A Non-Iterative Method for Training Feed Forward Networks.

Wouter F. Schmidt, Martin A. Kraaijveld and Robert P.W. Duin

Pattern Recognition Group
Faculty of Applied Physics
University of Technology Delft
The Netherlands
e-mail: wouter@duttnph.tudelft.nl

Abstract

In this paper a method is described to determine the weights of a feed forward network, with only one hidden layer, to perform a certain classification task. The Wiener least squares solution is part of this algorithm and is used to calculate weights for the input layer and the output unit. This is a non-iterative method and it is from a computational point of view much faster than standard methods.
The presented algorithm is applied to three data sets with know statistical properties. The networks are learned with relative small data sets and the final networks are capable to classify unknown patterns with classification errors approximating the Bayes error.

Introduction

Although single layer networks have shown to be very limited in their capabilities, their training procedures are very efficient and well understood (see e.g. Minsky and Papert[7], Duda and Hart[2], Devijver and Kittler[1], Nilsson[8], Widrow and Stearns[13], etc). Multi layer networks on the other hand, appear to be much more powerful, but training them appears to be more difficult and less tractable in an analytic sense. For example, the back propagation learning rule, which was proposed by Rumelhart et al[10]., appears to be very slow and there is a great risk that the learning procedure results in a suboptimal solution.
Although many authors have proposed improvements to the back propagation rule in order to speed up the algorithm (e.g. Silva and Almeida[11], Stornetta and Huberman[12] and Jacobs[6]) or to minimize the influence of local minima, the resulting learning procedure is still based on an iterative process to compute the weights of the network.
In this paper we propose a new and non iterative method to compute the weights of a network with one hidden layer. Our method builds upon other non iterative methods to solve this problem, and is motivated by previous work of Haersma Buma and Duin[4]. Tests of our method on various realistic and non trivial problems indicate that the procedure is very fast and appears to result in networks with a high performance.
Note that although our method is designed for networks with one hidden layer, this seems to be no important restriction, since there are a number of theoretical results that show that one hidden layer is sufficient for most problems (Hornik [5] and Funahashi [3]).
The structure of this paper is as follows. We will start with the review of some theory on non iterative procedures for single layer networks. Then we will discuss multi layer networks and we will present the new algorithm to compute the weights for a network with one hidden layer. The next paragraph will discuss some experiments in which we tested our method, followed by a discussion and some conclusions.

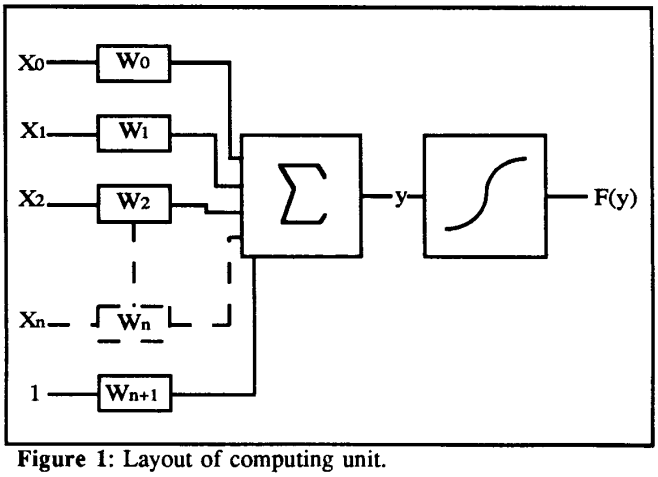


Figure 1: Layout of computing unit.

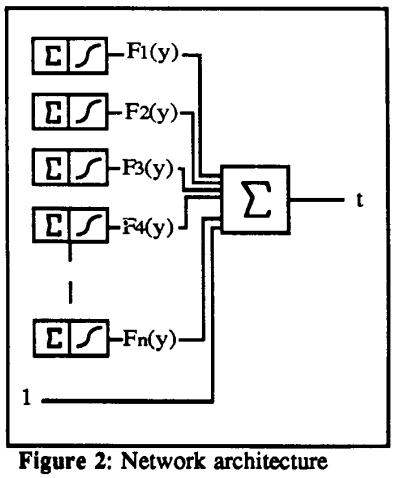


Figure 2: Network architecture

0-7803-0164-1/91/0000-0019$01.00©1991 IEEE

Single layer networks and the Wiener weight solution

The unit of a feed forward network, see figure 1, consists of a processing element which is capable to calculate the weighted sum of the inputs to the unit. Applied to this weighted sum is a so called squashing function F(x) which maps this value in the range between 0 and 1 (see figure 1). A squashing function is a continuous, monotonously increasing function, with the following properties:

$$\text{Lim}_{x\to-\infty} F(x) = 0 \qquad \text{Lim}_{x\to\infty} F(x) = 1 \tag{1}$$

Note that the inverse function of the squashing exists due to its monotonously property. In mathematical formulation the processing element can be described as (see also figure 1):

$$F(y) = F(\sum w_i x_i) = F(\mathbf{W}^T\mathbf{X}) \tag{2}$$

Here W and X are the weight and input vectors respectively* :

$$W = [w_1, w_2, ...,w_n, w_{n+1}]^T \qquad X = [x_1, x_2,.....,x_n, 1]^T \tag{3),(4}$$

The first part of the processing unit (which calculates the weighted sum y) is sometimes called adaptive linear combiner or ALC (see Widrow and Stearns [13]). The weight vector W is set in such a way that the processing unit performs the desired task. The performance of this unit is measured (with a certain data set) with the following function:

$$E^2_{\text{transfer}} = \sum_{\text{patterns}} (t - F(\mathbf{W}^T\mathbf{X}))^2 \tag{5}$$

In this formula the desired response is t and the unit response is $F(\mathbf{W}^t\mathbf{X})$. Because the inverse function of the squashing function exists it is possible to optimize the following error criterion:

$$E^2_{\text{alc}} = \sum_{\text{patterns}} (F_{\text{inv}}(t) - \mathbf{W}^T\mathbf{X})^2 \tag{6}$$

F_{inv} is the inverse function of the squashing function F. The main advantage to optimize equations 6 is that the solution of this equation can be obtained directly. So no adaptive process is needed to find the weight vector W which minimizes the error criterion. We use this method for classification problems and Finv(t) is constant for every sample which is member of the same class. This method is now equivalent to the Least Mean Square Error Procedure of Devijver and Kittler [1] page 151. The weight vector W for which equation 6 is minimal is as follows:

$$\frac{\partial E^2_{\text{alc}}}{\partial \mathbf{W}} = 2\mathbf{R}\mathbf{W}^* - 2\mathbf{P} = 0 \;\rightarrow\; \mathbf{R}\mathbf{W}^* = \mathbf{P} \tag{7}$$

Here $\mathbf{R} = \sum_{\text{patterns}} \mathbf{X}\mathbf{X}^T$ and $\mathbf{P} = \sum_{\text{patterns}} F_{\text{inv}}(t)\mathbf{X}^T$, which is the input correlation matrix and input-target correlation vector respectively. The optimal weight vector $\mathbf{W}^*$ is called the Wiener vector and can be found by solving the linear set of equations by standard numerical methods. (The mathematical proof can be found in Widrow and Stearns [13] page 19.)

A non iterative method for training three layer feed forward networks.

We use feed forward networks with only two active layers in this article (see figure 2). From the unit in the output layer the transfer function is removed so this unit only calculates the weighted sum of the outputs of the units in the hidden layer. If we call the weights of the output unit m_i, the network calculates the following sum:

$$f_{\text{net}}(x) = \sum_{i}^{\text{units}} m_i F(\mathbf{W}_i^t\mathbf{X}) \tag{8}$$

---

* Note that the W_{n+1} weight is always multiplied by one and can be regarded as a threshold value.

The network performance is measured with an analogue function as defined in the previous paragraph:

$$E^2_{net} = \sum_{patterns} (t - \sum_{i}^{units} m_i F(W_i{}^t X))^2 \qquad (9)$$

The main problem of these networks is how to find the set of weights which minimizes this error criterion. A direct method like the Wiener solution can not be applied due to the non linearity in the evaluation function.

In the following we will describe the new method to compute the weights for a network with one hidden layer. The units in the hidden layer have sigmoïd squashing functions:

$$F(x) = \frac{1}{1 + e^{-x}} \qquad (10)$$

We start our learning procedure with a network with one hidden unit. For this unit we can calculate the optimal Wiener weight vector using the network target values for this unit. The unit input weights are set to these weights. For the weights of the output unit we calculate the optimal Wiener weight solution (including the threshold weight) using the first unit output and the network target value.
Now the weights of the first input unit are fixed and we subtract the current network output from the targets for all learning samples, in order to create a new learning set, with new targets. The *new* target is scaled between 0.1 and 0.9 because the new targets may be negative or larger than one, which will cause problems if we want to calculate the P vector for the new unit. This scaling can not cause any problems because the weights of the output unit are calculated afterwards.
A second hidden unit is added now and the weights are optimized with the new target, equivalent to method used for the first unit. The weights are set to the Wiener solution and the weights of the output units are calculated with the Wiener solution as well, but this time with the outputs of the two units (and the threshold weight). The data set is evaluated again for all the learning patterns and again a *new* target is constructed by subtracting the current output from the desired output values. This process is repeated until for all input units the weights are calculated. This results in the following algorithm:

Step 1 *Use network target values to calculate the Wiener weight vector and set the first unit ($F_1(y)$ figure 2) to these weights. Calculate the weight and threshold of the corresponding output summating unit, also with the Wiener optimal vector.*

Step 2 *For each sample: evaluate the current network and subtract the current output from the target values, to determine the new target values. Scale the new target values between 0.1 and 0.9.*

Step 3 *Add a new hidden unit and set the weights of the new unit to the Wiener vector for the new target values. Set the output unit to the Wiener vector, using the original target values.*

Step 4 *If not all the units in the hidden layer are set go to step 2.*

With this method the first unit is used to approximate the desired signal and the second unit is used to correct for the error made by the first unit. The third unit is used to correct for the error introduced by the first and second unit etc. At a certain point the optimization can not find a proper vector any more and this will result in units which do not add any information to the desired target function. This is a severe problem because the **R** matrix of the output unit will become ill conditioned which will result in unstable numerical solutions. It is advisable to use Singular Value Decomposition methods (see Press e.a. [9]) when solving equation 7.

Experiments

We used the method explained in the previous paragraph for three classification problems. The statistics of all the data sets are known and the learn and test data are artificially generated. All three data sets contain two dimensional Gaussian distributed ($N(\mu,\sigma)$) samples with two different classes. The statistics are listed in table 1. For every data set a learning set with 100 samples per class is generated and a test set with 1000 samples per class. For data set 1 and 2 the class overlap between class A and B is 20%, which means that the optimal classifier has a performance of 80% (Bayes error). The optimal classifier of data set 1 is a linear function and the optimal classifier for data set 2 is a circle at the origin with radius $\approx$ 1.85. Data set 3 has less overlap and the optimal *linear* classifier has an error of about 88%. The Bayes classifier for this data set is a hyperbolic function, which has a performance which is a few percent better than the linear classifier.

Table 1: Statistics of data sets used in the experiments.

Data set	Class A Feature x_1	Class A Feature x_2	Class B Feature x_1	Class B Feature x_2
1	N(0.2, 0.333)	N(0.2, 0.333)	N(-0.2, 0.333)	N(-0.2, 0.333)
2	N(0, 1.645)	N(0, 1.645)	N(0, 0.6076)	N(0, 0.6076)
3	N(1,1)	N(1, 0.5)	N(2, 0.1)	N(0,2)

For the three data sets networks are computed with 1 to 10 units in the hidden layer. The determinant of the **R** matrix of the output layer is calculated to show how the system of equations 7 deteriorates during the process. The mean square error on the learning and test sets is measured together with the percentage of properly classified samples.
The network simulation is performed on a Sun 4 workstation with custom software written in 'C'. For the numerical calculation we used the Numerical Recipes in C library (see Press et. al [9]), with double precision floating point numbers (53 bit mantissa). Note that we used LU-decomposition for solving equation 7 although Singular Value Decomposition is a better choice. However, we used the LU-decomposition method because it is illustrative for the problem mentioned at the end of the previous paragraph.

Results

The results of the experiment described in the previous section are found in table 2,3 and 4 for data set 1,2 and 3 respectively.

Table 2: Simulation results for data set 1 (different mean, equal variance)

# Hidden Units	Determinant	MSE Learn	Correct Learn	MSE Test	Correct Test
1	8.13E-2	0.081	82.50%	0.091	78.85%
2	2.59E-7	0.081	83.00%	0.091	79.10%
3	-7.56E-12	0.080	83.00%	0.091	78.95%
4	1.05E-23	0.080	83.50%	0.092	78.65%
5	6.28E-38	0.080	83.50%	0.092	78.75%
6	3.90E-53	0.080	83.50%	0.092	78.75%
7	-1.79E-68	0.080	83.50%	0.092	78.75%
8	1.34E-84	0.080	83.50%	0.092	78.85%
9	-5.00E-100	0.080	83.50%	0.092	78.80%
10	-2.70E-115	0.080	84.00%	0.092	78.75%

Table 3: Simulation results for data set 2 (equal mean, different variance)

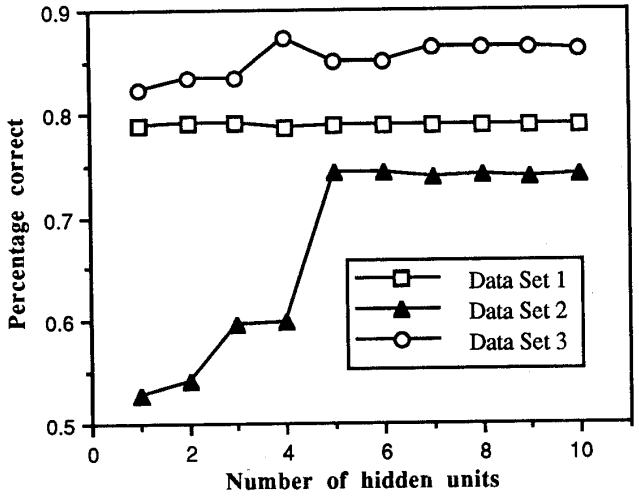
# Hidden Units	Determinant	MSE Learn	Correct Learn	MSE Test	Correct Test
1	3.03E-03	0.158	58.00%	0.161	52.70%
2	3.03E-09	0.157	55.50%	0.160	54.15%
3	1.20E-13	0.142	64.00%	0.170	59.55%
4	-1.64E-22	0.141	63.50%	0.161	59.90%
5	-2.45E-31	0.112	73.50%	0.122	74.25%
6	4.29E-42	0.112	74.50%	0.122	74.15%
7	1.59E-55	0.110	74.50%	0.121	73.70%
8	1.12E-68	0.107	76.50%	0.127	74.00%
9	-4.13E-84	0.107	77.00%	0.126	73.85%
10	-5.39E-99	0.107	77.00%	0.126	73.95%

In graph 1 the network performance (measured on the test set) is plotted versus the number of units in the hidden layer. In graph 2 the logarithm of the absolute value of the determinant of **R** of the output unit is plotted versus the number of units in the hidden layer.
For data set 1 the optimal classifier is a linear function and from this graph 1 it is clear that a network with one hidden unit has a good performance and the optimal network uses two units. From the determinant of the output unit (see graph 2 and table 2) we see that this value decreases rapidly and after 4 units this value decreases with a constant factor of 10^{-15} a 10^{-16}. Since an additional experiment that was performed with single precision floating point numbers showed that the slope of the determinant was of the order of 10^{-7}, we conclude that the

slope of the determinant is based on the machine precision: 53 bits mantissa ≈ 1.1 10^{-16} and 24 bits mantissa ≈ 6.0 10^{-8}. The last result implies the use of Singular Value Decomposition for solving equation 7.

Table 4: Simulation results for data set 3 (different mean, different variance)

# Hidden Units	Determinant	MSE Learn	Correct Learn	MSE Test	Correct Test
1	6.88E-2	0.082	84.50%	0.093	82.15%
2	1.03E-5	0.082	87.00%	0.093	83.40%
3	5.29E-11	0.079	86.00%	0.086	83.30%
4	3.60E-20	0.072	89.00%	0.081	87.20%
5	5.02E-29	0.057	90.00%	0.087	85.05%
6	2.85E-40	0.057	89.50%	0.093	85.05%
7	-2.45E-52	0.050	91.00%	0.133	86.40%
8	1.40E-65	0.050	91.00%	0.133	86.40%
9	-1.88E-81	0.050	91.00%	0.134	86.40%
10	1.80E-97	0.134	86.40%	0.169	86.05%



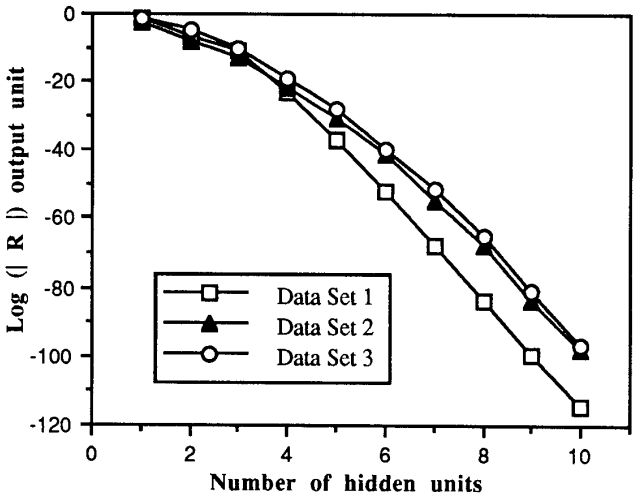
Graph 1: Performance on the test set against number of hidden units (measured on test set).

Data set 2 needs clearly more hidden units to construct the optimal classifier, which is a circle. The optimal classifier is found with 5 hidden units. Although the performance measured on the learning set increases, it deteriorates on the test set when more than 5 unit are used. We see that the determinant of the output unit decreases with a factor 10^{-16} after 7 hidden units. Note that since this is a relatively hard problem to solve, 100 samples per class in the learning set is not much.
On the last data set (3) the optimal network uses 4 hidden units. The determinant of the output unit decreases with machine precision after 8 units. With 4 hidden units the network approximates the optimal linear decision function. The performance of the optimal classifier can not be reached (probably more learning samples are needed).

Discussion

In this paper we described a non iterative method to calculate the weights for a feed forward network with only one hidden layer. The method was applied to three data sets with known statistical properties. The sizes of the used learning sets were realistic for practical problems (100 samples of every class) and the algorithm was able to find a set of weights which approximated the Bayes error.
The performance of the algorithm on data set 1 approximates the Bayes error and the performance on data set 2 is good, compared to the complexity of the function and the number of available learning samples. The performance on data set 3 approaches the ideal linear discrimination function but is not able to go beyond that error. However, data set 3 is not a simple problem due to the large difference in variances of the distributions and we used only a limited number of learning samples.
To find the optimal network size it is best to measure the network performance with an independent data set. However if such data set is not available the determinant of the output (R) covariance can be used to give an idea

of the relative importance of the last added unit. The process of adding units can definitely be stopped if the determinant decreases with a factor which is comparable to the machine precision.
The presented method shows to be capable to find good weights for a network which is used for classification problems. This method does not need long iteration



Graph 2: ^{10}Log of absolute value determinant R of the output unit versus the number of hidden units.

Acknowledgement

This work was sponsored by the Dutch Government as a part of the SPIN/FLAIR-DIAC project and by the Foundation of Computer Science in the Netherlands (SION), with financial support from the Dutch Organization for Scientific Research (NWO).

References

[1] P. Devijver and J. Kittler, "Pattern Recognition: A Statistical Approach", Prentice-Hall International, 1982.
[2] R. Duda and P. Hart, "Pattern Classification and Scene Analysis", John Wiley and Sons 1973.
[3] K. Funahashi, "On the Approximate Realization of Continues Mappings by Neural Networks.", Neural Networks, Vol 2 page 183-192, 1989.
[4] C. Haersma Buma and R. Duin, "Computation of Concave Piecewise Linear Discriminant Functions Using Chebyshev Polynomials", IEEE Transactions on Computers, Vol c-25, No 2, Februari 1976.
[5] K. Hornik, "Multilayer Feedforward Networks are Universal Approximators", Neural Networks, Vol 2 page 359-366, 1989.
[6] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," Neural Networks, Vol. 1, pp. 295-308, 1988.
[7] M. Minski and S. Papert, "Perceptron: An Introduction to Computational Geometry", MIT Ptress, Cambridge M.A. 1969.
[8] N. Nilsson, "Learning Machines", McGraw-Hill System Science Series, 1965
[9] W. Press, B.Flannery, S. Teukolsky and W. Vetterling, "Numerical Recipes in C", Cambridge University Press, 1988.
[10] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagation", Parallel Distributed Processing: Exploring in the Microstructure of Cognition", Vol 1, D.E Rumelhart and J.L. McClelland (Eds.), Cambridge, MA: MIT Press, pp 318-362.
[11] F.M. Silva and L.B. Almeida, "Acceleration Techniques for the Backpropagation Algorithm", Lecture Notes in Computer Science L.B. Almeida and C.J. Wellekens, Neural Networks, Proceedings Eurasip Workshop 1990 , Springer Verlag.
[12] W.S. Stornetta, B.A. Huberman, "An Improved Three Layer Back-Propagation Algorithm", Proceedings IEEE first conference on Neural Networks, San Diego 1987.
[13] B. Widrow and S.D.Stearns, "Adaptive Signal Processing", Prentice-Hall, Englewood Cliffs, New Jersey 1985.