Support Vector Neural Networks

Thilo-Thomas Frieß and Rob Harrison

Date: 14. September 1998

Research Report No. 725

The University of Sheffield,
Dept. of Automatic Control and Systems Engineering
Mappin Street, Sheffield, S1 3JD, England
email: friess@acse.shef.ac.uk

Abstract: The kernel Adatron support vector neural network (SVNN) is a new neural network alternative to support vector (SV) machines. It can learn large-margin decision functions in kernel feature spaces in an iterative "on-line" fashion which are *identical* to support vector machines. In contrast "conventional" support vector learning is batch learning and is strongly based on solving constrained quadratic programming problems. Quadratic programming is nontrivial to implement and can be subject to stability problems.

The kernel Adatron algorithm (KA) has been introduced recently. So far it has been assumed that the bias parameter of the plane in feature space is always zero, and that all patterns can be correctly classified by the learning machine. These assumptions cannot always be made. The kernel Adatron SVNN with bias and soft margin combines the speed and simplicity of neural networks with the predictive power of SV machines. However the SVNN does not, unlike to SV machines, suffer from any problems related to quadratic programming, and unlike to conventional neural networks the SVNN's cost function is always *convex*.

The support vector neural network is introduced, then experimental results using benchmarks and real data are presented which allow to compare the performance of SVNN's and SV machines.

Key Words: Neural Networks, Perceptron, Adatron, Kernel Functions, Method of Potential Functions, Statistical Mechanics, Support Vector Learning

1 Introduction

Support vector (SV) machines, developed by Vapnik and co-workers ([5], [7]), are known since 1995. There seems to be experimental evidence that this algorithm has the capability to obtain a high generalisation ability (e.g. [34], [41]). Unlike to neural networks SV machines are still not widely used in the community. SV learning is strongly based on quadratic programming which is not easy to implement, in particular for large-scale problems. In cases where the hessian is close to singular optimizers may fail to find solutions at all, in other cases sometimes only sub-optimal solutions can be found. In support vector learning most elements of the solution vector are zero (theoretically), however most optimizers return solutions where most components have very small positive or negative values. Users usually help theirselves by defining a small threshold and to cut all components which have an absolute value smaller than the threshold. This is dangerous because the loss caused by this effect depends on the input distribution which is not known. If the computer's memory is not large enough to store the hessian matrix decomposition strategies (like chunking) are required.

The kernel Adatron SVNN is an algorithm which allows to overcome these limitations. It can learn decision surfaces in kernel feature spaces and find *exact* solutions without explicitly solving a constrained quadratic programming problem. The algorithm is numerically highly stable and extremely simple.

Since the SVNN is a neural network algorithm this work implicitly establishes a link between (more cognitive oriented) neural network studies and a machine learning algorithm motivated mainly by ideas developed in a statisitcal context.

This document is structured as follows: In section one Rosenblatt-perceptrons, data dependent perceptrons, kernel perceptrons, and SV machines are considered. In section two the Adatron and kernel Adatron is considered, then the support vector neural network with bias and soft margin is introduced. It is shown how training patterns can be augmented in the feature space in order to learn the bias parameter. Section three shows computer experiments which allow to compare SVNN's and SV machines in four cases.

2 Perceptron Learning

2.1 Rosenblatt's Perceptrons

Rosenblatt's learning algorithm for perceptrons [38] can find a linear discriminant function, $f(x)$, for a given set of labelled training patterns. Each training pattern, x_i, is a vector in R^d and has a label $y_i \in \{+1, -1\}\ \forall i \in \{1..l\}$.
The vector w of the linear decision function

$$f(x) = \langle w, x \rangle \tag{1}$$

can be found by the following algorithm:

Rosenblatt-Perceptron

1. Choose a starting point (e.g. $w_d = 0 \; \forall d$)
2. WHILE not all patterns are correctly classified
3. choose pattern x_i ($i \in \{1..l\}$)
4. If NOT $sign(f(x_i)) = y_i$ update w by: $w \leftarrow w + \eta x_i y_i$ END IF
5. END WHILE

For the learning rate, η, usually a small positive value is chosen (or alternatively η is set to 1 and training patterns are re-scaled). Similar algorithms (e.g. neural networks) sometimes use an adaptive learning rate which can vary during the learning process.
If it is possible to classify all training patterns correctly, the algorithm terminates in a finite number of cycles. Then a weight vector, w, has been found such that $y_i = sign(f(x_i)) \; \forall i$. The weight vector, w, may be expressed as a weighted sum of patterns: $w = \sum_{i=1}^{l} \beta_i x_i$ where the β_i are multipliers for individual patterns, x_i. Another way to expand the vector, w, is given by: $w = \sum_{i=1}^{l} y_i \alpha_i x_i$, in this case the α_i are non-negative integers.

Similar to perceptrons, the expansion of w onto a weighted sum of some training patterns also holds for Vapnik's support vector (SV) machines - therefore SV machines can be regarded as a species of the Rosenblatt-perceptron, or alternatively Rosenblatt's-perceptron as a special form of a "support pattern" machine.

2.2 Perceptrons in the Data Dependent Representation

Decision function (1) can be rewritten in expanded form:

$$f(x) = \langle w, x \rangle = \langle (\sum_{i=1}^{l} \alpha_i x_i), x \rangle = \sum_{i=1}^{l} \alpha_i \langle x_i, x \rangle \tag{2}$$

and Rosenblatt's update rule (as described above) can be rewritten as:

$$\alpha_i \leftarrow \alpha_i + \eta y_i \tag{3}$$

Note that in (3) the elements of the vector of the multipliers, α, are updated instead of updating the weight vector, w, directly. Decision functions of the form (2) will be called data dependent representation (because the weight vector has been expanded on some data points), while decision functions of the form (1) will be called explicit representation (w is directly available).
To avoid target function ((1),(2)) passing through the origin augmented training patterns, x^a can be used. Then an affine function: $f(x) = \langle w, x \rangle + b = \langle w^a, x^a \rangle$ will be obtained which has one more degree of freedom:

$$\forall 1 \in \{1..d\} \; w_i^a = w_i, \; w_{d+1}^a = b \tag{4}$$

A pattern can be augmented by simply increasing its dimensionality by one and setting the $(d+1)$'th component of the augmented pattern vector to 1. Note that augmenting patterns is similar to introducing a bias unit which always has an activation equal to one.

To use augmented patterns in a data dependent perceptron it is required to replace (3) by the following update rule:

$$\alpha_i \leftarrow \alpha_i + \eta y_i \;\; ; \;\; b \leftarrow b + \eta y_i \tag{5}$$

The first part of this rule ($\alpha_i \leftarrow \alpha_i + \eta y_i$) ensures that the first d components in weight vector w^a are updated correctly; while the second part ($b \leftarrow b + \eta y_i$) ensures that the $(d+1)$'th component of w^a is also updated.
It is important to realize that in this representation only the $(d+1)$'th component of the augmented weight vector w^a is explicitly available, while all other d components are only implicitly given by the data dependent expansion on the training vectors.

2.3 Kernel Functions and Nonlinear Perceptrons

The perceptron with kernels, known as the method of potential functions [1], has the ability to learn nonlinear decision functions. The algorithm is based on the idea of nonlinear kernel functions which represent dot products in some Hilbert spaces. A kernel function allows the mapping of two patterns (x_u, x_v) at first into a high dimensional feature space $(\phi(x_u), \phi(x_v))$, and then the calculation of a dot product there. This is expressed by:

$$k(x_u, x_v) = \langle \phi(x_u), \phi(x_v) \rangle = \langle z_u, z_v \rangle \tag{6}$$

where the z are images of patterns x in feature space.
Any function k which satisfies Mercer's conditions may be used as a dot product in kernel-feature space [6].
Initially the kernel idea was applied in the potential function algorithm; an algorithm which uses kernel dot products to run a perceptron in a linearization space - the kernel feature space.
There has been an increased interest in using kernel dot products to create nonlinear algorithms. Examples are support vector machines [7], kernel principal component analysis [41], linear programming machines [14], ridge regression in dual variables [40], and kernel clustering [21]. It is also possible to shine some new light onto gaussian-based algorithms by considering Gaussians as dot products in some feature space [10].
Examples for kernel functions are the scalar product, the radial basis function (RBF) kernel, and the polynomial kernel:

$$k_{sp}(x_u, x_v) = \langle x_u, x_v \rangle \tag{7}$$

$$k_{RBF}(x_u, x_v) = exp(-||x_u - x_v||^2/\sigma^2) \tag{8}$$

$$k_{pol}(x_u, x_v) = (\langle x_u, x_v \rangle + 1)^d, \; d = 1, 2, .. \tag{9}$$

Kernel-based algorithms are elegant in the sense that the "kernel trick" allows algorithms to operate implicitly in very high (sometimes infinite-) dimensional feature spaces without explicitly expanding patterns into their feature space representation. Undesired side effects have been observed using support vector machines with polynomial kernels which are not scale or shift invariant. In these learning machines the optimal margin decision function will not lie in the middle of the two classes ([5], [13]). Further information about

kernel functions can be found in [41].

To plug kernels into Rosenblatt's algorithm the dot products from the data-dependent decision function (2) are simply replaced by kernel functions:

$$f(x) = \langle w, \phi(x) \rangle = \langle \sum_{i=1}^{l} \alpha_i \phi(x_i), \phi(x) \rangle = \sum_{i=1}^{l} \alpha_i k(x_i, x) \tag{10}$$

On the right hand side of (10) the weight vector, w, resides now in the feature space defined by the kernels, it cannot be accessed any more for updates. Recall that the dot product in feature space is available, but not the transform $\phi(x) = z_x$ for a pattern x. However in the data dependent representation of the kernel perceptron, multipliers are still accessible, therefore the learning scheme is now given by:

Kernel-Perceptron Learning (Method of Potential Functions)

1. Choose a starting point (e.g. $\alpha_i = 0 \; \forall i \in 1..l$)
2. WHILE not all patterns are correctly classified (using decision function (10))
3. choose pattern x_i ($i \in \{1..l\}$)
4. If NOT $sign(f(x_i)) = y_i$ update w by $\alpha_i \leftarrow \alpha_i + \eta y_i$ END IF
5. END WHILE

The b-parameter can be introduced in this perceptron by using update rule (5) instead of the rule suggested in step 4 and by using decision function: $f(x) = \sum_{i=1}^{l} \alpha_i k(x_i, x) + b$ instead of (10). This is equivalent to *augmenting the training patterns, $z_i = \phi(x_i)$, in kernel feature space* by one dimension. Then an augmented ($d + 1$ dimensional) weight vector w^a will be used instead of a d dimensional vector w.

2.4 Data Dependent Decision Functions

Owing to the data dependent representation of the weight vector in kernel perceptrons linear dependencies between training vectors x_i in feature space $\phi(x_i)$ may exist. Then a variety of data dependent perceptrons may represent exactly one weight vector in feature space.

In the following subsection it is shown how linear depenencies between patterns in feature space can be identified, such that data dependent perceptrons can be reduced to a minimal expansion. This result is different to the work by [36], where it is pointed out *that* linear dependencies in kernel feature space exist. Here it is shown *how* these dependencies in feature space can be identified in order to remove redundancies and compress the perceptrons decision funcion, regardless of the choice of regularisation parameters (if existing as in SV machines).

Data dependent kernel perceptrons use a decision function of the general form:

$$f(x) = \langle w, \phi(x) \rangle + b = \sum_{i=1}^{l^{sv}} \beta_i k(x_i^{sv}, x) + b \tag{11}$$

In this expansion all patterns with a non-zero multiplier, β, are denoted as "sv" patterns, there are l^{sv} such patterns ($l^{sv} \leq l$). As stated above, in data dependent perceptrons the weight vector, w, exists only in feature space; it lies in the subspace spanned by the vectors $\phi(x^{sv})$. In ([7], Endnote 6) it is explained that in support vector machines sometimes more patterns, x^{sv}, are found than necessary to expand w in feature space (*"the decision function is unique but not its expansion on support vectors"*).
It is a nontrivial task to eliminate the linear relationships in feature space because transform $\phi(x)$ is not available. Even if this transform were available it would be nearly impossible to calculate linear dependencies between the vectors, $\phi(x^{sv})$, because these vectors can have a very high number (possibly infinite) of dimensions.
If linear dependencies in feature space could be found, their images in input space could be removed from the decision function in a straightforward manner. Assuming that multipliers from the decision function are updated correctly, patterns from expansion (11) can be removed *without* loss.

Matrix P will be defined such that $P_{g,h}$ is the h'th component of vector $\phi(x_g)$; $g \in \{1..l^{sv}\}$. Note that matrix, P, is an l^{sv} by d^{fs} matrix where fs denotes the dimensionality of the feature space.
It is useful to define an l^{sv} by l^{sv} kernel correlation matrix $D = PP'; D_{i,j} = k(x_i, x_j)$ which has the following property:

$$rank(D) = rank(P) \tag{12}$$

This relationship is a property of a Gram matrix [22].
Except for cases where $rank(D) = l^{sv}$ there will be patterns $\phi(x_i)$ which can be expressed as a linear combination of l^{*sv} other patterns in feature space ($l^{*sv} \leq l^{sv}$).

$$\phi(x_i) = \sum_{j^*=1}^{l^{*sv}} \gamma_{j^*} \ \phi(x_{j^*}) \quad (i \neq j^* \ \forall j^*) \tag{13}$$

Linear dependencies between row-vectors of D are defined by:

$$\forall h \in \{1..l^{*sv}\} \ \ D_{i,h} = \sum_{j^*=1}^{l^{*sv}} \delta_{j^*} D_{j^*,h} \tag{14}$$

It can be shown that the linear dependencies between the row vectors in D are exactly the same as the linear dependencies between the row vectors in P.

Theorem: *For each linear dependency of the form (13) there exists exactly one linear dependency in D where the linear factors in (13) and (14) are equivalent*:

$$\forall i \ \ \gamma_i = \delta_i \tag{15}$$

Proof:

$$\forall_h : D_{i,h} = k(x_i, x_h) = \langle \phi(x_i), \phi(x_h) \rangle = \langle \sum_{j^*=1}^{l^{*sv}} \gamma_{j^*} \phi(x_{j^*}), \phi(x_h) \rangle = \tag{16}$$

$$\sum_{j^*=1}^{l^{*sv}} \gamma_{j^*} k(x_{j^*}, x_h) = \sum_{j^*=1}^{l^{*sv}} \gamma_{j^*} D_{j^*,h}$$

Therefore it follows that $\gamma_i = \delta_i \ \forall i$. This relationship allows the computation of the γ from matrix D, this leads to the δ which can be used to update the multipliers, β, if a pattern, x_i, is to be removed from the decision function.
All that's required is to find the γ is to solve linear equation systems where one row vector of D is expressed as a linear combination of other row vectors of D. This can be done using standard techniques for solving linear equation systems (e.g. Gauss elimination or Householder's method [26]).
Once linear dependencies, δ, in feature space have been found the decision function can be rewritten in a way such that one redundant (linearly dependent) pattern will vanish. Function (11) can be expanded:

$$f(x) = b + \sum_{i=1}^{l^{sv}} \beta_i k(x_i, \phi(x)) = \tag{17}$$
$$b + \beta_1 \langle z_1, \phi(x) \rangle + \beta_2 \langle z_2, \phi(x) \rangle + ... \beta_i \langle z_i, \phi(x) \rangle + ... \beta_{l^{sv}} \langle z_{l^{sv}}, \phi(x) \rangle$$

By using (13) and multiplying out

$$f(x) = b + \beta_1 \langle z_1, \phi(x) \rangle + \beta_2 \langle z_2, \phi(x) \rangle + ...$$
$$\beta_i \langle (\sum_{j^*=1}^{l^{sv}} \gamma_{j^*} z_{j^*}), \phi(x) \rangle + ... \beta_{l^{sv}} \langle z_{l^{sv}}, \phi(x) \rangle$$

$$= b + \beta_1 \langle z_1, \phi(x) \rangle + \beta_2 \langle z_2, \phi(x) \rangle + ... (\beta_j + \beta_i \gamma_j) \langle z_j, \phi(x) \rangle + ... \tag{18}$$
$$(\beta_k + \beta_i \gamma_k) \langle z_k, \phi(x) \rangle + ... \beta_{l^{sv}} \langle z_{(l^{sv}-1)}, \phi(x) \rangle$$

a function is obtained where one unit has been removed from the expansion.

So the update rule for multipliers β from (11) is given by:

$$if \ k \neq i \quad u(\beta_k) = \beta_k + \beta_i \gamma_{k^*} \tag{19}$$

In an algorithm which aims to compress the decision function without loss in each iteration a linear dependency of form (13) can be identified; then one pattern is removed from the decision function, and multipliers β are updated.

As mentioned in [39] Vapnik's leave-one-out bound works only in cases where all patterns are correctly classified by a support vector machine. In those cases the "compression ratio" (expected generalisation ability) given by the leave-one-out bound [46] is an upper bound on the compression function: $T_{MESR} = rank(P)/l$.

It has been pointed out recently [36] that in support vector machines with RBF kernels $rank(D) = l^{sv}$ is always satisfied, therefore in this kernel feature space there will be, unlike to polynomial feature spaces, no linear dependencies.

3 Support Vector (SV) Machines

SV machines ([5], [7], [46]) are data dependent kernel perceptrons using decision function (11). The perceptron found by the SV machine algorithm is the one which has the largest possible margin p betwen the decision plane and patterns closest to the plane.
There are different explanations why the large margin perceptron performs well. Independently in distict frameworks (e.g. statistical learning theory [46], learning by regularisation in the frequency domain [4], and statistical mechanics of neural networks [18]) a link between the large margin property of a learning machine an its generalisation has been established.
In this subsection the SV machine algorithm will be reviewed. Further information on SVM's can be found in e.g.: [46], [8], [7].
SV machines are perceptrons of the form:

$$\langle w_o, x \rangle + b_o = 0 \tag{20}$$

where all training patterns are classified such that:

$$y_i(\langle w, x_i \rangle + b) \geq 1, \ i = 1, ..., l \tag{21}$$

assuming that it is possible to classify all patterns correctly. If this assumption cannot be made an error function must be used (below).
The margin p is expressed as the distance between the training patterns closest to the plane:

$$p(w, b) = \min_{\{x_i : y_i = 1\}} \frac{\langle w, x \rangle}{|w|_2} - \max_{\{x : y = -1\}} \frac{\langle w, x \rangle}{|w|_2} \tag{22}$$

The perceptron with the maximum-margin plane is the one which minimizes the L_2 norm of the weight vector subject to (21):

$$p(w_o, b_o) = \frac{2}{|w_o|_2} = \frac{2}{\sqrt{\langle w_o, w_o \rangle}} \tag{23}$$

To maximise the margin a Lagrangian is formed:

$$L(w, b, \alpha) = \frac{1}{2}\langle w, w \rangle - \sum_{i=1}^{l} \alpha_i (y_i(\langle w, x_i \rangle + b) - 1) \tag{24}$$

where the α_i are nonnegative Lagrangian multipliers for the constraints.

At the point which minimises L with respect to w and b the following constraints are satisfied:

$$\frac{\delta L(w, b, \alpha)}{\delta w} = (w - \sum_{i=1}^{l} \alpha_i y_i x_i) = 0 \tag{25}$$

$$\frac{\delta L(w, b, \alpha)}{\delta b} = \sum_{i}^{l} y_i \alpha_i = 0 \tag{26}$$

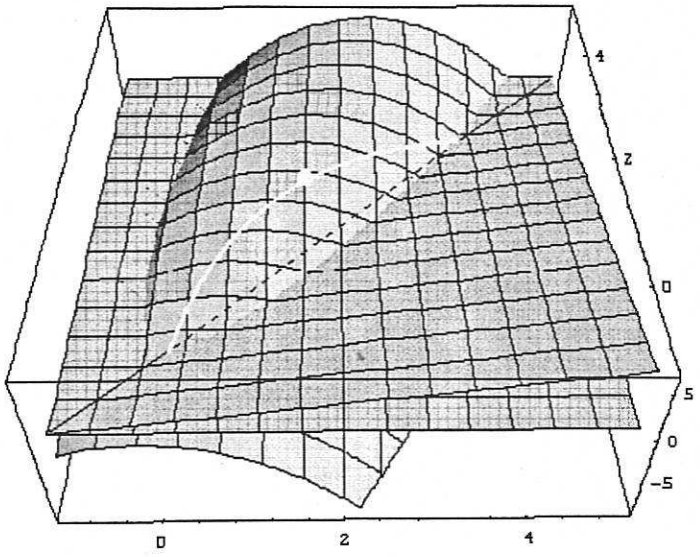

Figure 1: Visualisation of the quadratic optimisation problem (27) for a trivial two-pattern example (the diagram shows the corresponding maximisation problem). The cost function and the linear constraints are illustrated by the hyperbola and the two planes. Where the diagonal black line lies in the positive quadrant all constraints are satisfied. When solving the problem it is necessary to ensure that the feasible point lies in the constraint-space (here on the black line and in the positive quadrant) and maximises the corresponding (projected) value of the quadratic form.

Following [7] this leads to an l dimensional quadratic form:

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{27}$$

which must be optimised subject to constraint:

$$\alpha_i \geq 0 \ \forall i \in \{1..l\} \tag{28}$$

and subject to (26).
For nonlinear classification functions the dot product in (27) is replaced by a kernel function which leads to:

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{29}$$

which again can be optimised subject to (26) and (28).
Expression (29) can be written is vector notation:

$$W(\lambda) = \langle \lambda^T, 1 \rangle - \frac{1}{2} \lambda^T D \lambda \tag{30}$$

where 1 is an l dimensional unit vector, λ an l dimensional vector of multipliers α, and correlation matrix D is given by: $D_{i,j} = y_i y_j k(x_i, x_j)$ (as defined above).

It is common practice to use methods of mathematical programming to solve this constrained l dimensional quadratic programming problem, therefore SV machines implement a form of batch learning. Clearly, for a high number of patterns in the training set the optimisation problem to solve will be of a high dimensionality, and therefore hard to solve. This problem has been perceived recently, some algorithms to attack this problem have been proposed ([35], [23]).

To allow to cope with cases where not all training patterns can be correctly classified slack variables have been introduced; then expression (21) is replaced by:

$$y_i(\langle w, x_i \rangle + b) \geq 1 - \epsilon_i, \quad \forall i \in \{1, .., l\} \tag{31}$$

An error function (soft margin) can be introduced [7]:

$$\chi = (\sum_{i=1}^{l} \epsilon^{\tau})^{\psi} \tag{32}$$

which allows to implement different noise-models by choosing values for τ and ψ (Cortes and Vapnik give the quadratic form to optimize for choice $\tau = 1$,$\psi = 2$ (mean square error MSE) and for choice $\tau = 2$, $\psi = 1$ (least square error LSE)). Then it is required to optimize the functional:

$$\zeta = \frac{1}{2}\langle w, w \rangle + C(\sum_{i=1}^{l} \epsilon^{\tau})^{\psi} \tag{33}$$

which leads to a quadratic optimisation problem for the two choices of τ and ψ mentioned above. Also for the choice $\tau = 1$ and $\psi = 1$ (L_1 error function) the corresponding optimisation problem is quadratic [7].
To balance the ammount of error minimisation versus weight decay regularisation in the feature space (capacity control) the scalar C was introduced.

For the mean square error function (MSE) the optimisation problem to solve is given by:

$$W(\lambda, \kappa) = \langle \lambda^T, 1 \rangle - \frac{1}{2}(\lambda^T D \lambda + \frac{\kappa^2}{C}) \tag{34}$$

subject to constraints: $\langle \lambda^T, Y \rangle = 0$, $\kappa \geq 0$, $0 \leq \lambda_i \leq \kappa \; \forall i \in \{1..l\}$ (κ is a constant).

For the least square error function (LSE) the quadratic form is given by:

$$W(\lambda) = \langle \lambda^T, 1 \rangle - \frac{1}{2}(\lambda^T D \lambda + \frac{1}{C}\lambda^T \lambda) \tag{35}$$

subject to constraints: $\lambda \geq 0$, and $\langle \lambda^T, Y \rangle = 0$.

If the L_1 error fuction is used support vector machines minimize a quadratic form (29); subject to constraints: $0 \leq \alpha_i \leq C$ ($i \in \{1..l\}$), and subject to $\sum_{i=1}^{l} \alpha_i y_i = 0$.

Further details about SV machines and complexity regularisation by the C parameter can be found in ([8], [46]).

4 Kernel Adatron Support Vector Neural Networks

4.1 Adatron

The Adatron is a neural network algorithm [24] which can learn perceptrons (in explicit form) with the largest possible margin between the decision plane and patterns closest to the plane. During learning the following quadratic form is optimized ([15], [2]):

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{36}$$

subject to:

$$\alpha_i \geq 0 \ \forall i \in \{1..l\} \tag{37}$$

Therefore the Adatron's solution is identical to the one found by support vector machines assuming that the plane to learn has no bias parameter b [2].
The Adatron's learning algorithm is given by:

Adatron (rewritten in the data dependent representation)

define:

$$f_{Ad}(x) = y_i \sum_{j=1}^{l} y_j \alpha_j \langle x_i, x_j \rangle$$

$$M_{Ad} = \min_{i \in \{1..l\}} f_{Ad}(x_i)$$

1. Choose a starting point (e.g. $\alpha_i = 0.1 \ \forall i \in \{1..l\}$), choose a learning rate η, and choose a very small threshold t.
2. WHILE $M_{Ad} \geq t$
3. choose pattern x_i $(i \in \{1..l\})$
4. calculate a proposed update: $p_u = \eta(1 - f_{Ad}(x_i))$
5. IF $((\alpha_i + p_u) > 0)$ $\alpha_i \leftarrow \alpha_i + p_u$ END IF
6. END WHILE

In the seventies a highly similar algorithm has been studied in a more statistical context [45]. There it is pointed out that this perceptron algorithm performs a gradient descent in the quadratic cost function. In each step the gradient of the cost function in direction of one canonical basis vector is computed, then the feasible solution is updated. The Adatron's learning process therefore implements a form of gradient descend in the convex cost-function space defined by (36).

4.2 Kernel Adatron (KA)

The kernel Adatron algorithm, which has been proposed recently [15], is a nonlinear version of the Adatron.
Since it has been shown that the Adatron can be rewritten in the data dependent representation, it is possible to replace dot products by kernel functions.

All thats required is to replace the linear function $f_{Ad}(x)$ from the algorithm above by the following nonlinear function:

$$f_{Ad}(x) = y_i \sum_{j=1}^{l} y_j \alpha_j k(x_i, x_j) \tag{38}$$

The computation of function $f_{Ad}(x)$ can be performed efficiently by calculating a kernel correlation matrix D as defined above (the cache matrix). If a cache matrix is available function $f_{Ad}(x_i)$ $\forall i \in \{1..l\}$ can be implemented in an elegant way by computing only one dot product:

$$f_{Ad}(x) = y_i \langle (\alpha \circ y), D^j \rangle \tag{39}$$

where the operator $\circ$ represents the element-wise multiplication of two l dimensional vectors and D^j represents row-vector j of matrix D.

It is easy to see that the kernel Adatron minimises the same cost function (in kernel feature space) than the Adatron, therefore it also converges to the large-margin solution (now in the feature space).

4.3 Kernel Adatron with Bias Unit (KAb)

It has been assumed so far in the Adatron and kernel Adatron algorithm that the plane to learn will always pass through the origin; this assumption cannot be made in any case. Therefore the data dependent Adatron and kernel Adatron has been extended by a bias parameter. In this kernel Adatron weight vector w is not accessible, it resides in feature space. Recall that it is possible to access and update the b-parameter explicitly (as discussed above in section 2.3 for the kernel perceptron with bias).

The gradient descent performed by this algorithm will lead to a solution which is guaranteed to be *identical* to the solution obtained by support vector machines. This is interesting because the kernel Adatron with bias implements gradient descent in an *unconstrained* $l+1$ dimensional quadratic form (in the positive quadrant[1]) while support vector machines perform a *constrained* optimisation of an l dimensional quadratic form in the positive quadrant. The additional constraint which must be used during learning in SV machines (26) is caused by the way in which the bias term b has been implemented in the support vector algorithm.
It can be expected that a gradient descent in the positive quadrant without constraints is simpler and faster to implement than gradient descent subject to a linear constraint. This explains conceptually why the kernel Adatrons with bias unit has the ability to learn very fast.

[1]The solution vector will lie in the positive quadrant with respect of the first l components (the multipliers α), while the $(l+1)$'th component,b, can be positive or negative.

Kernel Adatron with Bias (KAb)

define:

$$f_{Ad}(x) = y_i(\sum_{j=1}^{l} y_j \alpha_j k(x_i, x_j) + b)$$

$$M_{Ad} = \min_{i \in \{1..l\}} f_{Ad}(x_i)$$

1. Choose a starting point[2] (e.g. $\alpha_i = 0 \; \forall i \in \{1..l\}$), choose a learning rate η, choose an initial value for b (e.g. $b = 0$), and choose a very small threshold t.
2. WHILE $M_{Ad} \geq t$
3. choose pattern x_i $(i \in \{1..l\})$
4. calculate a proposed update: $p_u = \eta(1 - f_{Ad}(x_i))$
5. IF $(\alpha_i + p_u) > 0)$ $\alpha_i \leftarrow \alpha_i + p_u$
$b \leftarrow b + y_i p_u$
END IF
6. END WHILE

When support vector machines have found a solution constraint:

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{40}$$

is always satisfied. Since the KAb's quadratic form is $l+1$ dimensional constraint:

$$\sum_{i=1}^{l} \alpha_i y_i = b \tag{41}$$

is satisfied at any feasible point. This constraint can be used (as done in the experiments below) to check the feasibility of the solutions found by the kernel Adatron.

4.4 Kernel Adatron SVNN with Bias and Soft Margin

Support Vector machines have the capability to cope with outliers and misclassified patterns by an error function (soft margin). Weight C on the error function allows to contol the capacity of the classifier, therefore parameter C can be regarded as a regularization parameter.
Let's consider the quadratic form for LSE again:

$$W(\lambda) = \langle \lambda^T, 1 \rangle - \frac{1}{2}(\lambda^T D \lambda + \frac{1}{C}\lambda^T \lambda) \tag{42}$$

subject to constraints: $\lambda \geq 0$, and $\langle \lambda^T, Y \rangle = 0$. For simplicity it will be assumed that a cache matrix is used in the kernel Adatron, then the kernel Adatron's cache matrix $D_{i,j} = k(x_i, x_j)$ has to be replaced by matrix E^{LSE}:

$$if \;\; i = j \;\; E^{LSE}_{i,j} = D(i,j) + 1/C \;\; else \;\; E^{LSE}_{i,j} = D(i,j)$$

[2]The suggested choice is a good choice in practice. Clearly, the kernel Adatron will converge for any choice owing to the convex cost space.

As usual in support vector learing, parameter C has to be chosen *a priori*.

The final algorithm can now be stated as:

Kernel Adatron with Bias and LSE Error Function (SVNN):

define:

$$f_{Ad}(x) = y_i(\langle(\alpha \circ y), E^{LSE\ j}\rangle + b)$$

$$M_{Ad} = \min_{i \in \{1..l\}} f_{Ad}(x_i)$$

1. Choose a starting point (e.g. $\alpha_i = 0\ \forall i \in \{1..l\}$), choose a learning rate η, choose an initial value for b (e.g. $b = 0$), and choose a very small threshold t.
2. WHILE $M_{Ad} \geq t$
3. choose pattern x_i $(i \in \{1..l\})$
4. calculate a proposed update: $p_u = \eta(1 - f_{Ad}(x_i))$
5. IF $(\alpha_i + p_u) > 0)$ $\alpha_i \leftarrow \alpha_i + p_u$
   $b \leftarrow b + y_i p_u$
   END IF
6. END WHILE

The algorithm is also given in the Appendix in MATLAB-pseudocode.

Obviously, this algorithm performs a gradient descent in the quadratic form (42). It is straightforward to argue backwards and to show that the SVNN implements a support vector machine with capacity control and LSE error function.
Another interesting approach to implement a soft margin and capacity control would be to choose a very small initial starting point (e.g. $\alpha_i = 0\ \forall i$ and $b = 0$), and to stop the learning process after a fixed number of iterations (an early stopping approach). As pointed out by Cristianini [9] this can be done more systematically by bounding the values of α into a box (the so called box constaint); this implements a linear (L_1) error function in the kernel Adatron ($\tau = 1$, $\psi = 1$).

Finally it should be mentioned that in the Adatron and its nonlinear version all multipliers go exactly to zero or to a value which is a pattern's corresponding multiplier. This is a positive effect compared to the numerical inaccuracies observed in quadratic optimizers.

5 Computer Experiments

Aim of the experiments presented in this section is to compare the performance of the kernel Adatron SVNN to the one of SV machines, both with LSE error function. For this purpose four benchmarks have been chosen, that is a two-dimensional set of patterns drawn from gaussian generators, the sonar benchmark [19], the mirror symmetry data [30], and the Wisconsin breast cancer database [43]. To avoid undesired side effects, as observed for polynomial kernels ([5], [13]), in all experiments a radial basis function (RBF) kernel (8) has been used.

In all following experiments the origin has been chosen as the starting point ($\alpha_i = 0 \; \forall i \in \{1..l\}$, $b = 0$). As a stopping criterion for the SVNN the following scheme has been applied: After every cycle the current value of function M_{Ad} has been measured. It is assumed that as soon as the sum of the latest m (e.g. $m = 10$) deviations from the mean is less than a very small threshold, r_r, the algorithm has approximated the optimal solution with high accuracy. This stopping criterion can be expressed as:

$$r_s = \sum_{s=0}^{m-1} | \, |v| - |M_{Ab}^{t-s}| \, | \tag{43}$$

where the mean of the last m values of M_{Ab} has been denoted as v. If $r_s < r_r$ the stopping criterion is satisfied. In all experiments below the stopping criterion has been defined by: $r_s = 0.01$, $m = 10$. Other choices of the two parameters are possible, this allows the user to specify the accuracy of the algorithm. If even more accuracy would be required, other stopping criteria would be possible, e.g. one which is a function of M_{Ab} as well as of the values of b.

5.1 Gaussian Experiment

Task of this experiment is to classify 150 training patterns drawn equally from two gaussian generators. The two classes are overlapping, for a bayesian optimal [17] decision surface (which is quadratic) some misclassifed patterns must be accepted.
To estimate the true risk the empirical error on 500 patterns from the same generator has been measured. The number of errors on the training set and test set is given in table 1 for three choices of C.

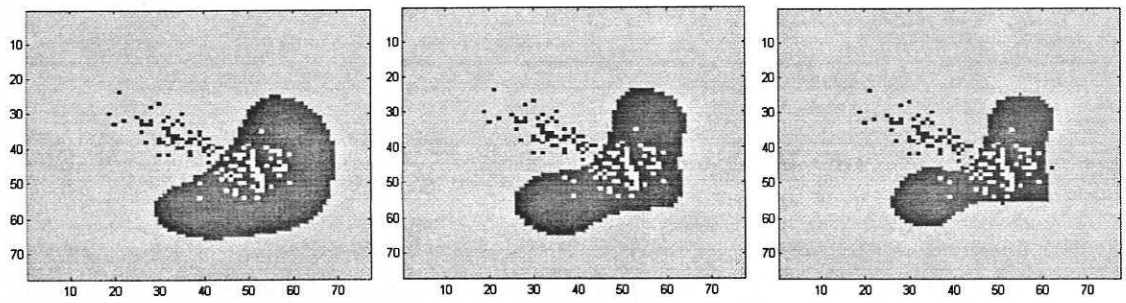

Figure 2: Decision space diagrams computed by the SVNN for three values of C (left: $C = 0.1$, middle: $C = 1$, right: $C = 10$), $\sigma = 0.8$ in all three cases.

The three diagrams in figure 2 illustrate the effect of choosing the C parameter. By choosing C the complexity of the decision function which will be learned is influenced.
The solid line in figure 3 (left) shows the process of optimising the regularized "margin-cost" functional[3] over the number of cycles.
Note that at each point constraint $\sum_{i=1}^{l} y_i \alpha_i = b$ is satisfied. Diagram 3 (left) also illustrates how the bias parameter b gradually converges to an optimal value where the overall cost is minimised.

[3]The "margin-cost" function M_{Ab} is given by the $l+1$ dimensional quadratic form which is minimized. Function M_{Ad} can be computed as indicated in the algorithm given in section 4.4.

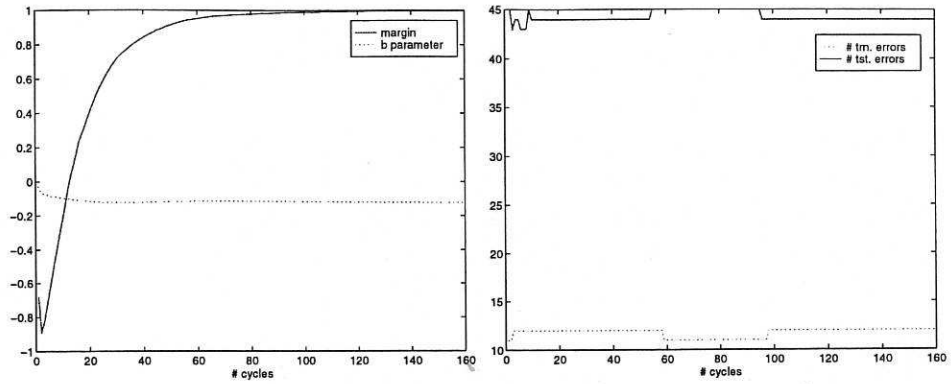

Figure 3: Convergence of the SVNN's regularized margin M_{Ab} and b parameter against the number of cycles (left), and (right) error rate on training set (dotted) and test set (solid) versus the number of cycles ($C = 1, \sigma = 0.8, \eta = 0.0625$).

Alg.	C	$\sigma(RBF)$	η	# Trn.Err	# Tst.Err	FLOPS
SVM	0.1	0.8	-	11	43	25854647
SVNN	0.1	0.8	0.25	12	44	1844307
SVNN	0.1	0.8	0.125	11	43	23777881
SVNN	0.1	0.8	0.0625	11	43	4880081
SVM	1	0.8	-	12	44	690986414
SVNN	1	0.8	0.25	12	44	23239716
SVNN	1	0.8	0.125	12	44	12135615
SVNN	1	0.8	0.0625	12	44	10971373
SVM	10	0.8	-	8	48	867615481
SVNN	10	0.8	0.25	7	48	47235273
SVNN	10	0.8	0.125	8	48	55600319
SVNN	10	0.8	0.0625	8	48	67590981

Table 1: Comparison of SVNN and SVM using the Gaussian patterns.

The values in table 1 show classification results on the gaussian data from the SVNN and SVM for three choices of capacity parameter C. To investingate the effect on choosing a value for the learning rate η the experiment has been repeated for three values of η.
The table shows that the SVNN approximates the SVM's solution quite well. In a few cases the SVNN's solution is slightly different, these small inaccuracies may have been caused by choosing a too large learning rate $\eta = 0.25$. Table 1 shows that the SVNN was able to find its solutions faster than the SVM (see the column which indicates the number of floatpoint operations (FLOPS) which have been required). It is interesting to note that, depending on the choice of C, a different (slightly larger or smaller) learning rate did lead to the fastest convergence. The table also shows that the SVNN is relatively robust to changes in C in the sense that it also converges in a reasonable time for a slightly too small or too large learning rate η.

5.2 Sonar Experiment

In this benchmark it is required to classify the sonar set of 208 patterns representing metal cylinders and rocks, where each pattern has 60 dimensions. As suggested in [19] the angle dependent data has been split into a training set and test set, each of 104 patterns.
In earlier studies ([15], [14]) an optimal value for kernel parameter σ has been determined. Therefore in this and the following two experiments emphasis is put on the comparison between the SVNN and SVM. For three values of kernel parameter σ, and each three choices of η, the SVNN's solution has been calcuated. For three cases diagrams of the convergence of the algorithm and performance on the training set and test set are given.

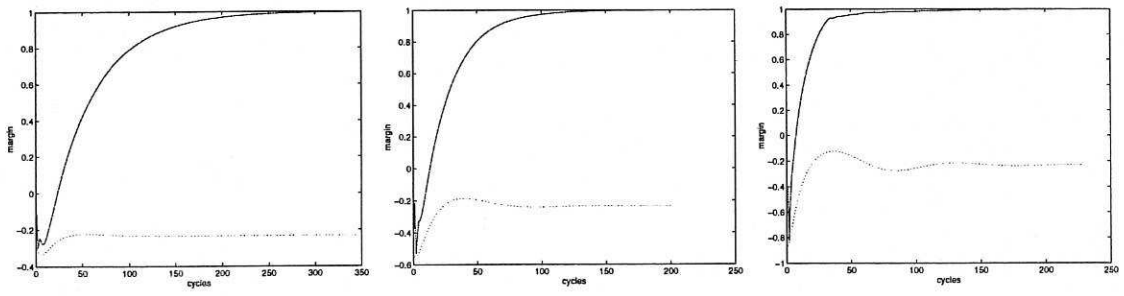

Figure 4: Sonar experiment: Learning curves for function M_{Ad} and bias parameter b against the number of cycles for three choices of the learning rate η: $\eta = 0.05$, $C = 10$ (left), $\eta = 0.10$, $C = 10$ (middle), $\eta = 0.20$, $C = 10$ (right). The plots show that for $\eta = 0.05$ (left) more iterations than for $\eta = 0.1$ (middle) were required. For a large choice of the learning rate$\eta = 0.20$ (right) osciallations in the value of b can be observed which disappear gradually.

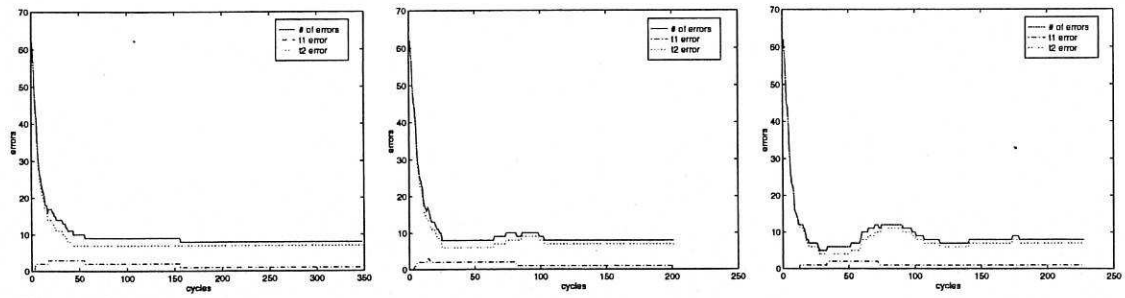

Figure 5: Sonar experiment: Number of classification errors on the test set. Type 1 errors are indicated by the dash-dotted line, type 2 errors by the dotted line, and the overall (type 1 + type 2) error by the solid line. The diagrams show the generalisation performance for three choices of learning rate, η, as given in the description of figure 4. In each case it can be observed that after a number of cycles the kernel Adatron has converged, the error rates are stable and constant. The right plot shows that fluctuations in the value of b (see figure 4) are related to changes in the error rate on the test set.

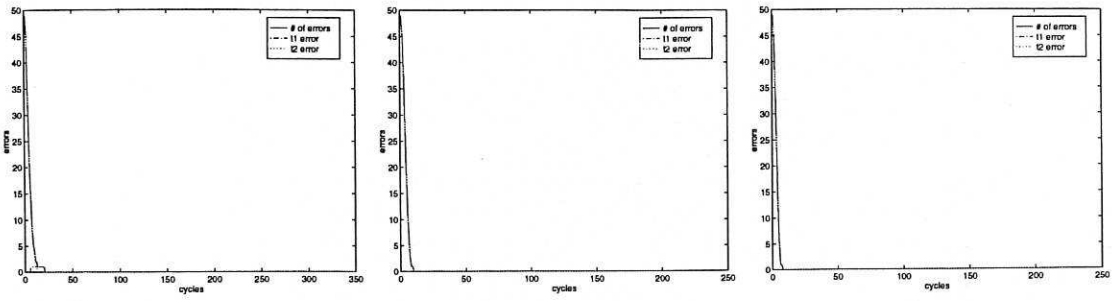

Figure 6: Sonar exeriment: Number of classification errors on the training set. The three diagrams show the empirical error versus the number of cycles for the values of η and C as given in figure 4 (type 1, type 2, and overall error is indicated as in figure 5).

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	0.8	0.1	-	8769622	10	30
SVNN	0.8	0.1	0.01	2731748	5	27
SVNN	0.8	0.1	0.05	1099362	5	27
SVNN	0.8	0.1	0.10	2331980	5	27
SVM	0.8	1.0	-	8769749	1	17
SVNN	0.8	1.0	0.01	16024034	1	17
SVNN	0.8	1.0	0.10	3831110	1	17
SVM	0.8	10	-	19420444	0	8
SVNN	0.8	10	0.01	43769280	0	8
SVNN	0.8	10	0.05	11591706	0	8
SVNN	0.8	10	0.10	6695145	0	8

Table 2: Sonar experiment: Comparision of SVNN and SVM for three values of η, $\sigma = 0.8$.

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	1.0	0.1	-	8769458	13	29
SVNN	1.0	0.1	0.01	2731748	12	26
SVNN	1.0	0.1	0.05	1099362	12	26
SVNN	1.0	0.1	0.10	2431922	12	26
SVM	1.0	1.0	-	8769868	1	15
SVNN	1.0	1.0	0.01	17056768	1	13
SVNN	1.0	1.0	0.05	4264192	1	13
SVNN	1.0	1.0	0.10	4197564	1	14
SVM	1.0	10	-	49335323	0	9
SVNN	1.0	10	0.01	56870687	0	9
SVNN	1.0	10	0.05	15348892	0	9
SVNN	1.0	10	0.10	8323604	0	9

Table 3: Sonar experiment: Comparision of SVNN and SVM for three values of η, $\sigma = 1.0$.

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	1.2	0.1	-	8769582	18	30
SVNN	1.2	0.1	0.01	2731748	14	26
SVNN	1.2	0.1	0.05	1165990	14	26
SVNN	1.2	0.1	0.10	2565178	14	26
SVM	1.2	1.0	-	14137906	2	15
SVNN	1.2	1.0	0.01	17622104	2	14
SVNN	1.2	1.0	0.05	4463764	2	14
SVNN	1.2	1.0	0.10	3431069	2	14
SVM	1.2	10	-	63186138	0	8
SVNN	1.2	10	0.01	68241212	0	9
SVNN	1.2	10	0.05	18806642	0	9
SVNN	1.2	10	0.10	10285299	0	9

Table 4: Sonar experiment: Comparision of SVNN and SVM for three values of η, $\sigma = 1.2$.

Table 2-4 shows that in most cases the SVNN's solutions are identical to the ones found by the SVM based on MATLAB's routine for quadratic programming. In all three cases where $C = 0.1$ the SVNN has returned a different solution for three different values for η. This solution has a lower error rate on the test set. The effect is explained by the sensitivity of the quadratic programming routine which has been used. Depending on the conditioning of the quadratic form the accuracy seems to vary. It is beliefed that in these cases the soluions found by the SVNN are more accurate than those determined by the SVM used.

5.3 Mirror Symmetry Experiment

All patterns from the mirror symmetry benchmark are synthetic patterns (200 patterns for the training set, 100 patterns for the test set). The aim of the benchmark [13] is to distinguish patterns which are symmetric about their centre from patterns which are not symmetric. The three tables in this subsection compare the performance of the SVNN and SVM for each three choices of σ, in each table three values of C and η have been applied.

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	5	0.1	-	60592406	18	35
SVNN	5	0.1	0.0078	12163400	15	34
SVNN	5	0.1	0.0156	7176406	15	34
SVNN	5	0.1	0.125	48653597	15	34
SVM	5	1.0	-	60590512	0	7
SVNN	5	1.0	0.0078	62398242	0	7
SVNN	5	1.0	0.0156	35273860	0	7
SVNN	5	1.0	0.125	39287782	0	8
SVM	5	10	-	60592277	0	5
SVNN	5	10	0.0078	133189230	0	5
SVNN	5	10	0.0156	73831838	0	5
SVNN	5	10	0.125	19217428	0	6

Table 5: Mirror symmetry: Comparison of SVNN and SVM for $\sigma = 5$.

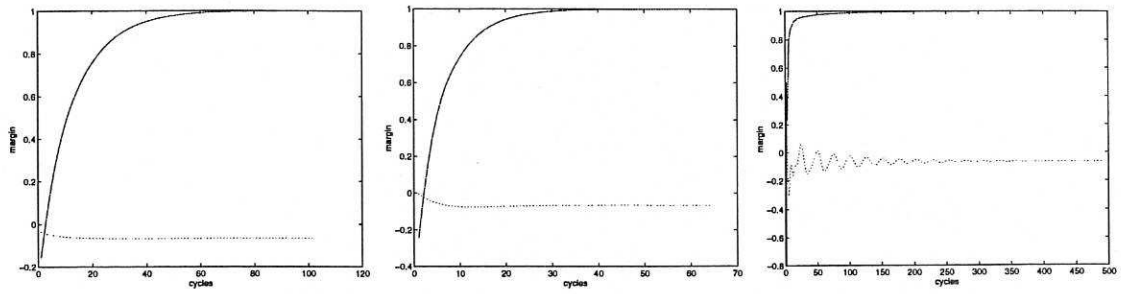

Figure 7: Mirror symmetry experiment: Learning curves for functional M_{Ad}, and bias parameter b, are plotted against the number of cycles for three choices of the learning rate η: $\eta = 0.0078$ (left), $\eta = 0.0156$ (middle), $\eta = 0.125$ (right). In all three cases $C = 0.1$, and $\sigma = 7$. The diagrams show that either a very small or a too large learning rate can slow down the learning process. For a very small value of η many updates are required, for a too large η osciallations in the value of M_{Ad} and b occur. After the learning process has converged in all three cases to the same solution has been found by the algorithm (see table 5).

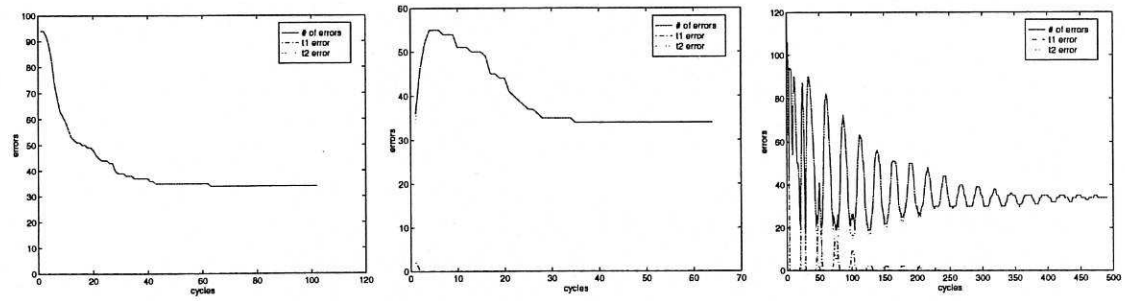

Figure 8: Mirror symmetry experiment: Number of classification errors on the testset. Type 1 errors are indicated by the dash-dotted line, type 2 errors by the dotted line and the overall error by the solid line. Parameters were chosen as indicated in figure 7. The right plot shows that osciallations on the value of b (see figure 7) lead to oscillations in the performance on the testset.

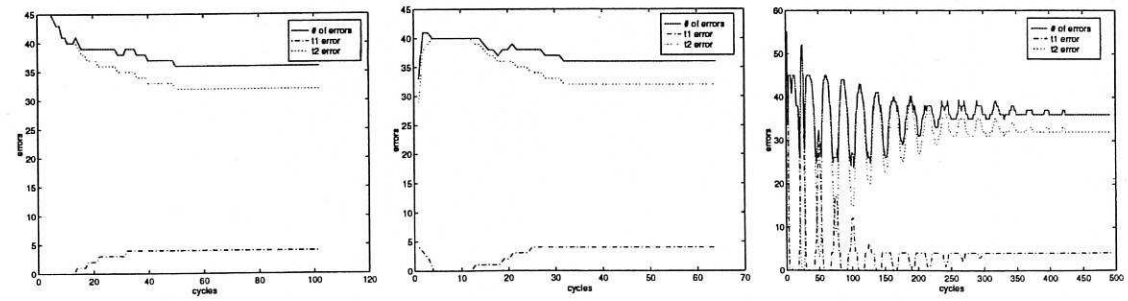

Figure 9: Mirror symmetry exeriment: Number of classification errors on the trainset. The three pictures show the empirical error versus the number of cycles. The values for η and C are given above in figure 7.

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	6	0.1	-	60591736	28	32
SVNN	6	0.1	0.0078	12285034	26	33
SVNN	6	0.1	0.0156	7419674	26	33
SVNN	6	0.1	0.125	54613663	26	33
SVM	6	1.0	-	60590825	0	10
SVNN	6	1.0	0.0078	68844844	0	10
SVNN	6	1.0	0.0156	39166148	0	10
SVNN	6	1.0	0.125	37098370	0	10
SVM	6	10	-	209722180	0	4
SVNN	6	10	0.0078	176604624	0	4
SVNN	6	10	0.0156	104113040	0	4
SVNN	6	10	0.125	19338090	0	5

Table 6: Mirror symmetry: Comparison of SVNN and SVM for $\sigma = 6$.

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	7	0.1	-	60592032	35	37
SVNN	7	0.1	0.0078	12406668	34	36
SVNN	7	0.1	0.0156	7784576	34	36
SVNN	7	0.1	0.125	61181899	34	36
SVM	7	1.0	-	60591130	0	14
SVNN	7	1.0	0.0078	73953472	0	14
SVNN	7	1.0	0.0156	42328632	0	13
SVNN	7	1.0	0.125	21164316	0	11
SVM	7	10	-	246209555	0	4
SVNN	7	10	0.0078	224888888	0	4
SVNN	7	10	0.0156	134639844	0	4
SVNN	7	10	0.125	22256325	0	4

Table 7: Mirror symmetry: Comparison of SVNN and SVM for $\sigma = 7$.

Tables 5-7 show that the SVNN has achieved a fine performance which is equivalent, or at least highly similar, to the one of SVM's. In the cases for $C = 0.1$ the SVNN's solutions are slightly different, a similar effect has been observed in the gaussian experiment. Since in these cases the performance of the testset is quite good it is assumed again that the SVM's solution is less accurate than the one found by the SVNN.

5.4 Wisconsin Breast Cancer Diagnosis

The original Wisconsin breast cancer database consists of 699 instances. For this experiment a subset of 200 patterns was randomly chosen for the training set and 200 patterns were chosen for the test set.

The plots in figure 10 show the effect on choosing a small, moderate, and far too large learning rate. While in the first two cases the algorithm converges nicely, the curve in the latter case shows that for an extreamely large choice of the learning rate, $\eta = 1$, the algorithm cannot converge any more. This is not surprising since the theory of the Adatron, and more generally of neural networks, provides usually an interval for apropriate values of the learning rate η.

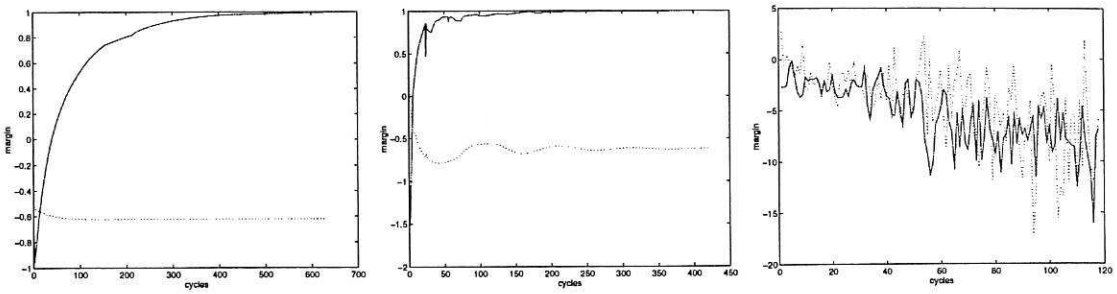

Figure 10: Wisconsin breast cancer diagnosis experiment: The values of functional M_{Ad}, and bias parameter b is plotted against the number of cycles for three choices of the learning rate η: $\eta = 0.04$ (left), $\eta = 0.4$ (middle), $\eta = 1$ (right). In all three cases $C = 10$ and $\sigma = 0.6$.

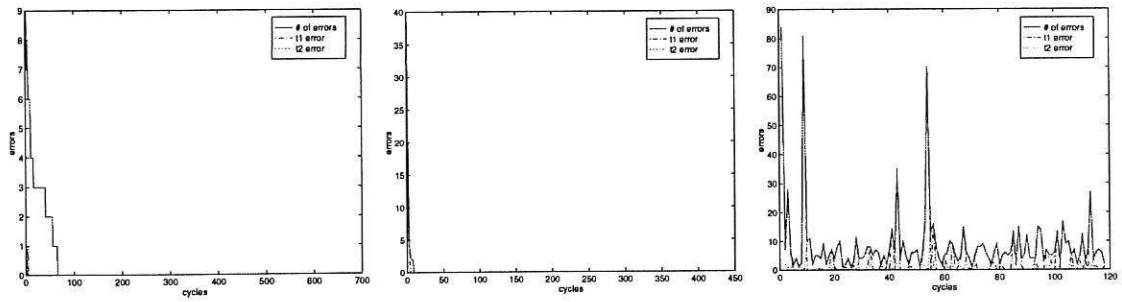

Figure 11: Wisconsin breast cancer diagnosis experiment: Number of classification errors on the testset (type 1, type 2, and overall errors are indicated as before). All other parameters were chosen as as denoted in figure 10.

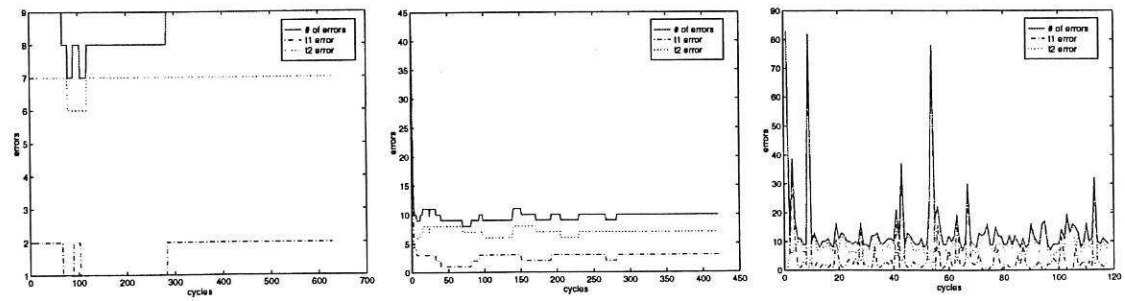

Figure 12: Wisconsin breast cancer diagnosis experiment: Number of classification errors on the training set for three values of η (all other parameters were chosen as specified in figure 10).

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	0.4	0.1	-	59726326	11	13
SVNN	0.4	0.1	0.004	21797649	10	12
SVNN	0.4	0.1	0.04	6021219	10	12
SVNN	0.4	0.1	0.4	1797987	8	13
SVM	0.4	1.0	-	2158131600	2	11
SVNN	0.4	1.0	0.004	110719152	2	11
SVNN	0.4	1.0	0.04	17188854	2	11
SVNN	0.4	1.0	0.4	101197821	2	11
SVM	0.4	10	-	2406014900	0	11
SVNN	0.4	10	0.004	253450767	0	11
SVNN	0.4	10	0.04	40193268	0	11
SVNN	0.4	10	0.4	60066207	0	10

Table 8: Wisconsin breast cancer diagnosis experiment: Comparision of SVNN and SVM for three values of η, $\sigma = 0.4$.

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	0.5	0.1	-	59922228	9	11
SVNN	0.5	0.1	0.004	22158936	9	11
SVNN	0.5	0.1	0.04	6262179	9	11
SVNN	0.5	0.1	0.4	1797933	8	11
SVM	0.5	1.0	-	2207641700	3	10
SVNN	0.5	1.0	0.004	121526988	3	10
SVNN	0.5	1.0	0.04	18509577	3	10
SVNN	0.5	1.0	0.4	95416857	3	10
SVM	0.5	10	-	2513699700	0	10
SVNN	0.5	10	0.004	291473100	0	10
SVNN	0.5	10	0.04	55497294	0	10
SVNN	0.5	10	0.4	55483005	0	10

Table 9: Wisconsin breast cancer diagnosis experiment: Comparision of SVNN and SVM for three values of η, $\sigma = 0.5$.

Alg.	$\sigma(RBF)$	C	η	FLOPS	# Trn.Err	# Tst.Err
SVM	0.6	0.1	-	59922468	8	10
SVNN	0.6	0.1	0.004	22399794	8	9
SVNN	0.6	0.1	0.04	6503100	8	9
SVNN	0.6	0.1	0.4	1917693	7	9
SVM	0.6	1.0	-	2239325100	3	9
SVNN	0.6	1.0	0.004	130774671	3	9
SVNN	0.6	1.0	0.04	19349376	3	9
SVNN	0.6	1.0	0.4	93243903	3	9
SVM	0.6	10	-	2719158000	0	9
SVNN	0.6	10	0.004	42473087	0	9
SVNN	0.6	10	0.04	75406500	0	9
SVNN	0.6	10	0.4	50538633	0	10

Table 10: Wisconsin breast cancer diagnosis experiment: Comparision of SVNN and SVM for three values of η, $\sigma = 0.6$.

The results in tables 8-10 indicate that the kernel Adatron SVNN has shown a fine performance, usually at least as good as SV machines, and usually substantially faster.
It is observed again that in cases where a small value for parameter C has been chosen the SVNN's solution seems to be more accurate than the one found by the SVM's quadratic optimizer. In cases where the quadratic form is very steep, which is the case for large choices of C, the optimizer's solutions seem to be accurate. However in cases where the quadratic form is very flat, as for very small choices of C, the optimizer seems to be slightly inaccurate.

6 Conclusion

The support vector neural network with bias and soft margin is an interesting neural network approach to support vector learning. It establishes a link between work done in a more statistical context (PAC and VC theory, SVM), statisitcal mechanics, and neural network research, which has traditionally ([38], [37]) been motivated by more cognitive aspects of neural information processing.

The SVNN is a powerful algorithm because it allows to compute *exact* solutions without numerical problems which can occur in conventional routines for *constrained* quadratic programming [35]. Furthermore the algorithm is very short, numerically stable, and conceptually simple because it is based on *unconstrained* quadratic optimization in the positive quadrant. As usual in neural networks the learning speed is controlled by the choice of a learning rate. It is known that as long as this parameter is chosen small enough the algorithm will always converge [24]. Further work will address the question how to choose a good value for the learning parameter.

Statistical mechanics provides a framework which proves that the Adatron's solution is identical to the one found by SV machines. Extensive experimental studies have confirmed this result for the SVNN with bias unit and LSE error function.

Acknowledgements Thanks to Thorsten Joachims for his useful comment at ICML98. Thanks also to Klaus-Robert Müller for his support which is gratefully acknowledged. This work has been funded by the STORM ESPRIT project, the Engineering and Physical Science Research Council (EPSRC), and Univ. of Sheffield, Dept. AC&SE.

References

[1] Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning, Automations and Remote Control, 25:821-837.

[2] Anlauf, J.K., and Biehl, M. (1989). Europhysics Letters, 10:687.

[3] Bartlett P., Shawe-Taylor J., (1998). Generalization Performance of Support Vector Machines and Other Pattern Classifiers. 'Advances in Kernel Methods - Support Vector Learning', Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.), MIT Press, Cambridge, USA.

[4] Beliezynski, B., Lukianiuk, A., (1998) Fast Regularized Network for Dynamic System Approximation, Eng. and Int. Sys. (EIS98 Conf. Proc.)

[5] Boser, B., Guyon, I., Vapnik, V. (1992). A training algorithm for optimal margin classifiers. Fifth Annual Workshop on Computational Learning Theory. 5, Pittsburgh, ACM Press.

[6] Courant A., Hilbert D. (1951). Methods of Mathematical Physics, Wiley Interscience.

[7] Cortes, C., and Vapnik, V. (1995). Support Vector networks, Machine Learning 20:273-297.

[8] Cortes, C. (1995). Prediction of Generalization Ability in Learning Machines. PhD Thesis, Department of Computer Science, University of Rochester.

[9] Cristianini, N., Frieß , T-T., Campbell, C., Shawe-Taylor, J., (1998). Simple Learning Algorithms for Support Vector Machines. to appear.

[10] Cristianini, N., Shawe-Taylor, J., Sykacek, P., (1998). Bayesian Classifiers are Large Margin Hyperplanes in a Hilbert Space, in Shavlik, J., ed., Machine Learning: Proceedings of the Fifteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA.

[11] Duda R. O., Hart P.E. (1973), Pattern Classification and Scene Analysis, Wiley Interscience, New York

[12] Fahlman, S.E. & Lebiere, C., (1990). The Cascade-Correlation Learning Architecture. In Touretzky (ed.), Advances in Neural Information Processing Systems 2, Morgan-Kaufman, 1990.

[13] Frieß, T-T., Harrison, R.F., (1998), Pattern Classification using Support Vector Machines, Eng. and Int. Sys. (EIS98 Conf. Proc.)

[14] Frieß, T-T., Harrison, R.F., (1998), Perceptrons in Kernel Feature Space, Research Report 720, The University of Sheffield, Department of Automatic Control and Systems Engineering.

[15] Frieß, T-T., Cristianini, N., Campbell, C., (1998). The Kernel Adatron: A Fast and Simple Learning Procedure for Support Vector Machines. in Shavlik, J., ed., Machine Learning: Proceedings of the Fifteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA.

[16] Freund Y., Schapire R. E., (1998) Large Margin Classification Using the Perceptron Algorithm, Proceedings of the Eleventh Annual Conference on Computational Learning Theory (COLT 98)

[17] Fukunaga K., (1990), Introduction to Statistical Pattern Recognition, Academic Press, San Diego

[18] Gardner E., Derrida, B. (1988), Optimal storage properties of neural network models. J. Phys. A:Math. Gen. 21, 271

[19] Gorman R. P. & Sejnowski, T. J. (1988) Neural Networks 1:75-89.

[20] Guyon, I., Matic, N., & Vapnik, V. (1996). Discovering Informative Patterns and Data Cleaning, Advances in Knowledge Discovery and Data Mining ed by U.M.Fayyad, G. Piatelsky-Shapiro, P. Smyth and R. Uthurusamy AAAI Press/ MIT Press.

[21] Grappel T., Obermayer K. (1988). Fuzzy Topographic Kernel Clustering, in Brauer W., ed,. Proceedings of the 5th GI Workshop Fuzzy Neuro Systems '98

[22] Horn A. R., Johnson H. R. (1985). Matrix Analysis, Cambridge University Press, New York

[23] Joachims T. (1988) Making large-scale SVM learning practical, 'Advances in Kernel Methods - Support Vector Learning', Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.), MIT Press, Cambridge, USA.

[24] Kinzel, W.,(1990) Statistical Mechanics of the Perceptron with Maximal Stability. Lecture Notes in Physics, Springer-Verlag, 368:175-188.

[25] Krauth W. and Mezard. M. (1987) J. Phys. A20:L745

[26] Kreyszig E., (1993) Advanced Engineering Mathematics, Wiley, New York

[27] Lang, K.L. & Witbrook, M.J. (1988). Learning to tell two spirals apart. In Proceedings of the 1988 Connectionists Models Summer School, Morgan Kaufmann (Palo Alto, CA).

[28] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, R., & Jackel, L.D. (1990). Handwritten digit recognition with a back-propagation network. In D. S. Touretzky (ed.), Advances in Neural Information Processing Systems 1, 396-404.

[29] Minnick, (1961). Linear Input Logic, IRE Transactions on Electronic Computers, 10, 6-16

[30] Minsky M.L. & Papert, S.A. (1969) Perceptrons, MIT Press: Cambridge.

[31] Muroga et al., (1961). Theory of Majority Decision Elements, Journal of Franklin Institute, 271, 376-419

[32] Opper, M. (1988). Learning Times of Neural Networks: Exact Solution for a Perceptron Algorithm. Physical Review A38:3824

[33] Opper, M. (1989). Learning in Neural Networks: Solvable Dynamics. Europhysics Letters, 8:389

[34] Osuna E., Freund R., Girosi F., (1997) "Training Support Vector Machines: An Application to Face Detection", Proc. Computer Vision and Pattern Recognition '97, 130-136

[35] Platt J. (1998). Fast Training of Support Vector Machines using Sequential Minimal Optimisation, 'Advances in Kernel Methods - Support Vector Learning', Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.), MIT Press, Cambridge, USA.

[36] Pontil M., Verri A. (1998) Properites of Support Vector Machines, Neural Computation, MIT AI Memo 1612.

[37] Rummelhart, D. E., Mc Clelland, J. L. (1984) Parallel Distributed Processing, Cambridge Massachusets

[38] Rosenblatt F., (1961). Principles of Neurodynamics, Spartan Press, New York

[39] Ripley B.D., (1996). Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge

[40] Saunders C., Gammermann A., Vork V. (1998). Ridge Regression Learning Algorithm in Dual Variables, in Shavlik, J., ed., Machine Learning: Proceedings of the Fifteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA.

[41] Schölkopf, B., (1997). Support Vector Learning. PhD Thesis. R. Oldenbourg Verlag, Munich.

[42] Smith, (1968). Pattern Classifier Design by Linear Programming, IEEE Transactions on Computers, 17, 367-372

[43] Ster, B., & Dobnikar, A. (1996) Neural networks in medical diagnosis: comparison with other methods. In A. Bulsari et al. (ed.) Proceedings of the International Conference EANN'96, p. 427-430.

[44] Vanderbei R. J., (1994) LOQO: An interior point code for quadratic programming, TR SOR-94-15, Statistics and Operations Research, Priceton University, NJ

[45] Vapnik, V., Chervonenkis A. (1979) Theory of Pattern Recognition, German Translation, Akademie Verlag, Berlin

[46] Vapnik, V. (1995) The Nature of Statistical Learning Theory, Springer Verlag.

[47] Watkin, T., Ran, A. & Biehl, M. (1993). The Statistical Mechanics of Learning a Rule, Rev. Mod. Phys. 65(2).

7 Appendix

7.1 MATLAB-Pseudocode of the Algorithm

```
% Support Vector Neural Network (SVNN)
%
% load training patterns and calculate matrix dvals = E^{LSE},
%   as defined above
% initialize vector y (the vector of labels)
% initialize all compoments of vector alpha to zero
% initialize bias parameter, b = 0

% define %  TRUE  = 1; FALSE = 0; STOP = FALSE;

while STOP == FALSE
 min_Margin = 100e250;
 for indexr=1:l
  alphay = alpha .* y;

   decval = y(indexr) * (dvals(indexr,:) * alphay + b);
   if decval < min_Margin min_Margin = decval; end;
   update = eta * ( 1 - decval);

   if (alpha(indexr) + update) > 0
        alpha(indexr) = alpha(indexr) + update ;
        b = b + update * y(indexr);
   end;
 end; % end for

 % if you want, check that b is feasible  (b == sumalphay)
 %   sumalphay = sum(alpha .* y)

 if Stopping_Criterion_is_satisfied STOP = TRUE; end;

end; % end while
```