Attribute Suppression with Multi-Layer Perceptron[*]

Fabrice ROSSI[†]
THOMSON-CSF/AIRSYS
7-9, rue des Mathurins
92221 BAGNEUX, FRANCE
e-mail: rossi@ceremade.dauphine.fr
fax: 33 1 40 84 39 09

Abstract

In this paper, we introduce a method that allows to evaluate efficiently the "importance" of each coordinate of the input vector of a neural network. This measurement can be used to obtain informations about the studied data. It can also be used to suppress irrelevant inputs in order to speed up the classification process conducted by the network.

1 Introduction

Solving a "real world" classification problem is a challenging task. The first problem (the *learning* problem) is to construct a classifier with satisfactory recognition rate. The second problem (the *implementation* problem) is to implement the obtained classifier efficiently enough so that it can be used for real problems (real time target recognition for instance). In the case of *supervised learning* studied in this paper, the classes are fixed and a set of already classified examples (the *learning set*) is provided for the classifier construction. With this knowledge, many different algorithms can be used to construct a "good" classifier (e.g. nearest-neighbor or linear classifier, feed-forward neural network, etc). Among them, neural networks seem to be a good choice: they can approximate Bayes optimal discriminant function [14] and are therefore theoretically the best way to solve classification problems. Moreover, whereas they can be quite difficult to train, the calculations they made remain simple enough to allow efficient implementations. A lot of work have been done in order to reduce the learning time for neural networks (e.g. [7, 17]). The implementation problem is also widely studied and answers are obtained with the help of specialized hardware (such as neural chips, e.g. [5, 9]) or with parallel implementation (e.g. [10]).

Unfortunately, in some applications, specialized hardware or parallel computers cannot be used (due to price or size constraints). In this case, the only way to speed up the implementation is to modify the classification process. In general, this process has two parts: first, a *pre-processing phase* extracts relevant attributes from raw data (e.g. auto-regressive coefficients in a speech recognition problem); second, the attributes are submitted to a classifier that computes which class the given input belongs to (this is the *pure classification phase*). In some cases, it is possible to reduce the "size" of the classifier, therefore reducing its computation time. For Multi-Layer Perceptrons (MLP), this goal can be achieved by suppressing connections between neurons (or even complete neurons): the goal is not only to reduce the classification time but also to improve the generalization performances of the network [6]. But it is impossible to reduce the network below a minimum size without loosing accuracy in the classification. Moreover, connection suppression methods are not general and cannot be easily applied to other classifiers.

Another time saving method is to suppress the less significant attributes: we save the time needed to compute them and we reduce the input size of the pure classification algorithm which will in general reduce its processing time. This method is closely related to projection methods (e.g. [2, 8, 15]) in which new attributes are computed as a function of the original attributes. The goal is of course to reduce the number of attributes submitted to the classifier, in order to simplify its task. This reduces also its computation time, but in order to calculate the new attributes, we need in general to compute all the original attributes. Moreover, the projection in itself introduces additional computation. Therefore, this type of attributes *reduction* does not decrease computation time as much as an attribute *suppression* method.

[*]This work was performed on Mrs Kim K. PHAM's responsibility, at THOMSON-CSF/AIRSYS.
Published in CESA Multiconference 96.
Available at
http://apiacoa.org/publications/1996/cesa96.pdf
[†]Up to date contact informations for Fabrice Rossi are avaible at http://apiacoa.org/

In this paper, we present a new method that allows to choose automatically which attributes to suppress. We assume that the studied classification task can be performed by a parametric classifier which can be for instance a MLP or a Radial Basis Function Network (RBFN)[11]. An analysis of this classifier allows to measure how much the classification depends on a particular attribute and therefore to suppress the less important ones. Previous results [4] imply that this method can be efficiently used for totally arbitrary feed-forward neural networks.

The remainder of this paper is organized as follows. Section 2 introduces the mathematical aspect of our attribute suppression method and compares it to existing methods. Section 3 gives some experimental results on artificial and real-world data.

2 Input Study

2.1 Classic methods

Classic methods can be separated into two groups: attribute selection methods and attribute projection methods.

Attribute selection:

In this method, the most relevant attributes are chosen with an iterative process: a first attribute is chosen in order to maximize a given criterion; then, a second one is chosen so that the pair maximizes the same criterion (but the first attribute remain fixed), etc. This approach has been used for decision tree construction for instance (e.g. [16]). Its main drawback is that it cannot solve efficiently complex problems in which classification is impossible *separately* with each axis but can be done with only few of them used *together*.

A far more powerful version of this algorithm has been proposed by Roberto Battiti ([1]). It is based on *mutual information* which provides a reliable way to compute the statistical relationship between a feature and the classification task. This method gives good results but not as good as our approach, as shown in section 3.

Attribute projection:

Projection methods are widely used in order to reduce the size of the problem: we compute combinations of the original attributes in order to obtain fewer new attributes. As long as we are sure to keep the most significant part of the data, we simplify here the task of the classifier as it will process fewer data. In the case of linear projection, computed by Principal Component Analysis (PCA), or Discriminant Analysis (DA), we can easily decide if some original attributes are useless. Indeed, if P is a projection matrix, the norm of its i-th column measures the importance of the i-th original attribute. If this norm is very small compared to the ones of the other columns, the new attributes do not depend significantly on the i-th original attribute, which can therefore be discarded. The main drawback of this method is that the projection is linear and therefore will not be able to "understand" a highly non linear classification problem. We might use a non-linear projection method (e.g. [15]) but in this case, the analysis of the relationship between original and projected attributes become very difficult and is in fact a special case of our method (see the following section).

2.2 Global attribute evaluation

The method presented in this section allows to choose which attributes to suppress in an efficient and reliable way. The key idea is to analyze an already trained classifier in order to measure how much its calculation depends on each attribute: the evaluation is global as in projection methods. Moreover, the method can be applied to non linear classifiers and therefore solve the linearity limitation.

Let $F(x,w)$ be a parametric classifying function (e.g. a MLP): x is an input vector, w is a weight vector that allows to modify the computation performed by F (e.g. the connection weights in a MLP) and $F(x,w)$ is the output of the function which belongs to $[0,1]^c$, where c is the number of classes of the problem. Let C_k be the studied k-th class and let $\chi_{C_k}(x)$ be the membership function corresponding to C_k, i.e. $\chi_{C_k}(x) = 1 \Leftrightarrow x \in C_k$. Let $C(x)$ be the perfect classifying function, i.e. $C(x) = (\chi_{C_1}(x), \chi_{C_2}(x), \ldots, \chi_{C_c}(x))$. The goal of the learning phase is to find w such that $F(x,w) \simeq C(x)$. In general w is chosen in order to minimize some distance criterion between $F(x,w)$ and $C(x)$ (for instance, the total quadratic error, $E(w) = \sum_{x \in T} \|F(x,w) - C(x)\|^2$, where T is the training set of the classification task).

Let us assume that we can compute the differential of F with respect to its first variable (i.e., the input vector), called $\frac{\partial F}{\partial x}(x,w)$. The k-th output of F, $F(x,w)_k$, is a good approximation of χ_{C_k}, the membership function corresponding to class C_k. Therefore, $\frac{\partial F_k}{\partial x_i}$ gives a good idea of the boundaries of class C_k and can be used in order to obtain the general direction of these boundaries. If a classification is possible, we can assume that each class consists of a set of clusters. Inside a cluster of class C_k elements, χ_{C_k} is constant and therefore F_k is also almost constant, which means $\frac{\partial F}{\partial x}(x,w) \simeq 0$. But at the cluster boundaries, $\chi_{C_k}(x)$ changes very rapidly. As a good approximation of χ_{C_k}, $F_k(x,w)$ will change very rapidly around this boundaries, and therefore, we can assume that high values of $\left\|\frac{\partial F_k}{\partial x}(x,w)\right\|$ will design cluster boundaries. Moreover, near a boundary, $\frac{\partial F_k}{\partial x}(x,w)$ will be perpendicular to the local separation hyperplane. Therefore, a low value for $\left|\frac{\partial F_k}{\partial x_i}(x,w)\right|$ (compared to $\left\|\frac{\partial F_k}{\partial x}(x,w)\right\|$) shows that the boundary normal is perpendicular to the i-th coordinate axis and therefore that the local separation hyperplane contains this axis: the i-th coordinate of the input vector is locally useless for separating this cluster from other classes. Therefore, $\frac{\left|\frac{\partial F_k}{\partial x_i}(x,w)\right|}{\left\|\frac{\partial F_k}{\partial x}(x,w)\right\|}$ is a good measure of the local importance of the i-th coordinate axis.

The remaining problem is to combine the $\left|\frac{\partial F_k}{\partial x_i}(x,w)\right|$ when x belongs to C_k in order to obtain a global understanding of the relative importance of the different coordinate axis. On a theoretical point of view, it will be interesting to compute the mean normal vector of C_k boundary. But C_k may contain several separated clusters for which the boundaries are parallel but with opposite normal vectors: therefore, computing the integral of $\frac{\partial F_k}{\partial x_i}(x,w)$ on C_k boundary is not really meaningful.

In order to obtain a simple criterion, we define the following matrix:

$$S_{k,i} = \int_{x \in \partial C_k} \frac{\left|\frac{\partial F_k}{\partial x_i}(x,w)\right|}{\left\|\frac{\partial F_k}{\partial x}(x,w)\right\|} dx, \tag{1}$$

where ∂C_k is the boundary of class C_k.

This value is quite simple to compute: when $\left\|\frac{\partial F_k}{\partial x}(x,w)\right\|$ is above a given threshold, we assume that $x \in \partial C_k$, even if $x \notin C_k$. Therefore, we just have to apply the following approximate formula:

$$S_{k,i} = \sum_{x \in T,\ \left\|\frac{\partial F_k}{\partial x}(x,w)\right\| > \epsilon} \frac{\left|\frac{\partial F_k}{\partial x_i}(x,w)\right|}{\left\|\frac{\partial F_k}{\partial x}(x,w)\right\|}, \tag{2}$$

where ϵ is the chosen threshold.

The obtained value $S_{k,i}$ is therefore the global score associated to the i-th coordinate axis as a classifying axis for class C_k. Finally, we can define a mean score, the vector S as:

$$S_i = \frac{1}{c} \sum_{k=1}^{c} S_{k,i} \tag{3}$$

S_i is the mean global score associated to the i-th coordinate axis as a classifying axis.

Each coordinate axis of the input space is associated to an attribute of the input vector. Therefore, a high axis score is equivalent to an important attribute: in order to suppress attributes, we just have to discard the one with the lowest score.

2.3 Boundary examples

As explained in the previous section, the criterion vector S is computed with the help of boundary examples. In the case of an easy classification problem, theses examples may not exist: if the clusters of the different classes are well separated, the boundaries computed by the classifier will not be close to the clusters but right in the “middle”, where no example exists. In this case, the criterion vector cannot be computed.

For neural networks, the boundaries are moved during training reflecting the weight changes. Before reaching the perfect case, the number of misclassified examples (and therefore of boundary examples) decreases. Therefore, we can compute the criterion vector S periodically during the training until the number of boundary examples reaches a minimum value. If S can be computed quickly, this method will not increase significantly the training time.

2.4 Links with previous works

The method explained in the two previous sections is closely related to an algorithm introduced in [13]. In this article, the authors introduce an attribute ranking method based on first order differentials. There are two important differences between the method presented here and their algorithm:

- Priddy and al. combine the individual differential $\left|\frac{\partial F_k}{\partial x_i}(x,w)\right|$ without normalization ;
- they take into account every examples (and even additional points which are not examples) without focusing on boundary examples.

In fact, the main justification of Priddy and al. is a statistical one, whereas we are working on geometrical arguments, which are in our opinion more suited to the attribute suppression goal.

2.5 Feed-forward neural network case

We have demonstrated in a previous paper [4, 3], that an extended back-propagation algorithm can be defined for arbitrary feed-forward neural networks (including in the same framework MLP, RBFN and Wavelet Networks [18] for instance). This algorithm allows to compute efficiently the differential of $F(x,w)$ with respect to its input x, if F is the output of a neural network (with w as generalized weight vector). It allows also to compute $\frac{\partial \mathcal{E}}{\partial w}(x,w)$, where $\mathcal{E}(x,w)$, is the error made by the network on example x (this gradient can be used to train the generalized network). The time needed to compute $\frac{\partial F}{\partial x}(x,w)$ is approximately d times the time needed to compute $\frac{\partial \mathcal{E}}{\partial w}(x,w)$ where d is the output dimension of F.

For a MLP (or a RBFN) with c output nodes (i.e., a c class problem), computing $\frac{\partial F}{\partial x}(x,w)$ involves performing c backward computation passes, each one similar to the classic back-propagation calculation. As a backward pass is not exactly the back-propagation calculation, the time needed to compute $\frac{\partial F}{\partial x}$ *is not exactly* equal to c times the time needed to compute $\frac{\partial \mathcal{E}}{\partial w}$, but we can assume that computing the criterion vector S defined in the sub-section 2.2 takes approximately the time needed to perform c back-propagation epochs (i.e., computing $\frac{\partial \mathcal{E}}{\partial w}(x,w)$ c times for each x in the training set). Therefore, as long as the criterion is not computed too frequently (e.g., every 10 epochs), its calculation will not slow down the training process.

3 Experiments

3.1 Synthetic data

We first show in this section a simple example that illustrates our method. This example is a modified version of the well known XOR problem. We have two classes C_1 and C_2. Each class is made of two clusters ($C_{1,1}$, $C_{1,2}$, etc.). Each cluster obeys to a Gaussian distribution in which both coordinates are independent and have 1.0 as standard deviation. The cluster centers are:

$C_{1,1}$	$C_{1,2}$	$C_{2,1}$	$C_{2,2}$
$(0,-4)$	$(0,4)$	$(-4,-4)$	$(4,4)$

Figure 1 shows the training set (500 examples in each cluster) after normalization (the total set is centered and scaled so that its standard deviation is 1.0 on both axis).

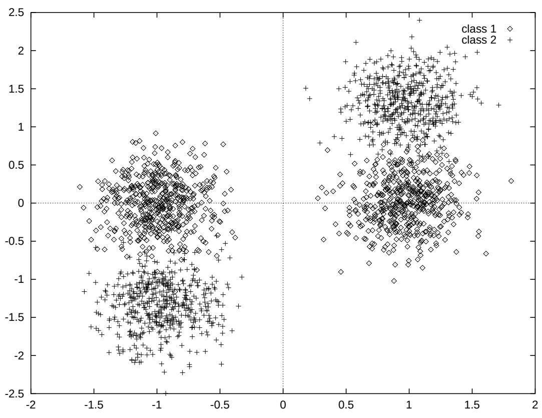

Figure 1: Training set

Obviously, the classification can be performed only with the y axis (in fact the x axis is useless for classification purpose). The problem is simple for a MLP but cannot be solved by a linear method: we need at least two straight lines to separate the classes.

A PCA gives for the projection the matrix $P_{pca} = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$. A DA gives another projection: $P_{da} = (0.76, 0.65)$. In both cases, it is impossible to use the matrices in order to choose which attribute to suppress (in fact, a DA based method will select the wrong attribute).

We have trained five randomly initialized MLP (2 layers, 2 neurons in the hidden layer and 2 output neurons, i.e. the minimal architecture, called a 2-2-2 MLP) on the given training set. After 30 iterations of Polak Ribiere Conjugate Gradient, PRCG ([17]), the classification rate and criterion vector for each MLP are given in the following table:

	1	2	3	4	5
classification rate	73 %	97 %	97 %	97 %	97 %
S	$(228.8, 329.2)$	$(289.5, 2042)$	$(169.2, 1299)$	$(81.19, 844.7)$	$(422.9, 2619)$

The first MLP failed to separate precisely the two classes because of a bad starting point. For the other, the ratio between S_2 and S_1 is larger than 6.2: this allows to choose the y axis as the one to keep. When this axis is chosen, a MLP with 2 layers, 1 input, 2 hidden neurons and 2 output neurons (i.e. a 1-2-2 MLP), obtains after 30 iterations of PRCG a mean classification rate of 96.5 %. The method described in [13] gives very similar results, with a high ratio between the importance of input 1 and 2. This method selects therefore the y axis and overcomes also linear method limitations (this is also the case of Battiti's algorithm [1]).

3.2 Real data

Some experiments were conducted on real world data. We have chosen the PROBEN1 data set ([12]) which is easily available by `ftp`[1]. We show here results based on the `thyroid` problem, in which biological informations are used in order to determine if the observed thyroid has over function, under function or normal function (therefore, there are three classes). The input vector has 21 attributes. We have not followed here the Proben rule, in which the classification rate is computed on the whole test set, regardless of the relative pattern number for each class. Indeed, in the `thyroid` problem, there are 92.6 % normal thyroid samples. Therefore, a classifier that gives always the normal answer obtain 92.6 % as classification rate, whereas its result is totally wrong. We have therefore chosen to compute the classification rate for each class and to take the mean of the three rates as the performance criterion of the classifier. In the case of the

[1]at `ftp.ira.uka.de:/pub/neuron/proben1.tar.gz`

previous classifier, we obtain therefore 30.87 % which gives a good idea of its bad performances. The goal of our simulation is to keep as few attributes as possible. We tried several different methods:

3.2.1 Neural methods

We trained a simple MLP with 21 inputs, 16 hidden neurons and 3 output neurons (i.e., a 21-16-3 MLP), from five different starting weight sets. After 500 iterations of PRCG, the best MLP (selected with the help of the validation set) obtain 95.60 % as classification rate on the test set (we have used the `thyroid1` set in which the inputs where rescaled to mean 0 and standard deviation 1). The method described in the previous section gives the following criterion vector (with $\epsilon = 0.001$):

Attribute	17	3	8	19	21	5
Value	105	31.7	23.0	10.8	9.53	6.39
Attribute	10	20	18	7	1	14
Value	6.25	5.12	4.53	3.58	3.17	3.07
Attribute	13	2	6	4	11	16
Value	2.87	2.79	2.02	1.79	1.61	1.33
Attribute	12	9	15			
Value	1.28	1.04	0.61			

This vector allows us to choose to keep 5 attributes: $(17, 3, 8, 19, 21)$.

The method described in [13] gives a different ordering and the five best attributes are: $(17, 3, 8, 10, 19)$.

3.2.2 Linear methods

We tried two different methods. A DA first give a projection matrix P from $\mathbb{R}^{21}$ to $\mathbb{R}^2$ (the maximum size of the projection space for the DA is the number of classes minus one). An analysis of its columns allows to rank the attributes. In our problem, the five best attributes are $(17, 3, 10, 19, 2)$.

We computed also the optimal linear classifier for our problem. We obtained in this case a “projection” matrix L, from $\mathbb{R}^{21}$ to $\mathbb{R}^3$. In order to use it to rank the attributes, we can apply a technique very close to the general one presented in subsection 2.2. In fact, the relative importance of attribute j for class k is given by the ratio of the j-th coordinate of the k line of L to the norm of this line (if we consider that $F(x) = Lx$ is the linear classification function, we have obviously $\frac{\partial F}{\partial x}(x) = L$ and we obtain exactly the criterion matrix of subsection 2.2 with the simple ratio computed here). Therefore, the total importance of attribute j is the sum of this ratio for k ranging from 1 to c, the number of classes. In our problem, the five best attributes are $(17, 19, 3, 10, 18)$.

It is important to notice that the linear optimal classifier has a performance rate of 58.37 % (on the *learning* set), which is really bad compared to the MLP.

3.2.3 Statistical Method

We have also used Battiti's algorithm, as described in [1]. This method combines two measures: the mutual information between a feature and class information, $MI(f, \mathcal{C})$, and the mutual information between a feature and the already selected features, $MI(f_1, f_2, \dots, f_p, f)$. The algorithm chooses the most important feature f_1 (highest mutual information between it and class information). Then, it chooses the second feature as the one which maximizes $MI(f, \mathcal{C}) - \lambda MI(f_1, f)$, etc. The parameter λ has to be heuristically chosen. Battiti says that values ranging from 0.5 to 1.0 give good results.

In order to have a fair comparison with our method, which does not tale into account redundancy, we have chosen two value for λ: 0 (no redundancy information) and 0.5. These choices give the following attribute sets:

λ	Attribute set
0	$(17, 21, 19, 18, 20)$
0.5	$(17, 21, 15, 5, 13)$

The first set is called Battiti 1 and the second one is Battiti 2.

3.2.4 Comparison

In order to compare the different methods (and in fact the selected attribute sets), we have trained a 5-16-3 MLP on the `thyroid1` sets. 10 different randomly chosen starting weight sets were used for the MLP. The performance for one starting point is the classification rate on the test set for the best MLP obtained after 500 iterations of PRCG (the MLP is selected with the help of its mean square error on the validation set). The following table shows the mean performance for the 10 starting weight sets and both attribute sets (in the table, the neural method proposed in [13] is called the Priddy method):

attribute set	mean classification rate
Neural based	98.23 %
Battiti 2	96.49 %
Linear based	95.06 %
DA based	94.39 %
Priddy method	93.86 %
Battiti 1	93.77 %

attribute set	standard deviation
Neural based	0.7563
Battiti 2	1.65
Linear based	2.135
DA based	1.278
Priddy method	1.887
Battiti 1	1.92

This table shows that our neural network based selection allows better performances than all other methods. Therefore, our method can overcome some limitations implied by the linear ones. Moreover, the use of boundary examples overcomes limitations of the method proposed in [13]. Our method can also overcome the mutual information based method proposed in [1].

4 Conclusion

In this paper we have introduced a new method that allows to suppress data attributes in a classification task. The goal of this suppression is to reduce the preprocessing and classification times. It is based on an analysis of the calculation performed by a parametric classifier such as a multi-layer perceptron. With the help of previous results, we have shown that this method was easy to use for arbitrary feed-forward neural networks and that it does not slow down significantly their training process. Experiments conducted on synthetic and real data show that this method is efficient and can therefore be used for real world applications.

References

[1] Roberto Battiti. Using Mutual Information for Selecting Features in Supervised Neural Net Learning. *IEEE Trans. On Neural Networks*, 5(4):537–550, July 1994.

[2] Gautam Biswas, Anil K. Jain, and Richard C. Dubes. Evaluation of projection algorithms. *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-3(6):701–708, November 1981.

[3] Cédric Gégout, Bernard Girau, and Fabrice Rossi. A General Feed-Forward Neural Network Model. Technical report NC-TR-95-041, NeuroCOLT, Royal Holloway, University of London, May 1995. Available at `http://apiacoa.org/publications/1995/neurocolt1995.pdf`.

[4] Cédric Gégout, Bernard Girau, and Fabrice Rossi. Generic Back-Propagation in Arbitrary Feedforward Neural Networks. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, *Int. Conf. on Artificial Neural Nets and Genetic Algorithms*, pages 168–171, Alès, April 1995. Springer Verlag.

[5] Il-Song Han. Modular neural network implementation of analog-digital mixed operation. In *Int. Conf. on Neural Networks*, pages 709–712. IEEE, 1993.

[6] Babak Hassibi, David G. Stork, and Gregory J. Wolff. Optimal brain surgeon and general network pruning. In *Int. Conf. on Neural Networks*, pages 293–299. IEEE, 1993.

[7] Nicolaos B. Karayiannis. Accelerating the training of feed-forward neural networks using generalized hebbian rules for initializing the internal representation. In *Int. Conf. on Neural Networks*, volume I, pages 32–37, Orlando (Florida), June 1994. IEEE.

[8] Matrin A. Kraaijveld, Jianchang Mao, and Anil K. Jain. A nonlinear projection method based on kohonen's topology preserving maps. *IEEE Trans. Neural Networks*, 6(3):548–559, May 1995.

[9] P. Masa, K. Hoen, and H. Wallinga. 70 inputs, 20 nanoseconds pattern classifier. In *Int. Conf. on Neural Networks*, volume 3, pages 1854–1859, Orlando (Florida), June 1994. IEEE.

[10] Alain Petrowski, Gérard Dreyfus, and Claude Girault. Perfromance analysis of a pipelined backpropagation parallel algorithm. *IEEE Trans. On Neural Networks*, 4(6):970–981, Nov. 1993.

[11] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–1497, September 1990.

[12] Lutz Prechelt. PROBEN1 — A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, September 94. `FTP site: ftp.ira.uka.de (file /pub/papers/techreports/1994/1994-21.ps.Z).`

[13] Kevin L. Priddy, Steven K. Rogers, Dennis W. Ruck, Gregory L. Tarr, and Matthew Kabrisky. Bayesian selection of important features for feedforward neural networks. *Neurocomputing*, 5:91–103, 1993.

[14] Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, Mark E. Oxley, and Bruce W. Suter. The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE Trans. On Neural Networks*, 1(4):296–298, December 1990. .

[15] J. W. Sammon Jr. A nonlinear mapping for data structure analysis. *IEEE Trans. Computer*, 18:401–409, May 1969.

[16] I. K. Sethi and Sarvarayudu G. P. R. Hierachical classifier design using mutual information. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, 4(4):441–445, July 1982.

[17] Patrick Van Der Smagt. Minimisation methods for training feedforward neural networks. *Neural Networks*, 7(1):1–11, 1994.

[18] Qinghua Zhang and Albert Benveniste. Wavelet networks. *IEEE Trans. On Neural Networks*, 3(6):889–898, November 1992.