X

# The Kernel Adatron with Bias Unit: Analysis of the Algorithm. Part 2

Thilo-Thomas Frieß and Rob Harrison

Research Report Number 728

Date: 21. October 1998

The University of Sheffield

Department of Automatic Control and Systems Engineering

Mappin Street, Sheffield, S1 3JD, England

email: friess@acse.shef.ac.uk

Acknowledgements: This work has been funded by the STORM ESPRIT project, the Engineering and Physical Science Research Council (EPSRC), and University of Sheffield, Department of Automatic Control and Systems Engineering. Thanks to all my colleagues at GMD First, to Alex Smola, Bernhard Schölkopf, Nello Cristianini, and Kathrin Klebl.

Abstract: The kernel Adatron with bias allows to find a large margin classifier in the kernel feature space. It is an adaptive perceptron (Adatron) which uses augmented patterns in the kernel feature space. The algorithm has been proposed firstly in [9]. In this document the convergence properties of the algorithm are investigated. As a by-product of the current analysis a simpler formulation of the algorithm is derived.

## 1 Introduction

The kernel Adatron, a non-linear version of the Adatron algorithm, allows to compute large-margin perceptrons in non-linear kernel feature spaces, assuming that the perceptron has not bias parameter. Recently the kernel Adatron with bias has been introduced [9], there it is shown how patterns can be augmented in the feature space in order to learn the bias parameter.

In this document the proof of the algorithm is given. As a by-product a new formulation of the optimisation problem arises which is even more simpler than the one which has been proposed in [9].

The document is structured as follows: In the first section perceptrons are briefly discussed, then the algorithms for the kernel Adatron and kernel Adatron with bias are considered. Afterwards the proof of the kernel Adatron with bias is presented. Finally another algorithm, which is an adaption of the kernel Adatron using "augmented" kernels is studied. It is shown that this algorithm cannot augment the training patterns in the feature space.

# 2 Perceptron Learning

## 2.1 Rosenblatt's Perceptron and Data Dependent Perceptrons

Rosenblatt's learning algorithm for perceptrons [22] 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 $y_i \in \{+1, -1\}$ $\forall i \in \{1..l\}$. If it is possible to classify all training patterns correctly, the algorithm terminates in a finite number of cycles [16]. 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 $\beta_i$ 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 $\alpha_i$ are non-negative numbers.

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.

The perceptron's decision function can be rewritten in expanded form:

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

and the perceptron's update rule $w \leftarrow w + y_i x_i$ can be rewritten as:

$$\alpha_i \leftarrow \alpha_i + \eta \tag{2}$$

Note that in (2) the elements of the vector of the multipliers, $\alpha$, are updated instead of updating the weight vector, $w$, directly. Decision functions of the form (1) will be called data dependent representation (because the weight vector has been expanded on some data points), while decision functions of the form $f(x) = \langle w, x \rangle + b$ will be called explicit representation ($w$ is directly available).

To avoid the perceptron's decision function 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 after learning which has one more degree of freedom:

$$\forall i \in \{1..d\}\ w_i^a = w_i,\ w_{d+1}^a = b \tag{3}$$

A pattern can be augmented simply by 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 (2) by the following update rule:

$$\alpha_i \leftarrow \alpha_i + \eta \ \ ; \ \ b \leftarrow b + \eta y_i \tag{4}$$

The first part of this rule ($\alpha_i \leftarrow \alpha_i + \eta$) ensures that the first $d$ components of the 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 some training vectors.

If augmented patters are used the relationship $b = \sum_{i=1}^{l} \alpha_i y_i$ is always satisfied. Imagine a perceptron which has just been initialised such that $b = 0$ and $\alpha_i = 0 \forall i$. After one update of multiplier $\alpha_i$ the value of $b$ is given by: $b = \eta y_i$. After the next update of another multiplier $\alpha_j$ it follows that $b = \eta y_i + \eta y_j$. Therefore, after a finite number of updates, $b = \sum_{i=1}^{l} \alpha_i y_i$ must be satisfied[1].

## 2.2 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:

$$k(x_u, x_v) = \langle \phi(x_u), \phi(x_v) \rangle = \langle z_u, z_v \rangle \tag{5}$$

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 [4].

Examples for kernel functions are the scalar product, the radial basis function (RBF) kernel, the polynomial kernel, and the $+\zeta$ kernel (a meta kernel):

$$k_{sp}(x_u, x_v) = \langle x_u, x_v \rangle \tag{6}$$

$$k_{RBF}(x_u, x_v) = exp(-||x_u - x_v||^2 / \sigma^2) \tag{7}$$

$$k_{pol}(x_u, x_v) = (\langle x_u, x_v \rangle + 1)^d, \ d = 1, 2, .. \tag{8}$$

$$k_{+\zeta}(x_u, x_v) = k(x_u, x_v) + \zeta, \ \zeta \geq 0 \tag{9}$$

Kernel-based algorithms are elegant in the sense that the "kernel trick" allows algorithms to operate implicitly in very high dimensional feature spaces without explicitly expand-

[1]This can also be seen by considering the decision function (see (22) below).

ing patterns into their feature space representation. Further information about kernel functions can be found in [24].

## 2.3 Adatron

The Adatron is a neural network algorithm [14] 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 optimised ([2], [10]):

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{10}$$

subject to:

$$\alpha_i \geq 0 \; \forall i \in \{1..l\} \tag{11}$$

**Adatron (rewritten in the data dependent representation)**

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

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

```
1. Choose a starting point (e.g. $\alpha_i = 0.1 \; \forall i \in \{1..l\}$), choose a learning rate $\eta$,
   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
```

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].

In the seventies a highly similar algorithm has been studied in a more statistical context [27]. There it is pointed out that this perceptron 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 (in step 4 of the algorithm), 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 (10).

## 2.4 Kernel Adatron (KA)

The kernel Adatron is a nonlinear version of the Adatron algorithm. 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 that's required is to replace the linear function $f_{Ad}(x)$ from the algorithm above by the following nonlinear function:

$$f_{Ad}(x) = y_i \sum_{j=1}^{l} y_j \alpha_j k(x_i, x_j) \tag{12}$$

For the training vectors the computation of function $f_{Ad}(x)$ can be performed efficiently by calculating a kernel correlation matrix $M_{i.j} = k(x_i, x_j)$ (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:

$$f_{Ad}(x_i) = y_i \langle (\alpha \circ y), M^i \rangle \quad i \in \{1..l\} \tag{13}$$

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

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.

## 2.5 Kernel Adatron with Bias Unit

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.

Kernel Adatron with Bias Unit

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

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

```
1. Choose a starting point[2] (e.g. $\alpha_i = 0 \; \forall i \in \{1..l\}$), choose a learning rate $\eta$,
   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
```

[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.

In this kernel Adatron the weight vector, $w$, is not accessible, it resides in feature space. Recall that it is possible to access and update the $b$-parameter explicitly by augmenting the weight vector $w$ in the feature space [9].

When support vector machines have found a solution constraint:

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{14}$$

is always satisfied. Since the kernel Adatron with bias uses augmented patterns constraint:

$$\sum_{i=1}^{l} \alpha_i y_i = b \tag{15}$$

is satisfied at any feasible point.

The gradient evaluated in step 4 of the algorithm minimises the following cost function:

$$L(w, b, \alpha) = \sum_{i=1}^{l} \alpha_i (1 - b y_i) - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{16}$$

which is also optimised by support vector machines [3]. The difference is that in SV machines the function is optimised with respect to (14), while in the kernel Adatron with bias term constraint (15) will always be satisfied.

## 3 Proof of the Algorithms

### 3.1 Proof of the Kernel Adatron with Bias

The kernel Adatron with bias performs a gradient descent in the quadratic form (16). Substituting (15) into this quadratic form leads to:

$$\begin{aligned}L(w,b,\alpha) &= \sum_{i=1}^{l}\alpha_i(1-by_i) - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i,x_j)\\ &= \sum_{i=1}^{l}\alpha_i - \sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i,x_j)\\ &= \sum_{i=1}^{l}\alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j[k(x_i,x_j)+2] \qquad (17)\end{aligned}$$

The latter formulation shows an efficient alternative to compute the solution; expression (17) can be optimised by the kernel Adatron algorithm without bias, then afterwards the value of $b$ is computed using relationship (15).

Clearly there are two ways to calculate the kernel Adatron with bias, either by minimising (17) with a KA in conjunction with calculating the value of $b$ using (15), or by the initial algorithm stated above. Both solutions must be identical because they optimise the same cost function.

During its learning process the kernel Adatron with bias performs a gradient descent in cost function (17). This gradient is given by:

$$\frac{\delta L(w,b,\alpha)}{\delta\alpha_i} = 1 - \sum_{j=1}^{l} y_i y_j \alpha_j k(x_i,x_j) + by_i = 1 - f_{Ad}(x_i) \qquad (18)$$

where function $f_{Ad} = y_i(\sum_{j=1}^{l} y_j\alpha_j k(x_i,x_j) + b)$, as defined above in section 2.5.

In each step of the algorithm the gradient in direction of one canonical base vector, $\alpha_i$, is computed. If a multipliers value remains nonnegative the weight vector, $w$ is updated. The weight, $w$, is represented by an expansion. Therefore multipliers, $\alpha$, are updated in the algorithm such that the perceptron's margin will be increased, that is the cost measured by the cost function is reduced.

The nested kernel function

$$k_{+2}(x_i, x_v) = k(x_i, x_v) + 2$$

is positive semidefinite, therefore the proof of the kernel Adatron with bias can be reduced to the proof of the kernel Adatron.

## 3.2 Proof of the Kernel Adatron

This proof has been pointed out already in [10], and is a straightforward extension of the original proof of the Adatron algorithm:

**1. Anlauf and Biehl [2]:** *Every stable point for the Adatron algorithm is a maximal margin point and vice versa.* By inserting the Kuhn-Tucker conditions for the maximal margin ($\alpha_i > 0 \Leftrightarrow f_{Ad}(x_i) = 1$, $\alpha_i = 0 \Leftrightarrow f_{Ad}(x_i) > 1$) in the algorithms updating rule it follows that the optimal margin is a fixed point. Vice versa by imposing $p_u \alpha_i = 0 \; \forall i$ the Kuhn-Tucker conditions are obtained.

**2. Anlauf and Biehl [2]:** *The algorithm converges in a finite number of steps to a stable point if a solution exists.* The quadratic functional:

$$L = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_i \langle \phi(x_i), \phi(x_j) \rangle \tag{19}$$

is upper bounded [2] and is increased monotonically at each updating step of the algorithm, so it will find a fixed point in a finite number of steps.

The proof of the kernel Adatron with L1 and L2 error function is straightforward and can be derived easily from the formulations given above.

## 4 A Thought Experiment

Apparently it may seem to be sufficient just to optimise function:

$$L = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j [k(x_i, x_j) + 1] \tag{20}$$

by the KA algorithm (without bias). The "trick" is here to learn the bias parameter, $b$, implicitly by augmenting the training patterns in the feature space. A pattern in feature space, $\phi(x)$, is augmented by introducing a new dimension, the new component of the augmented vector $[\phi(x), 1]$ has the value 1. This naive approach is wrong. The reason lies in the fact that in the derivation of the cost function (20) it has already been assumed that constraint:

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{21}$$

is satisfied. Therefore the decision function learned by this KA (without bias) *passes through the origin* in the feature space implied by kernel function:

$$k_{+1}(x_i, x_j) = k(x_i, x_j) + 1$$

On the other hand the decision function of this thought-experiment algorithm must be:

$$f(x_i) = \sum_{j=1}^{l} \alpha_j y_j k_{+1}(x_i, x_j) = \sum_{j=1}^{l} \alpha_j y_j k(x_i, x_j) + \sum_{j=1}^{l} \alpha_j y_j = \sum_{j=1}^{l} \alpha_j y_j k(x_i, x_j) + b \tag{22}$$

where it has been assumed that the following constraint has been satisfied.

$$\sum_{j=1}^{l} \alpha_j y_j = b \tag{23}$$

**Contradiction:** Constraint (23) and (21) cannot be satsified at the same time !

## 5 Conclusion

The kernel Adatron with bias has been studied. It has been shown that both the KA with bias and SV machines minimise the same quadratic cost functions, but with different constraints.

The optimisation problem for the kernel Adatron with bias has been reformulated, leading to an even simpler process which allows to find the multipliers, $\alpha$, by a KA, and then calculate the bias term of the decision function afterwards.

While in SV machines the bias term is always chosen such that $\sum_{i=1}^{l} \alpha_i y_i = 0$ is satisfied, in the kernel Adatron with bias augmented patterns are used, leading to a constraint of the form: $\sum_{i=1}^{l} \alpha_i y_i = b$.

It has been shown that just by replacing in the KA the kernel function, $k(x_u, x_v)$, by another kernel function, $k_{+1}(x_u, x_v) = k(x_u, x_v) + 1$, the bias parameter, which would stem from augmenting patterns in the feature space, cannot be learned correctly.

## 6 Appendix

Aim of this appendix is to show that SV machines optimise function (16). The formulation is more or less straightforward and in a similar form given in [5]. It is presented again only for the purpose of clarification.

To maximise the margin of a perceptron a Lagrangian is formed:

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

where the $\alpha_i$ are nonnegative Lagrangian multipliers for the constraints.

At the point which minimises $L$ with respect to $w$ the following constraint is satisfied:

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

Expression (25) can be inserted into (24) which leads to:

$$\begin{aligned} L &= \frac{1}{2}\langle w, w\rangle - \sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i (y_i(\langle w, \phi(x_i)\rangle + b) - 1) \\ &= \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j k(x_i, x_j) - \sum_{i=1}^{l} \alpha_i (y_i \langle \sum_{j=1}^{l} \alpha_i y_i \phi(x_j), \phi(x_i)\rangle + b) - 1) \\ &= \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j k(x_i, x_j) - \sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j k(x_i, x_j) - \sum_{i=1}^{l} \alpha_i y_i b + \sum_{i=1}^{l} \alpha_i \\ &= \sum_{i=1}^{l} \alpha_i (1 - b y_i) - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i\alpha_j y_i y_j k(x_i, x_j) \end{aligned} \tag{26}$$

If, as in SV machines, it is additionally assumed that $\sum_{i=1}^{l} \alpha_i y_i = 0$, this cost function can be rewritten as [5]:

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

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

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

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

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

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

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

[9] Frieß, T-T., Harrison, R.F., (1998), Support Vector Neural Networks, Research Report 725, *Department of Automatic Control and Systems Engineering*, The University of Sheffield.

[10] 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.

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

[12] Gardner E., Derrida, B. (1988), Optimal storage properties of neural network models. *Journal of Physics.* **21**, 271

[13] Gorman R. P. & Sejnowski, T. J. (1988). Finding the size of a neural network. *Neural Networks*,1:75-89.

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

[15] Minnick, (1961). Linear Input Logic, *IEE Transactions on Electronic Computers*, **10**, 6-16

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

[17] Muroga E., (1961). Theory of Majority Decision Elements, *Journal of Franklin Institute*, 271, 376-419

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

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

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

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

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

[23] Smola, A., Schölkopf, B., Müller, K-R., (1998). The Connection between Regularization Operators and Support Vector Kernels, to appear in *Neural Networks*

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

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

[26] 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.

[27] Vapnik, V., Chervonenkis A. (1979) Theory of Pattern Recognition (in Russian), German Translation, Wapnik, W., Tschervonenkis, A., Theorie der Zeichenerkennung, Akademie Verlag, Berlin

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

[29] Wahba, G., (1998). Support Vector Machines, Reproducing Kernel Hilbert Spaces and the Randomized GACV, in *Advances in Kernel Methods - Support Vector Learning*, Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.), MIT Press, Cambridge, USA.

[30] Watkin, T., Ran, A. & Biehl, M. (1993). The Statistical Mechanics of Learning a Rule, *Review of Modern Physics* 65(2).