NORDIC MATLAB CONFERENCE '95
PROCEEDINGS STOCKHOLM OKTOBER 31 -
NOVEMBER 1, 1995
NORRA LATIN II-49-II-54

Associative Memories for Clusters of Binary Vectors Using MATLAB^{TM} Neural Network Toolbox

Mats Gyllenberg
Department of Mathematics
University of Turku
FIN-20 500 Turku
FINLAND

Timo Koski
Department of Mathematics
Royal Institute of Technology
(KTH)
S-100 44 Stockholm
SWEDEN

Tore Lahti
Research and Development
Nokia Mobile Phones
FIN-90 571 Oulu
FINLAND

Abstract

Two-layer feedforward neural networks implementing associative memories for classification of binary feature vectors, trained by supervised learning and backpropagation of error minimizing crossentropy, are analysed by computer experiments. For this the *MATLAB^{TM} Neural Networks Toolbox* has been used and modified.

We apply adaptation of learning rate to increase the speed of convergence in training a network for a clustering example due to Gower. In particular, the expansion and rejection of identification properties of the memory are of interest and suggest a process of decision directed learning.

1 Associative Networks and Clustering

An *associative memory* is a device for storing given n pairs of input and target vectors, denoted by $\left(\mathbf{x}^{(p)}, t^{(p)}\right)_{p=1}^{n}$, respectively, so that when presented with $\mathbf{x}^{(p)}$ as input, the memory recalls $t^{(p)}$ as output,

$$\mathbf{x}^{(p)} \mapsto \textit{Association} \mapsto t^{(p)} \tag{1}$$

for every p, see [19, 20]. The memory is expected to recall or retrieve the appropriate output vector even when presented with a distorted version of a stored input vector. The storage required for the association is desired to have been achieved by some self-organizing process. The memory should also be able to expand its storage. The datahandling and matrix manipulation commands of MATLAB^{TM} are well suited for expanding the memory in the sense to be considered in the sequel.

In this paper we describe an associative memory for *classification* or *clustering* of binary vectors. The input vectors $\mathbf{x}^{(p)}$ are thus binary vectors of a given length d and the output vectors $t^{(p)}$ describe a partition of the set $\left(\mathbf{x}^{(p)}\right)_{p=1}^{n}$ into mutually disjoint sets. More precisely, $t^{(p)}$ is the *centroid* of the class [8] to which $\mathbf{x}^{(p)}$ is assigned. The set $\left(\mathbf{x}^{(p)}, t^{(p)}\right)_{p=1}^{n}$ is called the *training set*. The number of distinct vectors amongst $\left(t^{(p)}\right)_{p=1}^{n}$ is usually much smaller than n. As the computational implementation of the associative memory we use a *two layer artificial neural network*.

For the details and the background (identification of micro-organisms) for the designated way of dealing with clustering we refer to [10, 11, 12, 13]. A recent general survey of clustering (classification) and neural nets is [23]. Clustering and identification of binary vectors occurs in a number of different applications [1, 7, 8].

2 The Computational Architecture for the Associative Network

The association (1) is here implemented by the basic processing/storage units of neural networks, *neurons*, layered to a parallell computational architecture known as the feedforward multilayer network [14, 15]. The weights and biases of the neurons for the layer at depth l are collected in

$$\mathbf{W}^{(l)} = \left\{w_{j,i}^{(l)}\right\}_{i=0,\ldots,N_{l-1};j=1,\ldots,N_l}$$

and the corresponding outputs of the neurons are

$$\mathbf{u}^{(l)}(\mathbf{x}) = \left(u_1^{(l)}(\mathbf{x}), \ldots, u_{N_l}^{(l)}(\mathbf{x})\right),$$

respectively, where N_l is the number of neurons in the layer at depth l. With the *transfer function* F, a sigmoidal memoryless nonlinearity, the neural processing at depth l reads in the matrix notation of the manual *Neural Network Toolbox for Use with MATLAB^{TM}* [5] as

$$\mathbf{u}^{(l)}(\mathbf{x}) = F\left(\mathbf{W}^{(l)}\begin{pmatrix}1\\ \mathbf{u}^{(l-1)}(\mathbf{x})\end{pmatrix}\right), \tag{2}$$

where 1 denotes the constant input that multiplies the biases $w_{i,0}^{(l)}$. A network is in view of (2) a composition of functions, see [2]. The vector $\mathbf{u}^{(L)}(\mathbf{x})$ designates the *output of the network*, the other output vectors are usually called *hidden layer* outputs. The initial value for (2) is by convention $\mathbf{u}^{(0)}(\mathbf{x}) = \mathbf{x}$. In the computer we have $L = 2$ (a two layer network).

In (2) F is for our purposes exclusively the standard *logsigmoid function*

$$F(x) = \frac{1}{1+e^{-x}}, \tag{3}$$

which is applied componentwise to its vector argument to give the vector output of a layer just as the command **logsig(x)** of the Toolbox is described. Since the target vectors $t^{(p)}$ are constrained to be binary, the output of the network, with components between 0 and 1, must be further rounded off to the nearest binary value. The number of neurons at the output layer is here, as explained in the above, $N_L = d(= \mathbf{size}(\mathbf{x}, 1))$. Thus the network implements a mapping from the binary d-hypercube to itself.

We focus on two layer networks, since any map from the binary d-hypercube to the binary d-hypercube can be realized by means of a two layer neural network, as shown in [3, pp.90 - 95] or in [6]. There seems to be, on the other hand, no precise information about the minimum possible complexity to achieve or to approximate this realization. For representation of maps in the binary d-hypercube by three-layer networks there are some recent intriguing results in [18].

The common neural network architecture for a classification task has usually an output layer with as many neurons as there are pre-established classes. The neuron in the output $\mathbf{u}^{(L)}(\mathbf{x})$ with a high level of activity (≈ 1) singles out the class identity [23]. The present application of neural networks to clustering differs clearly from this in that the target vectors used do not in general contain just a single binary 1. Thus the output of the memory will not approximate the Bayesian classifier based on the training set in same fashion as shown in [22, 24], albeit that the analysis in [22, 24] is in certain respects formally applicable.

As pointed out in the above, the memory may possibly retrieve a vector not equaling any of the stored target vectors, a default which may occur in Bayesian classification only if a rejection threshold is imposed [13]. This behaviour of the network may conceivably be caused by too small a training set. However, e.g. in microbial identification items need to be continuously grouped and identified [28] in a time scale, which is much longer than the training times of two-layer neural networks, but which is on the other hand much smaller than the time needed to accrue large data bases.

3 The Crossentropy Criterion and Learning

3.1 Definition and Rationale of Crossentropy

Let $\mathbf{W} = \left\{\mathbf{W}^{(l)}\right\}_{l=1}^{L}$ comprise all the weights and biases in the network. The *crossentropy* or the *Kullback-Leibler distance* is defined ([4]) by

$$C_n(\mathbf{W}) = \sum_{p=1}^{n}\sum_{i=1}^{d} C\left(t_i^{(p)}, u_i^{(L)}\left(\mathbf{x}^{(p)}\right)\right), \tag{4}$$

where, for $0 \le t \le 1$ and $0 < u < 1$,

$$C(t,u) = -\left[(1-t)\log(1-u) + t\log(u)\right].$$

Learning or selforganization means adjusting $\mathbf{W}$ so that the network's output $\mathbf{u}^{(L)}\left(\mathbf{x}^{(p)}\right)$ as closely as possible resembles, in the sense of $C_n(\mathbf{W})$, the corresponding target vector t^p, a binary vector. Since the target vectors are fixed in advance, this is a case of supervised learning. The minimization of crossentropy by gradient descent i.e. *backpropagation* is mathematically known to converge at least for single layer networks [15, 29].

There are several different rationales for minimizing crossentropy as the training criterion. Here we refer to [10, 11] for an account of the possible biostatistical reasons and to [25] for the general information theory. Learning for single layer networks using minimization of crossentropy corresponds to the maximum likelihood estimation of a certain generalized logistic regression, as shown in [10, 11]. In the single layer case a relation of learning by crossentropy minimization to the correlation matrix memories of [20] can also be shown.

The well known pragmatic argument for minimization of crossentropy is that this has been observed to converge faster than backpropagation of mean square network error, [16, 26]. The MATLAB experiments reported in here and in [21] confirm this finding.

3.2 Backpropagation of Crossentropy using MATLAB

The extension of backpropagation to crossentropy as cost function consists of the evaluation of $\left(\partial/\partial w_{j,i}^{(l)}\right) C_n(\mathbf{W})$ from (4) by a repeated use of the chain rule of calculus.

The details of the analytic derivation of the backpropagation rule for the two-layer case using the logsigmoidal transfer function are given in [21]. Just as the analytic formulae are straightforward modifications of the standard expressions of backpropagation, the MATLAB computations for minimizing crossentropy can be organized following the flow in the M-file **trainbp.m** of $MATLAB^{TM}$ Neural Network Toolbox. The training is thus done by a simultaneous or *batch presentation* of the

training set to the two-layer network. The key alteration required consists of changing both the %BACKPROPAGATION PHASE and the % LEARNING PHASE in **tpb2.m** containing the coding for the *delta rule of backpropagation* as well as the weights and biases update, see [5, pp. 5-7 - 5-8], cf. [14, 15]. As we are restricting ourselves here to the logsigmoidal transfer function (3), the **getdelta** commands involved in the backpropagation phase of **tpb2.m** can be dispensed with. The delta rule for crossentropy reads then as follows:

$$\begin{aligned} deltaw2 &= diff * A1'; \\ deltab2 &= (sum(diff'))'; \\ deltaw1 &= ((ones1 * sumdiff). * A1diff) * P'; \\ deltab1 &= A1diff * sumdiff'; \end{aligned}$$

Here *deltaw*2 and *deltab*2 are the matrix and vector, respectively, of partial derivatives $\left(\partial/\partial w_{j,i}^{(2)}\right) C_n(\mathbf{W})$ with respect to the weights and biases,respectively, in the output layer with *deltaw*1 and *deltab*1 having the same interpretations in the first (or the hidden) layer. In addition, $diff$ is the vector of differences between the target vectors and the net's outputs, $A1$ is the output of the first layer, $A1diff = A1. * (1 - A1)$;, $sumdiff$ is the sum of the differences $diff$ over the whole training set and P is the $d \times n$ matrix of input vectors $\left(\mathbf{x}^{(p)}\right)_{p=1}^{n}$. Other obvious alterations in **tpb2.m** are also called for, but we omit them here. Actually it is not too time consuming to work out a complete crossentropy equivalent of the **trainbp.m** file in the Toolbox.

In e.g. [16, 26] it has been observed that when using the logsigmoidal transfer function (3) there is for the crossentropy backpropagation a small reduction in the number of computations per iteration available in in the output layer, if compared to backpropagation of mean square network error. This can also be seen in the code above.

For initialization of the backpropagation algorithm the weights and biases are chosen at random in $[-1, 1]$ with the **rands** command in the Neural Network Toolbox. Certain other methods of initialization have been analysed and used [21], too.

3.3 Adaptation of the Learning Rate

To increase the speed of convergence an adaptation of the learning rate has also been implemented using the Delta-Bar-Delta learning rule proposed by Jacobs [17]. Here the learning rate is changed during the training process. The pertinent learning rule assigns an individual learning rate to each weight in the net and this rate varies with the epoch, in the sense of [5, p. 3-7], during the training process.

The adaptation is based on the following heuristic considerations. First, if the gradient for the error with respect to a specific weight has had the same sign for some time, the learning rate for that weight should be increased. This will lead to a faster search for the minimum of the error surface. Second, if the sign of the gradient alternates, the learning rate should be decreased.

Let $\mu_{ji}(k)$ be the learning rate for that weight in the kth training epoch, here dropping for ease of writing superscripts for the layer depth. The Delta-Bar-Delta learning rule is

$$w_{j,i}(k+1) = w_{j,i}(k) - \mu_{ji}(k+1) \frac{\partial C_n(\mathbf{W})}{\partial w_{j,i}},$$

where $\mu_{ji}(k+1)$ is computed as follows. We set for each of the hidden and output units

$$\Delta_{ji} = \frac{\partial C_n(\mathbf{W})}{\partial w_{j,i}}.$$

Next, compute the delta-bar as the output of the following AR(1)-filter

$$\overline{\Delta}_{ji}(k) = (1-\beta)\Delta_{ji}(k) + \beta\overline{\Delta}_{ji}(k-1).$$

Parameter β can be chosen by the user within the range $0 < \beta < 1$. The learning rate at epoch $(k+1)$ will be:

$$\mu_{ji}(k+1) = \begin{cases} \mu_{ji}(k) + \kappa & \text{if } \overline{\Delta}_{ji}(k-1) * \Delta_{ji}(k) > 0, \\ (1-\gamma)\mu_{ji}(k) & \text{if } \overline{\Delta}_{ji}(k-1) * \Delta_{ji}(k) < 0, \\ \mu_{ji}(k) & \text{otherwise.} \end{cases}$$

The parameters κ, γ and β have the following (optional) values:

$$\kappa = 0.035, \gamma = 0.333, \beta = 0.7.$$

This rule has also been coded in MATLAB by adding to the previous changes in the **tpb2.m** format.

4 An Experiment: An Associative Memory for a Clustering Example

The example to follow is chosen mainly since it is rather concisely overviewed but is still found interesting. The experiment is concerned with training by backpropagation of crossentropy a two layer ($L = 2$, $N_L = 10$) network for the training set $\mathcal{X}^{12}$ from Gower [9] given by

$$\mathcal{X}^{12} = \begin{pmatrix} (0011111011), a_1 \\ (1101111110), a_2 \\ (0000000000), a_3 \\ (0011101011), a_1 \\ (0000001101), a_4 \\ (0011110011), a_1 \\ (1101111100), a_2 \\ (1101101000), a_2 \\ (0000000101), a_4 \\ (1101110000), a_2 \\ (0011111001), a_1 \\ (1101110100), a_2 \end{pmatrix},$$

where each row contains a binary vector $\mathbf{x}^{(p)}$ with the $(d =)10$ bits enclosed in parentheses and separated by a comma from the corresponding target vector with $p = 1,\ldots,12$ running from the top of the array down. In other words we wish to implement e.g.

$$\mathbf{x}^{(4)} = (0011101011) \mapsto \textit{Association} \mapsto t^{(4)} = a_1$$

and for all other p as given above. The target vectors are

$$a_1 = (0011111011)$$
$$a_2 = (1101111100)$$
$$a_3 = (0000000000)$$
$$a_4 = (0000000101).$$

In fact, the targets are the *centroids* obtained in an unsupervised clustering of $\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(12)}$ by computationally minimizing $1/12\sum_{p=1}^{12}\min_{1\le j\le 4} d_H\left(\mathbf{x}^{(p)}, a_j\right)$, where $d_H\left(\mathbf{x}^{(p)}, a_j\right) = \sum_{i=1}^{10}\left|x_i^{(p)} - a_{ij}\right|$ is the *Hamming* metric. This amounts also to training of a vector quantizer [8] with the Hamming metric as the distortion measure. For a further discussion of the techniques for unsupervised clustering of binary vectors in general and as applied to this example we refer to [12, 27] and their references.

An associative memory for $\mathcal{X}^{12}$ is implemented by backpropagation for minimization of crossentropy and adaptation of learning rate using batching with the MATLAB routines outlined above on a *Sun Sparcstation* in the UNIX environment. In the computer runs we also simultaneously calculated the squared network error.

After 1299 training epochs of running this program the two layer net reached the crossentropy 0.2526 and squared network error 0.05. The network thus trained correctly associates $\mathbf{x}^{(p)}$ to the respective centroids in $\mathcal{X}^{12}$. By this training the given pairs of input and target vectors are put in the storage locations of the neurons, i.e. in the weight matrices $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ for the respective layers. More details of the structure obtained are given in [21]. Let us introduce the *crossentropy between* y and $\mathbf{x}$ *given* $\mathbf{W}$ in a two layer net by

$$d_{C_e}(y,\mathbf{x}|\mathbf{W}) = \sum_{i=1}^{d} C(y_i, u_i^{(2)}(\mathbf{x})).$$

When the trained net was exposed to the vector

$$\mathbf{x}^{(13)} = (0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1)$$

the output became $\mathbf{x}^{(13)}$ itself. This can be viewed as a "bad retrieval" or in our terms rather as a rejection of identification. To see why this particular retrieval occurs, we calculate crossentropies and Hamming distances:

y	$d_{C_e}(y,\mathbf{x}^{(13)}\|\mathbf{W})$	$d_H(\mathbf{x}^{(13)},y)$
a_1	58.15	6
a_2	66.29	8
a_3	1.45	1
a_4	1.59	1
$\mathbf{x}^{(13)}$	0.80	0

We wish to emphasize here that the output is computed by matrix operations (2), not by checking of d_{C_e}. Thus, the net has learned to minimize the crossentropy, and in doing so it has given as output a vector not among the centroids stored in the memory.

Next, we augment in two subsequent stages, the training set with $(\mathbf{x}^{(13)}, a_3)$ and $(\mathbf{x}^{(14)}, a_2)$, where

$$\mathbf{x}^{(14)} = (1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1)$$

and denote the augmentation by $\mathcal{X}^{14}$. After subsequent training initialized by the weight matrix $\mathbf{W}$ obtained in the preceding training we obtain a new weight matrix $\mathbf{W}^*$ and the crossentropy assumes the value 0.05. The vector

$$\mathbf{x}^G = (1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0)$$

best predicts $\mathcal{X}^{14}$ as a single cluster. For $\mathbf{x}^G$ the expanded memory recalls a_1 as output, which is the centroid closest to $\mathbf{x}^G$ w. r. t. both the Hamming distance and the crossentropy, as

y	$d_{C_e}(y,\mathbf{x}^G\|\mathbf{W}^*)$	$d_H(\mathbf{x}^G,y)$
a_1	0.06	2
a_2	28.71	4
a_3	57.26	5
a_4	56.31	5
$\mathbf{x}^G$	9.22	0.

For the binary complement $\overline{\mathbf{x}}^G$ of $\mathbf{x}^G$ we get the following:

y	$d_{C_e}(y,\overline{\mathbf{x}}^G\|\mathbf{W}^*)$	$d_H(\overline{\mathbf{x}}^G,y)$
a_1	5.73	8
a_2	32.93	6
a_3	9.58	5
a_4	5.00	5
$\overline{\mathbf{x}}^G$	29.32	0.

The actual output is the binary complement $\overline{a}_2$. The crossentropy $d_{C_e}(\overline{a}_2,\overline{\mathbf{x}}^G|\mathbf{W}^*)$ equals 2.87 i. e. the net decided correctly in the sense of crossentropy minimization, whereas the output is not a centroid. Testing with 1000 binary 10×1 vectors randomly chosen the average distance from the stored centroids in $\mathcal{X}^{14}$ was 4.5 bits.

5 Conclusions

Supervised training of two-layer feedforward neural networks using minimization of the crossentropy as learning criterion has been demonstrated to be a convenient extension to MATLAB^{TM} Neural Network Toolbox. The addition of adaptive learning rates is also readily done. The resulting training process converges faster than the squared network error propagation.

We have experimented with a clustering example by exposing the fully trained associative memory to d - dimensional binary vectors $\mathbf{x}$ not included in the original

data set $\left(\mathbf{x}^{(p)}\right)_{p=1}^{n}$. The expectation is that the associative memory should find, primarily amongst the target vectors, the best resemblance to a given binary vector $\mathbf{x}$. When the memory is realized as a two-layer feedforward neural network, this would seem to suggest a fast and easy way of doing matrix computations to check the nearest neighbor criterion of clustering. It turns out that the associative memory will sometimes retrieve some vector different from the stored targets. By our experiments the bad retrievals are, however, nearest vectors in the sense of crossentropy criterion used in the training of the underlying neural network. The envisaged, but not yet automated, cycle with default of identification leading to a subsequent extension of memory and retraining with badly retrieved items constitutes in an overall view a non-supervised learning process related to decision directed learning [30] which is of interest in classification of micro-organisms.

References

[1] E.B.J. Andersen: *The Statistical Analysis of Categorical Data.* Second, Revised and Enlarged Edition, Springer-Verlag, Berlin, 1991.

[2] A. R. Barron and R.L. Barron: Statistical learning networks: a unifying view. E. Wegman (ed.): *Computing Science and Statistics: Proceedings of the 20th Symposium on the Interface.* American Statistical Association, Washington, 1989, pp. 192 - 203.

[3] A. Benveniste, M. Metivier and P. Priouret: *Adaptive Algorithms and Stochastic Approximation.* Springer-Verlag, Berlin, 1990.

[4] T.M. Cover and J.A. Thomas: *Elements of Information Theory.* John Wiley and Sons Inc., New York, London, 1991.

[5] H. Demuth and M. Beale: *MATLAB™ Neural Network Toolbox.* The MATHWORKS Inc., Natick, Mass., June 1992.

[6] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, and L. Jackel: Large Automatic Learning, Rule Extraction and Generalization. *Complex Systems*, 1, 1987, pp. 877 - 922.

[7] A.El Gamal, L.A. Hemachandra, I. Shperling, & V.K. Wei: Using Simulated Annealing to Design Good Codes. *IEEE Transactions on Information Theory*, Vol IT-33, 1987, pp. 116- 123.

[8] A.Gersho & R.M. Gray: *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston/ Dordrecht/ London 1991.

[9] J.C. Gower: Maximal Predictive Classification. *Biometrics*, 30, 1974, pp. 643 - 654.

[10] M. Gyllenberg and T. Koski: A Taxonomic Associative Memory Based on Neural Computation. *Binary*, Vol. 7, 1995, pp. 61 - 66.

[11] M. Gyllenberg and T.Koski: A Taxonomic Associative Memory of Most Typical Organisms Based on Neural Computation. *University of Turku, Institute of Applied Mathematics, Research Reports A 4*, Nov. 1994.

[12] M. Gyllenberg, T. Koski and M. Verlaan: On Quantization of Binary Vectors Using Stochastic Complexity. *Proceedings of the IEEE International Symposium on Information Theory*, IEEE 1994, p.390.

[13] M.Gyllenberg, T. Koski, E.Reilink and M.Verlaan: Probabilistic Aspects of Numerical Identification in Microbiology. A.N. Shiryaev et. al. (ed.): *Proceedings of the Fourth Russian-Finnish Symposium on Probability Theory and Mathematical Statistics.* TVP Science Publishers, Moscow, in the press.

[14] S. Haykin. *Neural Networks. A Comprehensive Foundation.* IEEE Press, Macmillan College Publishing Company, New York, 1994.

[15] J. Hertz, A. Krogh and R.G. Palmer: *Introduction to the Theory of Neural Computation.* Santa Fe Institute Lecture Notes Vol. 1, Addison Wesley, Redwood City, CA, 1991.

[16] M.J.J. Holt and S.Semnani: Convergence of Back-Propagation in Neural Networks Using a Log-Likelihood Cost Function. *Electronics Letters*, Vol. 26, No. 23, 8th November, 1990, pp. 1964 - 1965.

[17] R.A. Jacobs: Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks*, 1(4), 1988, pp. 295-307.

[18] S. Kak: New algorithms for training feedforward neural networks. *Pattern Recognition Letters*, 15, 1994, pp. 295 - 298.

[19] P. Kanerva: *Sparse Distributed Memory.* 2nd Printing, A Bradford Book, MIT Press, Cambridge Mass., London, 1990.

[20] T. Kohonen: *Self-Organization and Associative Memory.* 3rd Edition, Springer Verlag, New York, 1989.

[21] T. Lahti: *Supervised training of a neural net by minimizing crossentropy for classification of feature vectors in the binary hypercube.* M.Sc.-thesis, 1995:053 E, Luleå Univ. of Techn., Luleå, February 1995.

[22] M.D. Richard and R.P. Lippman: Neural Network Classifiers Estimate Bayesian Posteriori Probabilities. *Neural Computation*, 3, 1991, pp. 461-483.

[23] B.D. Ripley: Neural Networks and Related Methods for Classification. *Journal of the Royal Statistical Society*, series B, vol. 56, 1994, pp. 409 - 456.

[24] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley and B.W. Suter: The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function. *IEEE Transactions on Neural Networks*, Vol. 1, December 1990, pp. 296 - 298.

[25] J.E. Shore and R.W. Johnson: Axiomatic Derivation of the Principle of Maximum Entropy and the Principle of Minimum Cross-Entropy. *IEEE Transactions on Information Theory*, Vol. IT-26, Jan. 1980, pp. 26 - 37.

[26] S.A. Solla, E. Levin and M. Fleisher: Accelerated Learning in Layered Neural Networks. *Complex Systems*, Vol. 2, 1988, pp. 625 - 640.

[27] M. Verlaan: *Classification of Binary Vectors by Maximal Predictivity and Stochastic Complexity.* Techn. Rep. Luleå Univ. of Techn., 1993:08 T, 1993.

[28] W.R. Willcox, S.P. Lapage and B. Holmes: A Review of Numerical Methods in Bacterial Identification. *Antonie van Leeuwenhoek*, 46, 1980, pp. 233 - 299.

[29] B.S. Wittner and J.S. Denker: Strategies for Teaching Layered Networks Classification Tasks. *Neural Information Processing Systems*, American Institute of Physics, 1988, pp. 850- 859.

[30] T. Young and T. Calvert: *Classification and Pattern Recognition.* American Elsevier, New York, 1974.