collects information in the form of n measurable properties (*features*) of the customers, e.g. age, sex, income .... Let us denote each feature by a variable $X_i \subseteq \mathbb{R}$. Then each customer is completely described by an n–dimensional vector $\mathbf{x} = (x_1, \ldots, x_n) \in X \subseteq \mathbb{R}^n$. We call the space X the *input space* since all customers considered are represented as points in this space. Together with the description of its customers the bank records for each customer if he pays back his loan ($y = 1$) or not ($y = -1$). The space of all possible outcomes of the judgments is often called the *output space* Y and in the present case has only two elements. The bank is now equipped with a finite sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ of size ℓ. The purpose of classification learning is — based on the given sample S — to find a function $h : X \mapsto Y$ that assigns each new customer (represented by a vector $\mathbf{x}$) to one of the classes "reliable" ($y = 1$) or "unreliable" ($y = -1$). To find such a mapping — usually called *classifier* —, the bank assigns a risk to each hypothesis h. Here the risk of each classifier h is the probability of assigning a customer $\mathbf{x}$ to the wrong class, that is to classify a "reliable" customer to be "unreliable" and vice versa. Statistical Learning Theory makes this risk more explicit by assuming a probability P_{XY} that a randomly chosen customer (taken from the space X) is reliable ($y = 1$) or unreliable ($y = -1$). Let the cost of assigning a customer to class $\hat{y}$ whereas the true class is y be given by

$$L(y, \hat{y}) \quad = \quad \begin{cases} 0 & y = \hat{y} \\ 1 & y \neq \hat{y} \end{cases} \quad . \tag{1}$$

This so called *zero–one loss* represents one of the simplest ways of quantifying the cost of a classification error. In particular (e.g. economic) applications, other possibly more specific losses may be preferable that take into account the real economic cost associated with an error. The expected value of L over P_{XY} — often called the expected *loss* — is the probability of misclassification. More formally, the *risk* functional

$$R(h) \quad = \quad \int_{XY} L(y, h(\mathbf{x})) P_{XY}(\mathbf{x}, y) \; d\mathbf{x} \; dy \tag{2}$$

is the quantity to be minimized from the viewpoint of the bank. Since the bank is only given the finite sample S (*training set*), only the *empirical risk* (*training* *error*) is accessible

$$R_{\text{emp}}(h) \;=\; \frac{1}{\ell} \sum_{(\mathbf{x}_i,y_i)\in S} L(y_i, h(\mathbf{x}_i))\,. \tag{3}$$

Let us consider a set $\mathcal{H}$ of classifiers h which is called *hypothesis space.* In the context of Neural Network Learning one possible choice of $\mathcal{H}$ is the set of parameterized functions $h(\mathbf{x};\boldsymbol{\alpha}) = \text{sign}(\alpha_1 x_1 + \cdots + \alpha_n x_n)$ which is called a *perceptron* (Rosenblatt 1962). This classifier is described by a parameter vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ and the task of learning reduces to finding a vector $\boldsymbol{\alpha}^*$ that minimizes $R(h(\cdot;\boldsymbol{\alpha}))$ without extra knowledge about P_{XY}. As will be shown later in this section, minimizing $R_{\text{emp}}(h(\cdot;\boldsymbol{\alpha}))$ is — under certain circumstances to be described — a consistent way of minimizing $R(h(\cdot;\boldsymbol{\alpha}))$. This principle is called *Empirical Risk Minimization.* In the following we will abbreviate $h(\cdot;\boldsymbol{\alpha})$ by $\boldsymbol{\alpha}$ and $\mathcal{H} = \{h(\cdot;\boldsymbol{\alpha})|\boldsymbol{\alpha} \in \Lambda\}$ by Λ.

The Learning Problem To summarize the relevant results from the last example, we can formulate the task of statistical learning as follows: Given a finite sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell} \subset X \times Y$ and a hypothesis space $\mathcal{H} = \{h(\cdot;\boldsymbol{\alpha})|\boldsymbol{\alpha} \in \Lambda\}$ we want to find $\boldsymbol{\alpha}^*$ such that

$$\begin{aligned} \boldsymbol{\alpha}^* &= \text{argmin}_{\boldsymbol{\alpha}\in\Lambda} R(\boldsymbol{\alpha}) \\ &= \text{argmin}_{\boldsymbol{\alpha}\in\Lambda} \int_{XY} L(y, h(\mathbf{x};\boldsymbol{\alpha})) P_{XY}(\mathbf{x}, y)\; d\mathbf{x}\; dy \end{aligned} \tag{4}$$

while no knowledge except the sample is given about P_{XY}.

Empirical Risk Minimization In Vapnik and Chervonenkis (1971) a principle is formulated which can be used to find a classifier $\boldsymbol{\alpha}_\ell$ whose performance is close to the one of the optimal classifier $\boldsymbol{\alpha}^*$ — independently of the used hypothesis space and any assumptions on the underlying probability P_{XY}. The principle says that choosing $\boldsymbol{\alpha}_\ell$ such that

$$\begin{aligned} \boldsymbol{\alpha}_\ell &= \text{argmin}_{\boldsymbol{\alpha}\in\Lambda} R_{\text{emp}}(\boldsymbol{\alpha}) \\ &= \text{argmin}_{\boldsymbol{\alpha}\in\Lambda} \frac{1}{\ell} \sum_{(\mathbf{x}_i,y_i)\in S} L(y_i, h(\mathbf{x}_i;\boldsymbol{\alpha})) \end{aligned} \tag{5}$$

leads to the set of parameters $\boldsymbol{\alpha}_\ell$ that minimizes the deviation $|R(\boldsymbol{\alpha}^*) - R(\boldsymbol{\alpha}_\ell)|$ — under conditions explicitly stated in the paper. Since this principle can be explained as "choosing that classifier $\boldsymbol{\alpha}_\ell$ that minimizes the training error or empirical risk respectively", this principle is known as *Empirical Risk Minimization* (ERM). Although this principle had been used years before, Vapnik and Chervonenkis (1971) gave an explicit explanation of its applicability.

Generalization Let us recall what exactly the difference $|R(\boldsymbol{\alpha}^*) - R(\boldsymbol{\alpha}_\ell)|$ measures. In view of Neural Network Learning, this difference is often called generalization error. We can bound the generalization error above by

$$|R(\boldsymbol{\alpha}^*) - R(\boldsymbol{\alpha}_\ell)| \;\leq\; |R(\boldsymbol{\alpha}^*) - R_{\text{emp}}(\boldsymbol{\alpha}^*)| + \max_{\boldsymbol{\alpha}\in\Lambda} |R(\boldsymbol{\alpha}) - R_{\text{emp}}(\boldsymbol{\alpha})|\,, \tag{6}$$

where the second term is greater or equal to $|R(\boldsymbol{\alpha}_\ell) - R_{\text{emp}}(\boldsymbol{\alpha}_\ell)|$. Although $\boldsymbol{\alpha}^*$ is uniquely defined by Equation (4), $\boldsymbol{\alpha}_\ell$ strongly depends on the randomly drawn training set S. Thus, we have to bound $\max_{\boldsymbol{\alpha}\in\Lambda} |R(\boldsymbol{\alpha}) - R_{\text{emp}}(\boldsymbol{\alpha})|$. What is available from the data, however, is the empirical risk $R_{\text{emp}}(\boldsymbol{\alpha}_\ell)$. This implies that we should aim at minimizing the empirical risk while ensuring that the abovementioned generalization error remains small. Solely minimizing the empirical risk can lead to what is known as overfitting in the Neural Network community: The training data are well fit but no relieable predicition can be made with regard to data not contained in the training set.

The Basic Inequality Note from Equations (4) and (5) that if Λ contains a finite number of possible classifiers, the principle of choosing $\boldsymbol{\alpha}_\ell$ to approximate $\boldsymbol{\alpha}^*$ is consistent. Consistency means that the generalization error (6) can be bounded with probability one if ℓ tends to infinity. This is due to the Law of Large Numbers, since $R(\boldsymbol{\alpha})$ is the expectation of the loss of $\boldsymbol{\alpha}$ and $R_{\text{emp}}(\boldsymbol{\alpha})$ is the mean of the loss of $\boldsymbol{\alpha}$ which converges uniformly to the expectation independently of the distribution P_{XY}. Problems occur if we consider an infinite set of classifiers like in the abovementioned example. For this case, Vapnik and Chervonenkis proved the following basic result

$$P\left\{\max_{\boldsymbol{\alpha}\in\Lambda} |R_{\text{emp}}(\boldsymbol{\alpha}) - R(\boldsymbol{\alpha})| > \varepsilon\right\} \quad\leq\quad 4\exp\left\{\left(\frac{c_\Lambda(\ln\frac{2\ell}{c_\Lambda}+1)}{\ell} - \varepsilon^2\right)\ell\right\}, \tag{7}$$

where ℓ is the number of training examples and c_Λ is a constant depending on the investigated set of functions. It is called Vapnik–Chervonenkis–dimension (VC dimension) of Λ and is a measure of the capacity of the set of hypotheses under consideration. If we set the right–hand side of the inequality to the confidence δ, solve for ε, and drop the lower bound we get the following corollary: For each classifier $\boldsymbol{\alpha} \in \Lambda$ considered during the learning process, the inequality

$$R(\boldsymbol{\alpha}) \quad\leq\quad R_{\text{emp}}(\boldsymbol{\alpha}) + \sqrt{\frac{c_\Lambda(\ln\frac{2\ell}{c_\Lambda}+1) - \ln\delta/4}{\ell}} \tag{8}$$

holds with probability $1-\delta$. For the learning problem this means that the expected risk $R(\boldsymbol{\alpha})$, which is the quantity of interest, can be bounded above by a sum of two terms: the empirical risk $R_{\text{emp}}(\boldsymbol{\alpha})$, which can be calculated from the training set, and a capacity term which accounts for the complexity of the hypothesis space $\mathcal{H}$ in relation to the number of training examples. In this context, Empirical Risk Minimization as described above attempts to minimize $R_{\text{emp}}(\boldsymbol{\alpha})$ while keeping the capacity term fixed. The goal of *model selection* is to find a trade-off between the explanatory power and capacity control of the Neural Network. In order to allow for a better understanding of the capacity consider the following two definitions on the VC dimension c_Λ.

Definition 1 (Shattering). *A subset* $\mathbf{x}_1, \ldots, \mathbf{x}_n \in X^n$ *is* shattered *by the set of functions* Λ, *if for each of the* 2^n *possible class labelings* ($y_i = 1$ *or* $y_i = -1$) *there exists a function* $\boldsymbol{\alpha} \in \Lambda$ *that classifies the examples in this way.*

Definition 2 (VC–dimension). *The* VC–dimension c_Λ *of a set of functions* Λ *is the maximum number of elements* $\mathbf{x}_1, \ldots, \mathbf{x}_{c_\Lambda} \subset X^{c_\Lambda}$ *that can be shattered by the set of functions* Λ.

Let us consider a simple example in order to understand the concepts of shattering and VC dimension.

Example 2 (Shattering and VC–dimension). Let us examine the functions from the last example, that is all classifiers of the form $h(\mathbf{x}; \boldsymbol{\alpha}) = \text{sign}(x_1\alpha_1 + \cdots + x_n\alpha_n)$ and $n = 2$. Now consider the set $\mathbf{x}_1 = (1,1)'$ and $\mathbf{x}_2 = (1,2)'$. Then for each of the 4 different labelings there is a vector $\boldsymbol{\alpha}$ that can separate the two points, that is the set $\mathbf{x}_1, \mathbf{x}_2$ can be shattered by these classifiers (see Figure 1). Hence, the VC–dimension of these classifiers is at least $c_\Lambda = 2$. It can easily be verified that there exists no set of three points in $\mathbb{R}^2$ that can be shatter by these classifiers. Thus, the VC–dimension of these classifiers is exactly $c_\Lambda = 2$. Note, that there are also sets of two points, that cannot be shattered, e.g. $\mathbf{x}_1 = (1,1)'$ and $\mathbf{x}_2 = (2,2)'$.

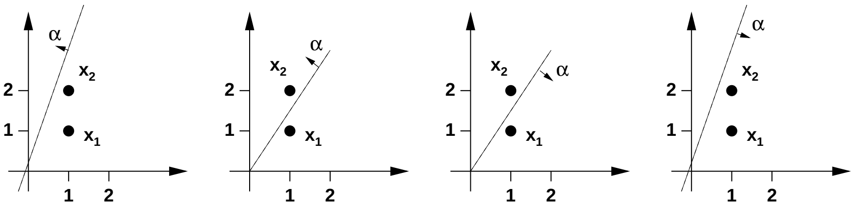


Figure 1: Shattering of a set of two points $\mathbf{x}_1, \mathbf{x}_2$ by linear decision functions. All points in the halfspace to which the arrow points are labeled $y = 1$.

Structural Risk Minimization Vapnik (1982) presented a new learning principle, the so called *Structural Risk Minimization* (SRM), which can easily be justified by Inequality (8). The idea of this principle is to define *a priori* nested subsets $\Lambda_1 \subset \Lambda_2 \subset \cdots \subset \Lambda$ of functions and applying the ERM principle (training error minimization) in each of the predefined Λ_i to obtain classifiers $\boldsymbol{\alpha}_\ell^i$. Exploiting the inequality, one is able to select that classifier $\boldsymbol{\alpha}_\ell^{i^*}$ which minimizes the right hand side of (8). Then the learning algorithm not only finds the best classifier in a given set of functions but also finds the best (sub)set of functions together with the best classifier. This corresponds to the model selection process mentioned earlier.

Prior Knowledge We want to make some remarks about prior knowledge. Without prior knowledge no learning can take place since only the combination of single instances and previous knowledge can lead to meaningful conclusions (Haussler 1988; Wolpert 1995). For a finite set of samples there exists an infinite number of compatible hypotheses and only criteria that are not directly derivable from the data at hand can single out the underlying dependency. In classical statistics, prior knowledge is exploited in the form of probability distributions over the data (Likelihood principle) or over the considered functions (Bayesian principle). If these priors do not correspond to reality, no confidence can be given by any statistical approach about the generalization error (6). In Neural Network learning, the explicit knowledge is replaced by restrictions on the assumed dependencies (finiteness of VC–dimension). Therefore, these approaches are often called worst–case approaches. Using SRM, the prior knowledge is incorporated by the *a priori* defined nesting of the set of functions. For practical purposes, this is less restrictive than the distribution assumptions of classical statistical approaches.

3 Algorithms for Neural Network Learning

In this section we give an overview on different types of Neural Networks and on existing algorithms for Neural Network learning. The aim is to highlight the basic principles rather than to cover the whole range of existing algorithms. A number of textbooks describe relevant algorithms in more detail (see Haykin (1994) or Bishop (1995)).

3.1 Feed–forward Neural Networks

In the past and due to its origins, the term *Neural Network* was used to describe a network of "neurons" (i.e. simple processing units) with a fixed dynamic for each neuron (Rosenblatt 1962). Such a viewpoint should indicate the close relationship to the field of Neuroscience. We want to abstract from the biological arigin and view Neural Networks as a purely mathematical models. We limit our attention to those Neural Networks that can be used for classification and focus on feed–forward networks[1]. In these networks computations are performed by feeding the data into the n units of an input layer from which they are passed through a sequence of hidden layers and finally to m units of the output layer. In **?**) it was shown, that (under some mild conditions) each continuous decision function can be arbitrarily well approximated by a Neural Network with only one hidden layer. Let us denote the number of units in the hidden layer by r. Hence, it is sufficient to consider a network described by

$$h(\mathbf{x};\boldsymbol{\alpha}) \quad = \quad f_2(f_1(\mathbf{x};\boldsymbol{\beta});\boldsymbol{\gamma})\,, \tag{9}$$

where $f_1 : \mathbb{R}^n \mapsto \mathbb{R}^r$ and $f_2 : \mathbb{R}^r \mapsto \mathbb{R}^m$ are continuous functions. $\boldsymbol{\alpha} = (\boldsymbol{\beta}, \boldsymbol{\gamma})'$ is the vector of adjustable parameters, consisting of $\boldsymbol{\beta}$ which is the vector of weights of the hidden layer and $\boldsymbol{\gamma}$ being the weight vector of the output layer. For illustrative purposes it is common practice to represent each unit where a computation is being performed ("neuron") by a node and each connection ("synapse") by an edge of a graph. An example of a two layer Neural Network is shown in Figure 2. In the following we will make the functions f_1 and f_2

[1] Note, that in particular dynamical system are often modelled by recurrent Neural Networks (Hopfield and Tank 1986). We will not consider these here because they do not offer new insights into the problem of classification and preference learning. See, e.g., Kuan and Liu (1995) or Haykin (1994) for a textbook account.

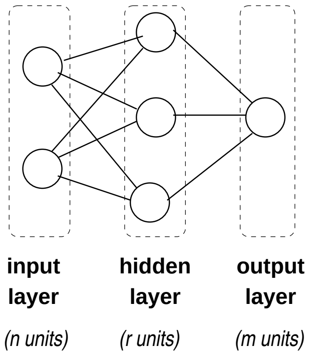


Figure 2: A sketch of a two layer Neural Network with input space $X = \mathbb{R}^n = \mathbb{R}^2$, hidden neuron space $\mathbb{R}^r = \mathbb{R}^3$ and output space $Y = \mathbb{R}^m = \mathbb{R}^1$. Note, that the input layer does not count as a computational layer.

more explicit to derive different type of Neural Networks and their respective learning algorithms.

Perceptron and the Delta Rule Consider the simple case $r = n, m = 1, Y = \{-1, +1\}, f_1(\mathbf{x}; \boldsymbol{\beta}) = \mathbf{x}$, and $f_2(\mathbf{x}; \boldsymbol{\gamma}) = \text{sign}(\sum_{i=1}^{n} \gamma_i x_i) = \text{sign}(\boldsymbol{\gamma}'\mathbf{x})$. This type of Neural Network is called a *perceptron* and was mentioned earlier in Example 1. Learning in such a network reduces to finding the vector $\boldsymbol{\alpha} = \boldsymbol{\gamma}$ that minimizes $R_{\text{emp}}(\boldsymbol{\alpha})$ (see Section 2). We will consider here only the case that there exists a vector $\boldsymbol{\alpha}$ such that $R_{\text{emp}}(\boldsymbol{\alpha}) = 0$, i.e. linear separability. Let us rewrite the empirical risk of a vector $\boldsymbol{\alpha}$ as $R_{\text{emp}}(\boldsymbol{\alpha}) = \frac{1}{\ell} \sum_{t=1}^{\ell} R_{\text{emp}}(\boldsymbol{\alpha}, \mathbf{x}_t)$ where

$$R_{\text{emp}}(\boldsymbol{\alpha}, \mathbf{x}_t) \quad \propto \quad (\text{sign}(\boldsymbol{\alpha}'\mathbf{x}_t) - y_t)^2 \, . \tag{10}$$

To minimize this functional one can use the so called *Delta rule* (Rosenblatt 1962) which is similiar to a stochastic gradient descent (e.g. Hadley (1964)) on the empirical risk functional. Starting with an initial guess $\boldsymbol{\alpha}_0$, the following iteration scheme is used

$$\boldsymbol{\alpha}_{t+1} \quad = \quad \boldsymbol{\alpha}_t - \frac{1}{2}(\text{sign}(\boldsymbol{\alpha}_t'\mathbf{x}_t) - y_t)\mathbf{x}_t \tag{11}$$

The following pseudo–code gives an overview of the learning procedure[2].

[2] Note, that if $\mathbf{x}_i$ is misclassified, $y_t = -\frac{1}{2}(\text{sign}(\boldsymbol{\alpha}_t'\mathbf{x}_t) - y_t)$.

Perceptron learning

```
$\boldsymbol{\alpha}_0$ := randomly initialized, $t := 0$
do
    classify each training example $\mathbf{x}_i$ using $\boldsymbol{\alpha}_t$
    if $\mathbf{x}_i$ is misclassified then
        $\boldsymbol{\alpha}_{t+1} := \boldsymbol{\alpha}_t + y_i\mathbf{x}_i$, $t := t+1$
    end if
while misclassified training examples exist
```

Multilayer perceptron and Backpropagation The addition of hidden neurons increases the VC dimension of the network and thus leads to more powerful models of the data. For the case of a two–layer perceptron one chooses $f_1(\mathbf{x};\boldsymbol{\beta}) = (g_1(\boldsymbol{\beta}_1'\mathbf{x}),\ldots,g_1(\boldsymbol{\beta}_r'\mathbf{x}))'$ and $f_2(\mathbf{z};\boldsymbol{\gamma}) = g_2(\boldsymbol{\gamma}'\mathbf{z})$, where $\mathbf{z}$ is the r–dimensional vector of hidden neuron activations, $\boldsymbol{\beta} = (\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_r)'$, and $g_1 : \mathbb{R} \mapsto \mathbb{R}$ and $g_2 : \mathbb{R} \mapsto \mathbb{R}$ are the "transfer" functions of the neurons. Usually these are differentiable sigmoidal functions, e.g., $g_i(a) = \tanh(c \cdot a)$. This type of Neural Network is called a *multilayer perceptron* (MLP). In the case of classification $m = 1$ and $Y = \{-1,+1\}$. Since in the last layer a continuous activation function g_2 is used, it becomes necessary to map the continuous output to a binary decision. This can be done by thresholding the continuous ouput, e.g., $\operatorname{sign}(h(\mathbf{x};\boldsymbol{\alpha}) - \theta)$. Again, learning in such a network reduces to finding the vector $\boldsymbol{\alpha} = (\boldsymbol{\beta},\boldsymbol{\gamma})'$ that minimizes $R_{\mathrm{emp}}(\boldsymbol{\alpha})$ (see Section 2). Using

$$R_{\mathrm{emp}}(\boldsymbol{\alpha},\mathbf{x}_t) \propto \frac{1}{2}\left(g_2\left(\sum_{j=1}^{r}\gamma_j g_1\left(\boldsymbol{\beta}_j'\mathbf{x}_t\right)\right) - y_t\right)^2 = \frac{1}{2}(h(\mathbf{x}_t,\boldsymbol{\alpha}) - y_t)^2 \qquad (12)$$

one calculates $\nabla_{\gamma} R_{\mathrm{emp}}(\boldsymbol{\alpha},\mathbf{x}_t)$ and $\nabla_{\beta_j} R_{\mathrm{emp}}(\boldsymbol{\alpha},\mathbf{x}_t)$ to apply gradient descent. Note that (12) relates to the fraction of misclassified objects, for $c \to \infty$. Successive application of the chain rule of differentiation gives

$$\begin{aligned} \nabla_{\gamma} R_{\mathrm{emp}}(\boldsymbol{\alpha},\mathbf{x}_t) &= (h(\mathbf{x}_t;\boldsymbol{\alpha}) - y_t)g_2'(\boldsymbol{\gamma}'\mathbf{z}_t)\mathbf{z}_t && (13) \\ \nabla_{\beta_j} R_{\mathrm{emp}}(\boldsymbol{\alpha},\mathbf{x}_t) &= (h(\mathbf{x}_t;\boldsymbol{\alpha}) - y_t)g_2'(\boldsymbol{\gamma}'\mathbf{z}_t)g_1'(\boldsymbol{\beta}_j\mathbf{x}_t)\mathbf{x}_t\,, && (14) \end{aligned}$$

where g' denotes the first derivative of g w.r.t. its argument, and $\mathbf{z}_t = f_1(\mathbf{x}_t;\boldsymbol{\beta})$. The following pseudo–code gives an overview of the resulting *backpropagation* learning procedure.

```
Backpropagation learning (MLP Networks)

$\boldsymbol{\alpha}_0 = (\boldsymbol{\beta}_{1,\ldots,r;0}, \boldsymbol{\gamma}_0)' :=$ randomly initialized
$\eta_0 := 1,\ t := 0$
do
    $\boldsymbol{\alpha}_{old} := \boldsymbol{\alpha}_t$
    for each training example $\mathbf{x}_i$ calculate $h(\mathbf{x}_i; \boldsymbol{\alpha}_t)$ and $\mathbf{z}_i = f_1(\mathbf{x}_i; \boldsymbol{\beta}_t)$
    $\boldsymbol{\gamma}_{t+1} := \boldsymbol{\gamma}_t - \eta_t (h(\mathbf{x}_i; \boldsymbol{\alpha}_t) - y_i) g_2'(\boldsymbol{\gamma}_t' \mathbf{z}_i) \mathbf{z}_i$
    $\boldsymbol{\beta}_{j;t+1} := \boldsymbol{\beta}_{j;t} - \eta_t (h(\mathbf{x}_i; \boldsymbol{\alpha}_t) - y_i) g_2'(\boldsymbol{\gamma}_t' \mathbf{z}_i) g_1'(\boldsymbol{\beta}_{j;t} \mathbf{x}_i) \mathbf{x}_i$
    $t := t + 1,\ \eta_t := 1/t$
while $\|\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{old}\| > \varepsilon$
```

In the beginning the learning rate η_t is large so as to enable the algorithm to find a good minimum of the empirical risk $R_{\text{emp}}(\boldsymbol{\alpha})$. In the course of learning η_t is decreased in order to allow for the fine tuning of the parameters. This is the basic principle used to learn the adjustable parameters $\boldsymbol{\alpha}$ of a MLP given a finite sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ without exploiting further prior knowledge. More refined methods exist if prior assumptions are made about the type of the surface $R_{\text{emp}}(\boldsymbol{\alpha})$ over the space of $\boldsymbol{\alpha}$, e.g., conjugate gradient learning (Hestenes and Stiefel 1952; Johansson et al. 1992), momentum learning (Rumelhart et al. 1986), or Quickprop (Fahlman 1989). Moreover, backpropagation can easily be extended to more than one hidden layer, in which case its recursive nature trough the layers becomes apparent.

Radial Basis Function Networks. Another popular Neural Network architecture is obtained for $f_1(\mathbf{x}; \boldsymbol{\beta}) = (g_1(\mathbf{x}, \boldsymbol{\beta}_1, \sigma_1), \ldots, g_1(\mathbf{x}, \boldsymbol{\beta}_r, \sigma_r))'$ and $f_2(\mathbf{z}; \boldsymbol{\gamma}) = g_2(\boldsymbol{\gamma}'\mathbf{z})$ where $g_1 : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}$ is a function that acts locally in the $\boldsymbol{\beta}_j$, $g_2 : \mathbb{R} \mapsto \mathbb{R}$ is a sigmoidal function like in the MLP, $\mathbf{z}$ is the r–dimensional vector of hidden neuron activations, and $\boldsymbol{\beta} = ((\boldsymbol{\beta}_1, \sigma_1), \ldots, (\boldsymbol{\beta}_r, \sigma_r))'$. This type of Neural Network is called a *radial basis function network* (RBF). Usually the function $g_1(\mathbf{x}, \boldsymbol{\beta}_j, \sigma_j)$ is given by a Gaussian of the form

$$g_1(\mathbf{x}, \boldsymbol{\beta}_j, \sigma_j) \quad = \quad \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\beta}_j\|^2}{2\sigma_j^2} \right) . \tag{15}$$

Again, we consider the case of binary classification. Similiarly to backpropagation the empirical risk becomes

$$R_{\text{emp}}(\boldsymbol{\alpha}, \mathbf{x}_t) \propto \frac{1}{2} \left(g_2 \left(\sum_{j=1}^{r} \gamma_j g_1 \left(\mathbf{x}_t, \boldsymbol{\beta}_j, \sigma_j \right) \right) - y_t \right)^2 \tag{16}$$

In order to apply gradient descent, we calculate the gradients $\nabla_{\gamma} R\mathrm{emp}(\boldsymbol{\alpha}, \mathbf{x}_t)$, $\nabla_{\boldsymbol{\beta}_j} R\mathrm{emp}(\boldsymbol{\alpha}, \mathbf{x}_t)$, and $\nabla_{\sigma_j} R\mathrm{emp}(\boldsymbol{\alpha}, \mathbf{x}_t)$. Successive application of the chain rule of differentiation yields

$$\begin{aligned}
\nabla_{\gamma} R\mathrm{emp}(\boldsymbol{\alpha}, \mathbf{x}_t) &= (h(\mathbf{x}_t; \boldsymbol{\alpha}) - y_t) g_2'(\boldsymbol{\gamma}' \mathbf{z}_t) \mathbf{z}_t \,, \\
\nabla_{\boldsymbol{\beta}_j} R\mathrm{emp}(\boldsymbol{\alpha}, \mathbf{x}_t) &= (h(\mathbf{x}_t; \boldsymbol{\alpha}) - y_t) g_2'(\boldsymbol{\gamma}' \mathbf{z}_t) \exp\left(-\frac{\|\mathbf{x}_t - \boldsymbol{\beta}_j\|^2}{2\sigma_j^2}\right) \frac{\mathbf{x}_t - \boldsymbol{\beta}_j}{\sigma_j^2} \,, \\
\nabla_{\sigma_j} R\mathrm{emp}(\boldsymbol{\alpha}, \mathbf{x}_t) &= (h(\mathbf{x}_t; \boldsymbol{\alpha}) - y_t) g_2'(\boldsymbol{\gamma}' \mathbf{z}_t) \exp\left(-\frac{\|\mathbf{x}_t - \boldsymbol{\beta}_j\|^2}{2\sigma_j^2}\right) \frac{\|\mathbf{x}_t - \boldsymbol{\beta}_j\|^2}{\sigma_j^3} \,.
\end{aligned}$$

The following pseudo–code gives an overview of the backpropagation learning procedure for RBF Networks (Powell 1992). In Section 5 we will present another learning algorithm that can be used for RBF Networks.

RBF Network learning

```
$\boldsymbol{\alpha}_0 = (\boldsymbol{\beta}_{1,\ldots,r;0}, \boldsymbol{\sigma}_0, \boldsymbol{\gamma}_0)' :=$ randomly initialized
$\eta_0 := 1$, $t := 0$
do
    $\boldsymbol{\alpha}_{old} := \boldsymbol{\alpha}_t$
    for each training example $\mathbf{x}_i$ calculated $h(\mathbf{x}_i; \boldsymbol{\alpha}_t)$ and $\mathbf{z}_i = f_1(\mathbf{x}_i; \boldsymbol{\beta}_t)$
    $\boldsymbol{\gamma}_{t+1} := \boldsymbol{\gamma}_t - \eta_t (h(\mathbf{x}_i; \boldsymbol{\alpha}_t) - y_i) g_2'(\boldsymbol{\gamma}_t' \mathbf{z}_i) \mathbf{z}_i$
    $\boldsymbol{\beta}_{j;t+1} := \boldsymbol{\beta}_{j;t} - \eta_t (h(\mathbf{x}_i; \boldsymbol{\alpha}_t) - y_i) g_2'(\boldsymbol{\gamma}_t' \mathbf{z}_i) \exp\left(-\frac{\|\mathbf{x}_i - \boldsymbol{\beta}_{j;t}\|^2}{2\sigma_{j;t}^2}\right) \frac{\mathbf{x}_i - \boldsymbol{\beta}_{j;t}}{\sigma_{j;t}^2}$
    $\sigma_{j;t+1} := \sigma_{j;t} - \eta_t (h(\mathbf{x}_i; \boldsymbol{\alpha}_t) - y_i) g_2'(\boldsymbol{\gamma}_t' \mathbf{z}_i) \exp\left(-\frac{\|\mathbf{x}_i - \boldsymbol{\beta}_{j;t}\|^2}{2\sigma_{j;t}^2}\right) \frac{\|\mathbf{x}_i - \boldsymbol{\beta}_{j;t}\|^2}{\sigma_{j;t}^3}$
    $t := t + 1$, $\eta_t := 1/t$
while $\|\boldsymbol{\alpha}_t - \boldsymbol{\alpha}_{old}\| > \varepsilon$
```

3.2 Locality and Regularization

Global vs. local approximation The main conceptual difference between MLP's and RBF networks is that the former perform a global approximation in input space while the latter implement a local approximation. The hidden neurons of an RBF network specialize to localized regions in data space by fitting a set of Gaussians ("receptive field") to the data. In the extreme case, where $r = \ell$, i.e. there are as many hidden neurons as data points in the training set, the ERM principle cannot lead to consistent learning because such an RBF networks can be shown to have infinite VC dimension. Using fewer neurons than data points, however, speeds up convergence during learning. In contrast to the local approximation performed by RBF Networks an MLP considers the data space as a whole and is thus able to capture complex dependencies underlying the data. The hidden neurons in both, the MLP and the RBF, perform a preprocessing of the data by learning a mapping of the input space to the space of hidden neurons. We will meet this idea later again when considering the extension of Support Vector Learning to the nonlinear case (see Section 5). The advantage of preprocessing the data is the reduction of their dimensionality. This problem may arise when the input data are of high dimensionality and thus the input data density is small. This phenomenon is referred to as the *curse of dimensionality*, i.e. the increase of necessary samples to obtain a small generalization error grows exponentially in the number of dimensions (number of parameters in a Neural Network). This can easily be seen, if one solves Equation (8) for ℓ with ε and δ fixed and assuming that c_Λ scales linearly with the number of parameters.

Regularization Techniques Conceptually, with the techniques discussed in Section 3.1 only the term $R_{\text{emp}}(\boldsymbol{\alpha})$ on the right hand side of Inequality (8) is minimized during the learning process. The Neural Network community also developed approaches that take into account model complexity as expressed in the second term of the right hand side of (8). In the case of RBF and MLP's it was shown that reduction of $\|\boldsymbol{\alpha}\|^2$ minimizes their VC dimension (model complexity) (Cortes 1995; Shawe-Taylor et al. 1996). Bartlett (1998) also showed that backpropagation learning when initialized with small weights leads to a class of functions with small VC–dimension. Another way to incorporate this into the learning process is to the minimize $R_{\text{emp}}(\boldsymbol{\alpha}) + \kappa\|\boldsymbol{\alpha}\|^2$ where κ has to be chosen beforehand. Such a technique is also called *regularization* (Poggio and Girosi 1990) and was successfully used in the *weight decay* learning algorithm (Hinton 1987). The Support Vector algorithm to be presented in Section 5 makes use of a similar technique.

4 Economic Applications of Neural Networks – An Overview of the Literature

With the application of backpropagation to Neural Network learning (Rumelhart et al. 1986) and the revived interest into Neural Networks, Economists started to adopt this tool as well, since the Neural Networks for classification and regression can easily be adopted them to economic problems. It seems reasonable to distinguish at least two major applications of Neural Networks in Economics: First, the classification of economic agents, i.e. customers or company, and second, the prediction of time series. A third, though less common application of Neural Networks is to model bounded rational economic agents. Let us review the literature that is concerned with these three applications in turn.

4.1 Classification of Economic Agents

As discussed above, one of the main abilities of Neural Networks is to classify a set of data into different categories. Thus, Neural Networks can be used as an alternative to more traditional methods such as discriminant analysis or logistic regression[3]. A special feature of Neural Networks that distinguishes them from traditional methods is their ability to classify data which are not linearly separable[4]. The majority of papers that use Neural Networks for classification tasks in Economics can be found in the area of *bankruptcy prediction* of economic agents, mainly banks. Most of these papers have been published in the early 1990's, a period that witnessed a significant rise of bankruptcies in the U.S..

The approach is to use financial ratios calculated from a firm's balance as input to the Neural Network to obtain an estimate for the probability of bankruptcy as output. Examples are Odom and Sharda (1990) and Rahimian et al. (1993) who used five financial ratios that have been suggested by Altman (1968) for discriminant analysis. Both papers use a two-layer Neural Network trained using backpropagation as discussed in Section 3.1. They report an improvement of the classification quality as compared to discriminant analysis. While the latter classified 60% of the firms correctly, Neural Networks classified 70–80% in the right way. Tam (1991) and Tam and Kiang (1992) analyzed a Neural Network with 19 input neurons, i.e., they used 19 financial ratios. In their studies they compared a simple feed forward network with no hidden layer with a two-layer network trained using backpropagation. The performance of the latter was better on average than the one of the former. However, both types of networks performed on average better than other more traditional classification methods[5]. Other applications with similar results are e.g. Poddig (1995), Salchenberger et al. (1992), Altman et al. (1994), and Erxleben et al. (1992). The latter report "nearly identical" performance for discriminant analysis and neural networks.

Further discusions of the classification properties are given, e.g., by Brockett et al. (1994) for insurance companies, Marose (1990) for the creditworthiness of bank customers, Grudnitzki (1997) for the valuation of residential properties in the San Diego County, Jagielska and Jaworski (1996), who applied Neural Networks to predict the probability of credit card holders to run into illiquidity, or Martin-del Brio and Serrano-Cinca (1995), who classified Spanish companies into more than one category. Finally, Coleman et al. (1991) suggested an integration of a Neural Network and an expert system such that courses of action can be recommended to prevent the bankruptcy. As an overall result, Neural Nets seem to perform well when applied to economic classification problems and they often appear to be superior to classical methods.

[3] See the discussion in Ripley (1994). A literature review of traditional methods of Business Evaluation can be found in Raghupati et al. (1993).

[4] See the illustration in Trippi and Turban (1990, p. 12) or Blien and Lindner (1993).

[5] See, e.g., Tam (1991, Table 9). The authors compared the performance of NN's with different types of discriminant analysis, with logistic regression, with the method of nearest neighbours and with classification tree methods.

4.2 Time Series Prediction

Probably the largest share of economic applications of Neural Networks can be found in the field of prediction of time series in the capital markets. Usually, linear models of financial time series (like exchange rates or stock exchange series) perform poorly and linear univariate models consistently give evidence for a random walk.[6] This has been taken in favour of the efficient market hypothesis where efficieny means that the market fully and correctly reflects all relevant information in determining security prices[7]. However this hypothesis is not generally accepted and, therefore, an often followed strategy is to try to use nonlinear models to improve the fit and thus the prediction[8]. As mentioned earlier Neural Networks are flexible functional forms that allow to approximate any continuous — hence also nonlinear — function. Therefore, they can be expected to provide effective nonlinear models for financial time series and thus to allow for better predictions.

One of the first researcher to use Neural Networks in the capital markets was probably White (1988), who applied a two-layer neural network on a series of length 1000 of IBM stocks. Rather than to obtain predictions his aim was to test the efficient market hypothesis. He could not find evidence against it which suggests that a random walk is still the best model for a financial market. However, the network used in his study was rather simple and, therefore, a number of authors challenged White's results. Bosarge (1993) suggested an expert system with a neural network at the its core. He found significant nonlinearities in different time series (S&P 500, Crude Oil, Yen/Dollar, Eurodollar, and Nikkei-index) and was able to improve the quality of forecast considerably. Similar results have been reported by Wong (1990), Tsibouris and Zeidenberg (1995), Refenes et al. (1995), Hiemstra (1996) or Haefke and Helmenstein (1996)[9].

Other authors reported results that point to the opposite direction. In a survey of the literature, Hill et al. (1994) report mixed evidence as to forecasting results of Neural Networks, although they performed "as well as and occasionally better than" statistical methods. Mixed evidence is also reported in a paper by Kuan and Liu (1995) where they compare feedforward and recurrent Neural Networks as prediction tools for different currency exchange rates. The same applies to Verkooijen (1996), who linked financial time series to fundamental variables like GDP or trade balance within a Neural Network. Chatfield (1993) expressed caution as to comparisons between Neural Networks and linear prediction methods, because often the chosen linear methods seemed inappropriate. A major problem in the implementation of Neural Networks as predicting tools seems to be the fact that no objective guidelines exist to choose the appropriate dimension (i.e. the number of hidden layers or neurons) of the Neural Network, a problem refered to earlier as the model selection problem. Usually, implemen-

[6] See, e.g., the discussion in Meese and Rogoff (1983) or Lee et al. (1993).

[7] See Fama (1970) or Malkiel (1992) for a discussion.

[8] See, e.g., Engle (1982), Granger (1991) or Brock et al. (1991)

[9] See Trippi and Turban (1990) or Refenes (1995) for a number of other papers whose conclusion goes into the same direction.

tations refer to rules of thumb and to a trial-and-error procedures, although systematic methods have been suggested such as the Support Vector method to be presented in the following section. See also Kuan and Liu (1995), Weigend et al. (1992) or Anders and Korn (1997) for a discussions of formalized methods. Thus, as an overall result, it seems that Neural Networks have the potential to be used as forecasting tools. Their strength can be seen in the prediction of nonlinear time series. However further results are needed to make them reliable instuments for the "everyday-forecaster".

Applications of time series prediction in other than financial fields are Franses and Draisma (1997) or Swanson and White (1997) for macroeconomic variables, Church and Curram (1996) for consumers' expenditure, or Kaastra et al. (1995) for agricultural economics.

4.3 Modelling Bounded Rational Econmic Agents

A third, less common application of Neural Networks in Economics can be found in the modelling of learning processes of bounded rational adaptive artificial agents. Here, neurons are interpreted as agents who update their perception of the environment according to the information they receive. Their decisions (the output of the neuron) then exert an influence on the environment which might be fed back to the agent. It was probably Sargent (1993) who first proposed Neural Networks in this context. Beltratti et al. (1996, p. 79) argued that Neural Networks were apt to model human behaviour since they could interpolate between the learned examples and introduce some degree of uncertainty in their replies. Neural Networks can be seen as an implementation of the ideas suggested by Arthur (1993).

Cho (1994) used a Neural Network to model strategies for repeated games. He argued in favour of this tool, because it was capable of capturing complex equilibrium strategies although instructions were stationary, fixed, simple, and independent of the target payoff vector. Cho and Sargent (1996), in a revision of the paper by Cho, suggested that agents should be able to memorize the complete history of the game. This was implemented by an extension of the input vector, i.e., the dimension of the input space with every iteration step. However, as they show, memory could as well be implemented using a recurrent network with an an additional storage unit in the input layer which includes some summary statistics.

Luna (1996) used Neural Networks to model the emergence of economic institutions. The Neural Networks allowed to model feedback between a learning environment and the formation of institutions, and vice versa. Orsini (1996) proposed a Neural Network to model the consumption behaviour of individuals whose expectations about group behaviour played a crucial role on individual and aggregate outcomes. Packalén (1998) used a Neural Network to relax the assumption of linear forecast functions (that is usually made in the adaptive learning literature) and to extend them to nonlinear functions. He used three different rational expectation models as benchmarks to show how convergence to rational expectation equilibria can occur.

5 Support Vector Networks for Classification

In Section 3 the classical techniques for learning in a Neural Network were described. The learning techniques described there are essentially based on the ERM principle. In this section we want to present a new Neural Network learning technique that utilizes the SRM principle, the so called *Support Vector Learning*. It has been successfully applied in the field of character recognition (Cortes and Vapnik 1995), object recognition (Schölkopf 1997; Osuna et al. 1997a), and text categorization (Joachims 1997).

We start by developing the learning algorithm for the perceptron under the assumption that the training set can be classified without training error (objects are linearly separable). Then we extend the learning algorithm to the case where the objects are not linearly separable. Furthermore, by using a technique known as the *kernel trick* we show how the learning algorithm can be extended to the (nonlinear) case of MLP's and RBF Networks.

Case of Linearly Separable Data Consider we want to learn the vector $\boldsymbol{\alpha}$ of a perceptron (see Equation (9)). Instead of minimizing $R_{\text{emp}}(\boldsymbol{\alpha})$ (see Section 3), we assume that there exist vectors $\tilde{\boldsymbol{\alpha}}$ which achieve zero training error $R_{\text{emp}}(\tilde{\boldsymbol{\alpha}}) = 0$. In order to minimize the generalization error (6), it follows from the basic Inequality (8) that — everything else being equal — minimization of the VC–dimension c_Λ leads to the optimal classifier $\boldsymbol{\alpha}_\ell$. Therefore, in the spirit of SRM we have to define a structure nested subsets on the set of linear classifiers such that we can at least bound their VC dimension above. The following theorem gives such a structuring for the set of all linear classifiers.

Theorem 1 (VC dimension of hyperplanes (Vapnik 1995)). *Suppose all the data X lives in a ball of radius D and a training set S is correctly classified by all classifiers*

$$\mathcal{H}_S = \{sign(\tilde{\boldsymbol{\alpha}}'\mathbf{x}) | \tilde{\boldsymbol{\alpha}} \in \mathbb{R}^n, R_{emp}(\tilde{\boldsymbol{\alpha}}) = 0\}.$$

Consider all $\tilde{\boldsymbol{\alpha}}$ whose norm is bounded by a constant A

$$\|\tilde{\boldsymbol{\alpha}}\| \leq A.$$

Then the VC dimension $c_{\mathcal{H}_S}$ of $\mathcal{H}_S$ is bounded above by

$$c_{\mathcal{H}_S} \leq \min(\lceil D^2 A^2 \rceil, n). \tag{17}$$

A proof can be found in (Burges 1998; Shawe-Taylor et al. 1996; Vapnik 1998). This theorem shows that a perceptron can overcome the curse of dimensionality even if the parameter space is very high dimensional(Bartlett 1998). The importance of this theorem lies in the fact, that minimization of the VC dimension of perceptrons can be achieved by minimizing $\|\tilde{\boldsymbol{\alpha}}\|^2 = \tilde{\boldsymbol{\alpha}}'\tilde{\boldsymbol{\alpha}}$ under the restriction that $R_{\text{emp}}(\tilde{\boldsymbol{\alpha}}) = 0$. More formally, we arrive at the problem

$$\begin{aligned} \textbf{minimize} \quad & \frac{1}{2}\|\tilde{\boldsymbol{\alpha}}\|^2 = \frac{1}{2}\tilde{\boldsymbol{\alpha}}'\tilde{\boldsymbol{\alpha}} && (18) \\ \textbf{subject to} \quad & \tilde{\boldsymbol{\alpha}}'\mathbf{x}_i \geq +1 \qquad \forall y_i = +1 && (19) \\ & \tilde{\boldsymbol{\alpha}}'\mathbf{x}_i \leq -1 \qquad \forall y_i = -1\,. && (20) \end{aligned}$$

According to the classical technique of nonlinear optimization (c.f. Hadley (1964)), we introduce ℓ lagrangian multipliers $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_\ell)'$. This yields

$$L(\tilde{\boldsymbol{\alpha}}, \boldsymbol{\lambda}) \quad = \quad \frac{1}{2}\tilde{\boldsymbol{\alpha}}'\tilde{\boldsymbol{\alpha}} - \sum_{i=1}^{\ell} \lambda_i (y_i \tilde{\boldsymbol{\alpha}}'\mathbf{x}_i - 1)\,. \qquad (21)$$

The solution is thus obtained by solving

$$(\boldsymbol{\alpha}_\ell, \boldsymbol{\lambda}_\ell) \quad = \quad \min_{\tilde{\boldsymbol{\alpha}}} \max_{\boldsymbol{\lambda} \geq 0} L(\tilde{\boldsymbol{\alpha}}, \boldsymbol{\lambda})\,. \qquad (22)$$

Setting the partial first derivatives of $L(\tilde{\boldsymbol{\alpha}}, \boldsymbol{\lambda})$ to zero, we obtain the Kuhn–Tucker conditions

$$\begin{aligned} \boldsymbol{\alpha}_\ell \quad &= \quad \sum_{i=1}^{\ell} \lambda_{i;\ell} \mathbf{x}_i y_i && (23) \\ \boldsymbol{\lambda}_\ell' \mathbf{y} \quad &= \quad 0\,. && (24) \end{aligned}$$

Substitute (23) and (24) into (21) yields the dual problem

$$\begin{aligned} \textbf{maximize} \quad & W(\boldsymbol{\lambda}) = \sum_{i=1}^{\ell} \lambda_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \lambda_i \lambda_j y_i y_j \mathbf{x}_i' \mathbf{x}_j && (25) \\ \textbf{subject to} \quad & \boldsymbol{\lambda} \geq \mathbf{0} && (26) \\ & \boldsymbol{\lambda}'\mathbf{y} = 0\,. && (27) \end{aligned}$$

This is a standard quadratic programming problem and thus learning a perceptron with the Support Vector Method arrives at finding the solution vector $\boldsymbol{\lambda}_\ell$. Note, that classification with such a network requires only the optimal vector $\boldsymbol{\lambda}_\ell$ since by virtue of Equation (23)

$$h(\mathbf{x}; \boldsymbol{\alpha}_\ell) \quad = \quad \text{sign}(\boldsymbol{\alpha}_\ell' \mathbf{x}) = \text{sign}\left(\sum_{i=1}^{\ell} \lambda_{i;\ell} y_i \mathbf{x}_i' \mathbf{x}\right)\,. \qquad (28)$$

Equation (23) states that the linear classifier is completely described by $\boldsymbol{\lambda}_\ell$ and the training set. All training points $\mathbf{x}_i$ where $\lambda_{i;\ell} \neq 0$ are called *support vectors*, because they "support" the construction of the optimal classifier. Note that only a few of the $\lambda_{i;\ell} \neq 0$ and it is this sparseness that makes (28) so appealing. Property (23) will later be exploited for application of the kernel trick.

Case of not Linearly Separable Data In the last paragraph a learning algorithm for the case of $R_{\text{emp}}(\tilde{\boldsymbol{\alpha}}) = 0$ was derived. This restriction is rather severe and can be relaxed by introducing a slack variable ξ_i for each training point that measures the violation of the constraints (19) and (20). If we use the approximation $R_{\text{emp}}(\boldsymbol{\alpha}) \approx \sum_{i=1}^{\ell} \xi_i$ we arrive at the problem

$$\begin{aligned} &\textbf{minimize} && \frac{1}{2}\|\boldsymbol{\alpha}\|^2 + C\sum_{i=1}^{\ell}\xi_i && (29)\\ &\textbf{subject to} && \boldsymbol{\alpha}'\mathbf{x}_i \geq +1 - \xi_i \qquad \forall y_i = +1 && (30)\\ &&& \boldsymbol{\alpha}'\mathbf{x}_i \leq -1 + \xi_i \qquad \forall y_i = -1 && (31)\\ &&& \boldsymbol{\xi} \geq 0\,, && (32) \end{aligned}$$

where C has to be defined before learning and is a parameter that trades the minimization of $\|\boldsymbol{\alpha}\|^2$ and the "training error" $\sum_{i=1}^{\ell}\xi_i$. Using the same technique as in the case of linearly separable data, we arrive at the dual problem

$$\begin{aligned} &\textbf{maximize} && W(\boldsymbol{\lambda}) = \sum_{i=1}^{\ell}\lambda_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\lambda_i\lambda_j y_i y_j \mathbf{x}_i'\mathbf{x}_j && (33)\\ &\textbf{subject to} && \mathbf{0} \leq \boldsymbol{\lambda} \leq C\mathbf{1} && (34)\\ &&& \boldsymbol{\lambda}'\mathbf{y} = 0\,. && (35) \end{aligned}$$

This is again a quadratic programming problem. The difference from the separable case can be seen in (26) and (34). If we set $C = \infty$, which means that we are not willing to allow any violation of the constraints (19) and (20), the learning algorithm for the case of not linearly separable data simplifies to the case of linearly separable data.

The Kernel Trick Until now, we restricted our attention to the case of perceptron learning. If we want to extend the Support Vector method to nonlinear decision functions $h(\mathbf{x};\boldsymbol{\alpha})$ we define — similar to the MLP's and RBF Networks — mappings $\mathbf{z} = f_1(\mathbf{x};\boldsymbol{\beta})$ and apply the learning technique described in the last paragraph to $\mathbf{z}$. Now taking into account that learning with the Support Vector method is equivalent to minimization of Equation (33) and classification can be carried out according to Equation (28), only the inner products $K(\mathbf{x},\mathbf{x}_i) = f_1(\mathbf{x};\boldsymbol{\beta})'f_1(\mathbf{x}_i;\boldsymbol{\beta})$ and $K(\mathbf{x}_i,\mathbf{x}_j) = f_1(\mathbf{x}_i;\boldsymbol{\beta})'f_1(\mathbf{x}_j;\boldsymbol{\beta})$ are necessary for the calculations. Therefore, instead of applying f_1 to each vector $\mathbf{x}$ we only need to replace inner products $\mathbf{x}'\mathbf{x}_i$ and $\mathbf{x}_i'\mathbf{x}_j$ in Equation (33) and (28) by the corresponding function $K(\mathbf{x},\mathbf{x}_i)$ and $K(\mathbf{x}_i,\mathbf{x}_j)$. According to the Hilbert–Schmidt theory (Courant and Hilbert 1953), *each* symmetric function $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ that satisfies the Mercer conditions (Mercer 1909), corresponds to an inner product in some space $\mathcal{F}$. This is the space, to which the predefined function $f_1(\cdot;\boldsymbol{\beta})$ maps. Such functions $K(\cdot,\cdot)$ are called *kernels*. In this sense, to extend the Support Vector method to nonlinear decision functions, kernels need to be found that can easily be calculated and at the same

Name	Kernel function	$\dim(\mathcal{F})$
linear	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i'\mathbf{x}_j$	n
polynomial	$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i'\mathbf{x}_j + 1)^\beta$	$\binom{n+\beta-1}{\beta}$
RBF	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\beta\|\mathbf{x}_i - \mathbf{x}_j\|^2)$	∞
Two–layer Neural Networks	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0\mathbf{x}_i'\mathbf{x}_j + \beta_1)$	∞

Table 1: A list of suitable kernel functions for Support Vector Networks (taken from Vapnik (1995)).

time map to an appropriate feature space $\mathcal{F}$. A list of such kernels is shown in Table 1. The following pseudo–code gives an overview of the Support Vector learning procedure (Vapnik 1982; Boser et al. 1992; Cortes and Vapnik 1995; Schölkopf 1997; Vapnik 1998).

Support Vector Network learning

```
Define $C$ (trade-off between $\|\boldsymbol{\alpha}_\ell\|^2$ and $R_{\mathrm{emp}}(\boldsymbol{\alpha}_\ell)$)
Define a kernel $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ (see Table 1)
Compute $\mathbf{Q}$, where $\mathbf{Q}_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$
Solve the QP problem: $\boldsymbol{\lambda}_\ell = \max\left[\mathbf{1}'\boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda}\right]$ subject to $\mathbf{0} \leq \boldsymbol{\lambda} \leq C\mathbf{1}$
    and $\boldsymbol{\lambda}'\mathbf{y} = 0$
Classify new $\mathbf{x}$ according to $h(\mathbf{x}) = \mathrm{sign}(\sum_{i=1}^{\ell} \boldsymbol{\lambda}_{i;\ell} y_i K(\mathbf{x}_i, \mathbf{x}))$.
```

In the case of large training sets efficient decomposition algorithms for the QP problem exist (Osuna et al. 1997a). These algorithms exploit the sparseness of the $\boldsymbol{\lambda}_\ell$ and show fast convergence on real world datasets (Osuna et al. 1997b; Joachims 1997).

Selection of kernel parameters When using the kernel trick the parameters β or (β_0, β_1) (see Table 1) have to be determined beforehand. Although there exists no automatic method to determine the optimal parameter values, the result of Theorem 1 can be used to select the best parameter values among a finite set of parameter values, e.g., the degree of the polynomial kernels. Equation (17) gives an upper bound on the VC dimension of the learned classifier. After training with the Support Vector method we can compute A or $\|\boldsymbol{\alpha}_\ell\|^2 = \boldsymbol{\alpha}_\ell'\boldsymbol{\alpha}_\ell$ utilizing Equation (23) by

$$\|\boldsymbol{\alpha}_\ell\|^2 = \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} y_i y_j \lambda_{i;\ell}\lambda_{j;\ell} K(\mathbf{x}_i, \mathbf{x}_j)\,. \tag{36}$$

Furthermore, as Theorem 1 requires to know the radius of the ball containing all the data, we can bound this quantity above by the maximum distance of a data point from the center of the data

$$D^2 \quad \le \quad \max_{i=1,\ldots,\ell} \left\| f_1(\mathbf{x}_i;\boldsymbol{\beta}) - \frac{1}{\ell}\sum_{j=1}^{\ell} f_1(\mathbf{x}_j;\boldsymbol{\beta}) \right\|^2 \tag{37}$$

which can again be calculated in terms of the inner products $K(\mathbf{x}_i, \mathbf{x}_j)$, alone. Therefore, in order to select the best kernel parameter, we fix a parameter, train a Support Vector Network, and calculate D^2 and $\|\boldsymbol{\alpha}_\ell\|^2$. The optimal kernel parameter is given by that parameter which minimizes the product of these terms.

6 Support Vector Networks for Preference Learning

In this section we want to show how Neural Networks can be applied to the problem of *preference learning*[10]. Let us again start by considering a motivating example.

Example 3 (Preference Learning). Consider an economic agent who is confronted with the task of choosing a basket of goods $\{\mathbf{x}_i\}_{i=1}^N$ amongst different alternatives. Think of $\mathbf{x}_i$ as a vector which denotes either numbers of different goods or different levels of relevant features of a certain good. The agent's task amounts to deciding whether he prefers an $\mathbf{x}_i$ to another one, i.e., he will have to order the bundles according to his *preference.* From a limited number of purchases the agent will try to infer his preferences for other baskets (i.e. for future purchases). Thus the agent has to learn his preferences as expressed in an assignment of *utilities* to feature vectors $\mathbf{x}_i$ from a limited sample. Although he may not be able to assign scores to each vector he will be able to rank the baskets (ordinal levels).

To illustrate what we call *preference learning* we denote by Y the output space. Then a particular application is given by the problem of learning the ordinal utility y_i the agent assigns to a combination of goods described by the feature vector $\mathbf{x}_i$. The problem is no longer a mere classification problem since the ordering of the utilities has to be taken into account by the learning process. The learned function should be *transitive* and *asymmetric.*

6.1 Theoretical Background

The preference learning problem The most important problem in solving preference learning problems is the definition of an appropriate loss for each decision $f(\mathbf{x};\boldsymbol{\alpha})$ whereas the true ordinal utility is given by y. Since the y's are ordinal, no knowledge is given about the difference $y - f(\mathbf{x};\boldsymbol{\alpha})$. On the other hand, the loss given in Equation (1) weights each incorrect assignment $f(\mathbf{x};\boldsymbol{\alpha})$ by the same amount and thus is inappropriate as well. This leads to the

[10] Se also the work by Tangian and Gruber (1995), Herbrich et al. (1998) or Wong et al. (1988).

problem, that no risk can be formulated which shall be minimized by a Neural Network learning algorithm.

Reformulation of the problem To overcome this drawback we now consider all pairs $(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$ of objects (e.g. combination of goods), where i denotes the ith possible permutation. If in the training set $\mathbf{x}_i^{(1)}$ has higher ordinal utility than $\mathbf{x}_i^{(2)}$, we say that $\mathbf{x}_i^{(1)}$ is *preferred* (by the customer) over $\mathbf{x}_i^{(2)}$ and denote this by the class $z_i = +1$. In turn, if the ordinal utility of $\mathbf{x}_i^{(2)}$ is higher than $\mathbf{x}_i^{(1)}$'s utility, we denote this by $z_i = -1$. We can now formulate a criterion for the optimal decision function. The optimal decision function is given by the $\boldsymbol{\alpha}_\ell$ that minimizes the probability of misclassifying pairs of objects. Therefore, if we consider decision functions on pairs of objects, we arrive at a "usual" classification problem, this time on pairs of objects.

A latent utility model To derive an Neural Network algorithm we make the assumption, that there is an unknown cardinal utility $U(\mathbf{x})$ an object $\mathbf{x}$ provides to the customer. Moreover we assume, that if $\mathbf{x}^{(1)}$ is preferred over $\mathbf{x}^{(2)}$ then $U(\mathbf{x}^{(1)}) > U(\mathbf{x}^{(2)})$, and vice versa. The advantage of such a model is the fact, that *transitivity* and *asymmetry* are fulfilled for each decision function. In terms of Statistical Learning Theory this means, that our hypothesis space is maximally reduced — we only want to learn decision functions with these properties. Since we are interested in cardinal utility functions that classify all pairs of objects correctly they have to fulfill

$$U(\mathbf{x}_i^{(1)}) > U(\mathbf{x}_i^{(2)}) \quad \Leftrightarrow \quad U(\mathbf{x}_i^{(1)}) - U(\mathbf{x}_i^{(2)}) > 0 \qquad \forall z_i = +1 \tag{38}$$

$$U(\mathbf{x}_i^{(1)}) < U(\mathbf{x}_i^{(2)}) \quad \Leftrightarrow \quad U(\mathbf{x}_i^{(1)}) - U(\mathbf{x}_i^{(2)}) < 0 \qquad \forall z_i = -1\,. \tag{39}$$

A linear model of the latent utility Let us start by making a linear model $U(\mathbf{x}; \boldsymbol{\alpha}) = \boldsymbol{\alpha}'\mathbf{x}$ of the latent utility. The last two equations become

$$\boldsymbol{\alpha}'\mathbf{x}_i^{(1)} - \boldsymbol{\alpha}'\mathbf{x}_i^{(2)} = \boldsymbol{\alpha}'(\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}) > 0 \qquad \forall z_i = +1 \tag{40}$$

$$\boldsymbol{\alpha}'\mathbf{x}_i^{(1)} - \boldsymbol{\alpha}'\mathbf{x}_i^{(2)} = \boldsymbol{\alpha}'(\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}) < 0 \qquad \forall z_i = -1\,. \tag{41}$$

According to the idea of Support Vector learning we make these constraints stronger (see Equations (19) and (20)) where $\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}$ now serves as a description of the pair of objects $(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$. In accordance with the Support Vector algorithm for classification, in order to minimize the generalization error on the pairs of objects we have to minimize $\|\boldsymbol{\alpha}\|^2$. This leads to the same algorithm as described in Section 5, this time applied to the difference vectors $\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}$.

A nonlinear model of the latent utility Since a linear model of the latent utility is often too restrictive, we want to extend the approach to nonlinear utility functions $U(\mathbf{x})$. This can be achieved by considering a mapping $f_1(\mathbf{x};\boldsymbol{\beta})$ which has to be defined beforehand. Then the constraints of the optimal classifiers on pairs of objects become

$$\boldsymbol{\alpha}'(f_1(\mathbf{x}_i^{(1)};\boldsymbol{\beta}) - f_1(\mathbf{x}_i^{(2)};\boldsymbol{\beta})) \geq +1 \qquad \forall z_i = +1 \tag{42}$$

$$\boldsymbol{\alpha}'(f_1(\mathbf{x}_i^{(1)};\boldsymbol{\beta}) - f_1(\mathbf{x}_i^{(2)};\boldsymbol{\beta})) \leq -1 \qquad \forall z_i = -1\,. \tag{43}$$

In order to learn using the Support Vector method, we have to compute the matrix $\mathbf{Q}$ where the element in the i-th row and j-th column is given by

$$\begin{aligned} \mathbf{Q}_{ij} &= z_i z_j (f_1(\mathbf{x}_i^{(1)};\boldsymbol{\beta}) - f_1(\mathbf{x}_i^{(2)};\boldsymbol{\beta}))'(f_1(\mathbf{x}_j^{(1)};\boldsymbol{\beta}) - f_1(\mathbf{x}_j^{(2)};\boldsymbol{\beta})) && (44)\\ &= z_i z_j \Big(f_1(\mathbf{x}_i^{(1)};\boldsymbol{\beta})' f_1(\mathbf{x}_j^{(1)};\boldsymbol{\beta}) - f_1(\mathbf{x}_i^{(1)};\boldsymbol{\beta})' f_1(\mathbf{x}_j^{(2)};\boldsymbol{\beta}) - \\ &\qquad f_1(\mathbf{x}_i^{(2)};\boldsymbol{\beta})' f_1(\mathbf{x}_j^{(1)};\boldsymbol{\beta}) + f_1(\mathbf{x}_i^{(2)};\boldsymbol{\beta})' f_1(\mathbf{x}_j^{(2)};\boldsymbol{\beta}) \Big)\,. && (45) \end{aligned}$$

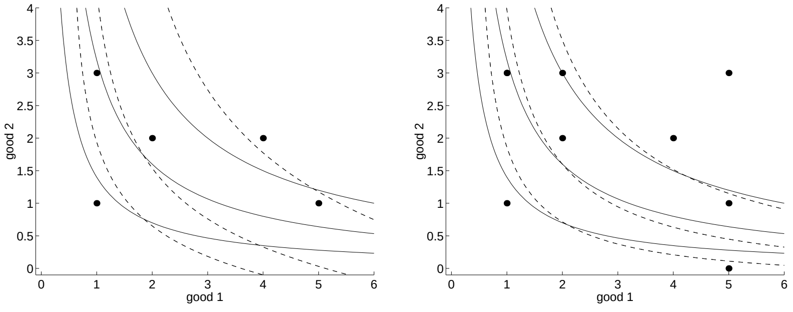
The advantage of this decomposition is the applicability of the kernel trick (see Section 5). Instead of defining $f_1(\mathbf{x};\boldsymbol{\beta})$ we replace all inner products by a function K (see Table 1) that can easily be calculated and thus learn an optimal latent nonlinear utility function. The following pseudo–code gives an overview of the learning procedure for preference relations.

Learning preference relations with Support Vector Networks

```
Define $C$ (trade-off between $\|\boldsymbol{\alpha}_\ell\|^2$ and $R_{\mathrm{emp}}(\boldsymbol{\alpha}_\ell)$)

Define a kernel $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ (see Table 1)

Compute $\mathbf{Q}$, where

    $\mathbf{Q}_{ij} = z_i z_j \Big( K(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(1)}) - K(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(2)}) - K(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}) + K(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(2)}) \Big)$

Solve the QP problem: $\boldsymbol{\lambda}_\ell = \max \left[\mathbf{1}'\boldsymbol{\lambda} - \frac{1}{2}\boldsymbol{\lambda}'\mathbf{Q}\boldsymbol{\lambda}\right]$ subject to $\mathbf{0} \leq \boldsymbol{\lambda} \leq C\mathbf{1}$
and $\boldsymbol{\lambda}'\mathbf{y} = 0$

Compute the latent utility of unseen $\mathbf{x}$ according to
$U(\mathbf{x}) = \sum_{i=1}^{\ell} \boldsymbol{\lambda}_{i;\ell} y_i (K(\mathbf{x}_i^{(1)}, \mathbf{x}) - K(\mathbf{x}_i^{(2)}, \mathbf{x}))$.
```

6.2 An Economic Application

Let us illustrate the above discussion by an example. Consider a situation where two goods compete, i.e. $\mathbf{x} = (x_1, x_2)$ is a vector that describes a basket of two goods. Assume an agent who has purchased a limited number of combinations. The agent will order these combinations according to his preferences and assign a utility level to these combinations such as to achieve the highest possible utility with the next purchase.



(a)

(b)

Figure 3: Learning of a preference structure of combinations of goods. The learned latent utility (dashed lines) is superimposed on the predefined (true) latent utility (solid lines). Training set consists **(a)** of five and **(b)** of ten observations.

To simulate this situation we generated a limited number of combinations and classified them according to an underlying true latent utility function

$$U(\mathbf{x}) \quad = \quad \frac{x_1 x_2}{2}\,, \tag{46}$$

such as to implement the agent's preference structure. Note that this utility function is ordinal in the sense that any homogenous transformation of this function would not affect the resulting order of combinations. Note also that the only given information is the set of ordered objects, i.e., we do not refer to a cardinal utility scale. Then the process of learning the utility function is simulated with a Support Vector Network that represents metaphorically the learning capacity of the agent. We assume the agent to be bounded rational, i.e., his classification procedure is not restricted to a certain type of utility function. We therefore start with a polynomial of degree 5 that is able to locally approximate any continuous function. Figure 3 shows the results for a training set of 5 goods (a) and 10 goods (b).

The dashed lines represent the learned utility function. The model selection strategy that is based on equations (36) and (37), selects an ordinal utility function of polynomial degree $\beta = 2$ out of polynomial degrees $\beta = 1, \ldots, 5$. This choice exactly corresponds to the model from which the true utility function (46) was chosen. Note that all combinations are classified correctly and how close the learned latent utility is to the unknown true latent utility.

7 Summary

After introducing some basic results from statistical learning theory, we gave an overview of the basic principles of neural network learning. We presented three commonly used learning algorithms: Perceptron learning, backpropagation learning, and radial basis function learning. Then we gave an overview of existing economic applications of neural networks, where we distinguished between three types: Classification of economic agents, time series prediction and the modelling of bounded rational agents. While according to the literature Neural Networks operated well and often better than traditional linear methods when applied to classification tasks, their performance in time series prediction was often reported to be just as good as traditional methods. Finally, choosing Neural Networks as models for bounded rational artificial adaptive agents appears to be a viable strategy, although there exist alternatives.

In Section 5 we presented a new learning method, so called Support Vector Learning, which is based on Statistical Learning Theory, shows good generalization and is easily extended to nonlinear decision functions. Finally, this alorithm was used to model a situation where a buyer learns his preferences from a limited set of goods and orders them according to an ordinal utility scale. The agent is bounded rational in that he has no previous knowledge about the form of the utility function. The working of the algorithm was demonstrated on a toy example, that illustrated the good generalization behavior and the model selection performed by the algorithm.

8 Acknowledgments

We are indebted to Prof. Ulrich Kockelkorn for fruitful discussions and Craig Saunders, Mark O. Stitson, Jason Weston, Leo Bottou, Bernhard Schölkopf and Alex Smola for providing their Support Vector implementation (Saunders et al. 1998).

References

Altman, E. L. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *Journal of Finance 23*, 589–609.

Altman, E. L., G. Marco, and F. Varetto (1994). Corporate distress diagnosis: Comparisons using linear discriminant analysis and neural networks. *Journal of Banking and Finance 18*, 505–529.

Anders, U. and O. Korn (1997). Model selection in neural networks. Technical Report 96-21, ZEW. `http://www.zew.de/pub_dp/2196.html`.

Arthur, W. B. (1993). On designing economic agents that act like human agents. *Journal of Evolutionary Economics 3*, 1–22.

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.

Beltratti, N., S. Margarita, and P. Terna (1996). *Neural Networks for Economic and Financial Modelling.* Intl. Thomson Computer Press.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford: Clarendon Press.

Blien, U. and H.-G. Lindner (1993). Neuronale Netze – Werkzeuge für Empirische Analysen ökonomischer Fragestellungen. *Jahrbücher für Nationalökonomie und Statistik 212*, 497–521.

Bosarge, W. E. (1993). Adaptive processes to exploit the nonlinear structure of financial market. In R. R. Trippi and E. Turban (Eds.), *Neural Networks in Finance and Investing*, pp. 371–402. Probus Publishing.

Boser, B., I. Guyon, and V. N. Vapnik (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152.

Brock, W. A., D. H. Hsieh, and B. LeBaron (1991). *Nonlinear Dynamics, Chaos and Instability: Statistical Theory and Economic Evidence.* MIT Press.

Brockett, P. W., W. W. Cooper, L. L. Golden, and U. Pitaktong (1994). A neural network method for obtaining an early warning of insurer insolvency. *The Journal of Risk and Insurance 6*, 402–424.

Burges, C. J. (1998). A tutorial on Support Vector Machines for pattern recognition. *Data Mining and Knowldge Discovery 2*(2).

Chatfield, C. (1993). Neural networks: Forecasting breakthrough of passing fad? *International Journal of Forecasting 9*, 1–3.

Cho, I. K. (1994). Bounded rationality, neural network and folk theorem in repeated games with discounting. *Economic Theory 4*, 935–957.

Cho, I. K. and T. J. Sargent (1996). Neural networks for econding and adapting in dynamic economies. In H. M. Amman, D. A. Kendrick, and J. Rust (Eds.), *Handbook of Computational Economics, Vol. 1*, pp. 441–470. Elsevier.

Church, K. B. and S. P. Curram (1996). Forecasting consumer's expenditure: A comparison between econometric and neural network models. *International Journal of Forecasting 12*, 255–267.

Coleman, K. G., T. J. Graettinger, and W. F. Lawrence (1991). Neural networks for bankruptcy prediction: The power to solve financial problems. *AI Review July/August*, 48–50.

Cortes, C. (1995). *Prediction of Generalization Ability in Learning Machines.* Ph. D. thesis, University of Rochester, Rochester, USA.

Cortes, C. and V. Vapnik (1995). Support Vector Networks. *Machine Learning 20*, 273–297.

Courant, R. and D. Hilbert (1953). *Methods of Mathematical Physics.* New York: Jon Wiley.

Engle, R. F. (1982). Autoregressive conditional heteroskedasticity with estimates of the variance of U.K. inflations. *Econometrica 50*, 987–1007.

Erxleben, K., J. Baetge, M. Feidicker, H. Koch, C. Krause, and P. Mertens (1992). Klassifikation von Unternehmen. *Zeitschrift für Betriebswirtschaft 62*, 1237–1262.

Fahlman, S. (1989). Faster learning variations on backpropagation: An empirical study. In *Proccedings of the 1988 Connectionist Models Summer School*, pp. 38–51. Morgan Kaufmann.

Fama, E. (1970). Efficient capital markets: A review of theory and empirical work. *Journal of Finance 25*, 383–417.

Feng, C. and D. Michie (1994). Machine learning of rules and trees. In *Machine Learning, Neural and Statistical Classification*, pp. 50–83.

Franses, P. H. and G. Draisma (1997). Regcognizing changing seasonal patterns using artificial neural networks. *Journal of Econometrics 81*, 273–280.

Granger, C. W. J. (1991). Developements in the nonlinear analysis of economic series. *Scandinavian Journal of Economics 93*, 263–281.

Grudnitzki, G. (1997). Valuations of residential properties using a neural network. *Handbook of Neural Computation 1*, G6.4:1–G6.4:5.

Hadley, G. (1964). *Nonlinear and Dynamic Programming.* London: Addison–Wesley.

Haefke, C. and C. Helmenstein (1996). Neural networks in the capital markets: An application to index forecasting. *Computational Economics 9*, 37–50.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artifical Intelligence 38*, 177–221.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation.* Macmillan College Publishing Company Inc.

Herbrich, R., T. Graepel, P. Bollmann-Sdorra, and K. Obermayer (1998). Learning a preference relation for information retrieval. In *Proceedings of the AAAI Workshop Text Categorization and Machine Learning*, Madison, USA.

Hestenes, M. and E. Stiefel (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards 49*(6), 409–436.

Hiemstra, Y. (1996). Linear regression versus backpropagation networks to predict quarterly stock market excess returns. *Computational Economics 9*, 67–76.

Hill, T., L. Marquez, M. O'Connor, and W. Remus (1994). Artificial neural network models for forecasting and decision making. *International Journal of Forecasting 10*, 5–15.

Hinton, G. (1987). Learning translation invariant recognition in massively parallel networks. In *Proceedings Conference on Parallel Architectures and Laguages Europe*, pp. 1–13. Springer Verlag.

Hopfield, J. and D. Tank (1986). Computing with neural curcuits. *Science 233*, 625–633.

Jagielska, I. and J. Jaworski (1996). Neural network for predicting the performance of credit card accounts. *Computational Economics 9*, 77–82.

Joachims, T. (1997). Text categorization with Support Vector Machines: Learning with many relevant features. Technical report, University Dortmund, Department of Artifical Intelligence. LS–8 Report 23.

Johansson, E., F. Dowla, and D. Goodmann (1992). Backpropagation learning for multilayer feedforward neural networks using the conjugate gradient method. *International Journal of Neural Systems 2*(4), 291–301.

Kaastra, I., B. S. Kermanshahi, and D. Scuse (1995). Neural networks for forecasting: an introduction. *Canadian Journal of Agricultural Economics 43*, 463–474.

Kuan, C. M. and T. Liu (1995). Forecasting exchange rates using feedforward and recurrent neural networks. *Journal of Applied Econometrics 10*, 347–364.

Lee, T. H., H. White, and C. W. J. Granger (1993). Testing for neglected nonlinearity in time series models. *Journal of Econometrics 56*, 269–290.

Luna, F. (1996). Computable learning, neural networks and institutions. University of Venice (IT), `http://helios.unive.it/~fluna/english/luna.html`.

Malkiel, B. (1992). Efficient markets hypotheses. In J. Eatwell (Ed.), *New Palgrave Dictionary of Money and Finance.* MacMillan.

Marose, R. A. (1990). A financial neural network application. *AI Expert May*, 50–53.

Martin-del Brio, B. and C. Serrano-Cinca (1995). Self-organizing neural networks: The financial state of spanisch companies. In A. P. Refenes (Ed.), *Neural Networks in the Capital Markets*, pp. 341–357. Wiley.

Meese, R. A. and A. K. Rogoff (1983). Empirical exchage rate models of the seventies: Do they fit out of sample? *Journal of International Economics 13*, 3–24.

Mercer, T. (1909). Functions of positive and negative type and their connection with the theory of integral equations. *Transaction of London Philosophy Society (A) 209*, 415–446.

Odom, M. D. and R. Sharda (1990). A neural network model for bankruptcy prediction. *Proceedings of the IEEE International Conference on Neural Networks, San Diego II*, 163–168.

Orsini, R. (1996). Esternalita locali, aspettative, comportamenti erratici: Un modello di consumo con razionalita limitata. *Rivista Internazionale di Scienze Economiche e Commerciali 43*, 981–1012.

Osuna, E., R. Freund, and F. Girosi (1997a). An improved training algorithm for Support Vector Machines. In *Proccedings of the IEEE NNSP*.

Osuna, E. E., R. Freund, and F. Girosi (1997b). Support Vector Machines: Training and applications. Technical report, Massachusetts Institute of Technology, Artifical Intelligence Laboratory. AI Memo No. 1602.

Packalén, M. (1998). Adaptive learning of rational expectations: A neural network approach. Paper presented at the 3rd SIEC workshop, Ancona. See `http://www.econ.unian.it/dipartimento/siec/HIA98/papers/Packa.zip`.

Poddig, T. (1995). Bankruptcy prediction: A comparison with discriminant analysis. In A. P. Refenes (Ed.), *Neural Networks in the Capital Markets*, pp. 311–323. Wiley.

Poggio, T. and F. Girosi (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science 247*, 978–982.

Pollard, D. (1984). *Convergence of Stochastic Processess*. New York: Springer–Verlag.

Powell, M. (1992). The theory of radial basis functions approximation in 1990. In *Advances in Numerical Analysis Volume II: Wavelets, Subdivision algorithms and radial basis functions*, pp. 105–210. Oxford University.

Raghupati, W., L. L. Schkade, and B. S. Raju (1993). A neural network approach to bankruptcy prediction. In R. R. Trippi and E. Turban (Eds.), *Neural Networks in Finance and Investing*, pp. 141–158. Probus Publishing.

Rahimian, E., S. Singh, T. Thammachote, and R. Virmani (1993). Bankruptcy prediction by neural network. In R. R. Trippi and E. Turban (Eds.), *Neural Networks in Finance and Investing*, pp. 159–171. Probus Publishing.

Refenes, A. P. (1995). *Neural networks in the Capital Markets*. Wiley.

Refenes, A. P., A. D. Zapranis, and G. Francis (1995). Modelling stock returns in the framework of APT: C comparative study with regression models. In A. P. Refenes (Ed.), *Neural Networks in the Capital Markets*, pp. 101–125. Wiley.

Ripley, B. D. (1994). Neural networks and related methods for classification. *Journal of the Royal Statistical Society 56*, 409–456.

Rosenblatt, M. (1962). *Principles of neurodynamics: Perceptron and Theory of Brain Mechanisms*. Washington D.C.: Spartan–Books.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by backpropagating errors. *Nature 323*, 533–536.

Salchenberger, L., E. Cinar, and N. Lash (1992). Neural networks: A new tool for predicting bank failures. *Decision Sciences 23*, 899–916.

Sargent, T. S. (1993). *Bounded Rationality in Macroeconomics*. Clarendon Press.

Saunders, C., M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola (1998). Support Vector Machine reference manual. Technical report, Royal Holloway, University of London. CSD–TR–98–03.

Schölkopf, B. (1997). *Support Vector Learning.* Ph. D. thesis, Technische Universitä Berlin, Berlin, Germany.

Shawe-Taylor, J., P. L. Bartlett, R. C. Williamson, and M. Anthony (1996). Structural risk minimization over data–dependent hierarchies. Technical report, Royal Holloway, University of London. NC–TR–1996–053.

Swanson, N. R. and H. White (1997). A model selection approach to real-time macroecoomic forecasting using linear models and artificial neural networks. *The Review of Economics and Statistics LXXIX*, 540–550.

Tam, K. Y. (1991). Neural networks and the prediction of bank bankruptcy. *OMEGA 19*, 429–445.

Tam, K. Y. and Y. M. Kiang (1992). Managerial applicationf of neural networks: The case of bank failure predictions. *Management Science 38*, 926–947.

Tangian, A. and J. Gruber (1995). Constructing quadratic and polynomial objective functions. In *Proceedings of the 3rd International Conference on Econometric Decision Models*, Schwerte, Germany, pp. 166–194. Springer.

Trippi, R. R. and E. Turban (1990). Auto learning approaches for building expert systems. *Computers and Operations Research 17*, 553–560.

Tsibouris, G. and M. Zeidenberg (1995). Testing the efficient markets hypotheses with gradient descent algorithms. In A. P. Refenes (Ed.), *Neural Networks in the Capital Markets*, pp. 127–136. Wiley.

Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data.* New York: Springer–Verlag.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory.* New York: Springer–Verlag.

Vapnik, V. (1998). *Statistical Learning Theory.* New York: John Wiley and Sons.

Vapnik, V. and A. Chervonenkis (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Application 16*(2), 264–281.

Verkooijen, W. (1996). A neural network approach to long-run exchange rate prediction. *Computational Economics 9*, 51–65.

Weigend, A. S., B. A. Huberman, and D. E. Rumelhart (1992). Predicting sunspots and exchange rates with connectionist networks. In M. Casdagli and S. Eubank (Eds.), *Nonlinear Modeling and Forecasting*, pp. 395–432. SFI Studies in the Science of Complexity, Proc. Vol. XII.

White, H. (1988). Economic prediction using neural networks: The case of IBM daily stock returns. *Proceeding of the IEEE International Conference on Neural Networks II*, 451–458.

Wolpert, D. H. (1995). *The Mathematics of Generalization*, Chapter 3,The Relationship between PAC, the Statistical Physics Framework, the Bayesian Framework, and the VC framework, pp. 117–215. Addison Wesley.

Wong, F. S. (1990). Time series forecasting using backpropagation neural networks. *Neurocomputing 2*, 147–159.

Wong, S. K. M., Y. Y. Yao, and P. Bollmann (1988). Linear structure in information retrieval. In *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 219–232.