Feed Forward Neural Networks With Random Weights

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

Pattern Recognition Group Delft, Faculty of Applied Physics, Delft University of Technology, P.O. Box 5046, 2600 GA Delft, The Netherlands

Abstract

In the field of neural network research a number of spectacular experiments are described, which seem to be in contradiction with the classical pattern recognition or statistical estimation theory. Systems with very large number of parameters are reported to be successful which contradicts with all experience of history work.

In this paper we try to give some experimental understanding why this could be possible by showing that a large fraction of the parameters (the weights *of the neural network) are of less importance and do not need to be measured with high accuracy. The remaining part is capable to implement the desired classifier and because this is only a small fraction of the total number of weights, the reported experiments seem to be more realistic from a classical point of view.*

Problem description

When analyzing the multi-layer feed forward networks from a statistical or parameter estimation point of view, it is obvious that these systems have a very large number of parameters. However, systems with large numbers of free parameters do not perform very well in parameter estimation problems because the convergence to a proper set of values of the parameters requires a very large number of training examples. In the literature on statistics of parameter estimation it has been shown that the variance in the parameters increases when the model contains more parameters (the Cramér-Rao error bound, e.g. see Mood[6]), which results in poor performance (see Sydenham[9]).

A related phenomenon is the so called *peaking* effect, where the overall performance of a pattern recognition system decreases when the number of features (and the number of parameters) increases. More (relevant) features should imply a better classification performance, but because the parameters of the classifier are determined by a finite (and probably too small) data set, the performance measured on a test set decreases at some point. (See Duin[3] or Vapnik[10])

Despite of the bounds formulated by the statistical laws, a number of reported neural network experiments claim to be successful. A good example is the NETtalk experiment of Sejnowski e.a.[8]. Here 18.629 parameters are estimated with only ≈5000 training examples, with a reported performance of the network classifier of 98%. This seems unrealistic from a classical viewpoint. Similar results can be found in: Angel e.a.[1].

Here we will describe some experimental evidence that indicates why these classification results might be possible. It is shown that there are an enormous amount of possible choices for the parameters which result into a *statistical acceptable* solution of a classification problem. This makes the parameters selection much easier because the learning set is only needed to make a rough selection in the parameter space, the final classifier is less sensitive to the found parameters. This implies that it must be relatively easy for a learning rule like back propagation (see Rumelhart[7]) to find a sufficiently good choice for the the parameters of the neural network.

The approach that we will follow is that the weights of the hidden layer(s) are chosen randomly, whereas the output layer is trained by a single layer learning rule or a pseudo-inverse technique. The experiments show that a near Bayes performance can be reached for a number of non-trivial learning problems. It can be concluded that the parameters found in the hidden layer do not add very much functionality to the classifier. This redundancy of the solutions in parameter space might indicate the nature of some of the successes in the literature.

We would like to emphasize that the method presented in this paper, to train feed forward networks by setting some parameters to random values, is not presented as an alternative learning method. This method is introduced only to analyse the functional behavior of the networks with respect to learning.

0-8186-2915-0/92 $3.00 © 1992 IEEE

Network Architecture

The networks considered here are standard feed forward single hidden layer networks, which means that the processing units are fully connected to units in a previous layer but are not connected to units in the same layer (see figure 1). Only the outputs of the units in a layer are connected to units in a next layer. Therefore there is no feed back in the system. Furthermore only one hidden layer is used. As Funahashi[4] and Hornik[5] have shown this is no fundamental restriction to the approximation capabilities of a network.

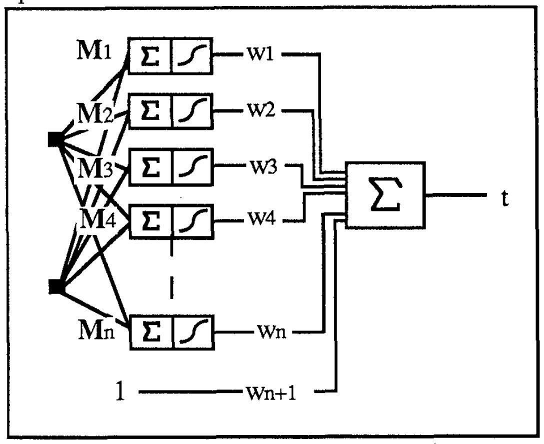

Figure 1 The network feed forward architecture

The units in the hidden layer of our feed forward network, consist of a processing element which calculates 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. The typical squashing function, used in all our experiments, is the sigmoid function:

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

In mathematical formulation the processing element can be described as:

$$O_{\text{hidden}} = F(\sum m_i x_i) = F(\mathbf{M}^T\mathbf{X}) \qquad (2)$$

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

$$\mathbf{M} = [m_1, m_2, ...,m_n, m_{n+1}]^T$$

$$\mathbf{X} = [x_1, x_2,.....,x_n, 1]^T \qquad (3)$$

The output units of our feed forward networks are slightly different from the units found in the hidden layer. These units also compute the weighted sum of the inputs but on this result no squashing function is applied. This is no fundamental restriction on the capabilities of the networks. Since the inverse function of the squashing function exists this only introduces a transformation of the target values by a non linear function. In mathematical formulation the output units perform the following function:

$$O_{\text{output}} = \sum w_i x_i = \mathbf{W}^T\mathbf{X} \qquad (4)$$

Note that weights in the hidden layer are called m_i and weights in the output units are called w_i. When formulas 2 and 4 are combined the total function computed by a network (with one output) can be formulated as:

$$O_{\text{net}}(\mathbf{X}) = \sum_{1}^{\text{hidden}} w_i\, F(\mathbf{M}_i{}^T\mathbf{X}) + w_{\text{hidden+1}} \qquad (5)$$

The Fisher solution

The network weights are optimized by using the well known L_2 criterion:

$$E^2 = \sum_{\text{patterns}} (t - \sum_{1}^{\text{hidden}} w_i\, F(\mathbf{M}_i{}^T\mathbf{X}) + w_{\text{hidden+1}})^2$$

$$= \sum_{\text{patterns}} (t - \mathbf{W}^T\mathbf{O}_{\text{hidden}})^2 \qquad (6)$$

Where t the desired or target value is for a given input **X**, and $\mathbf{O}_{\text{hidden}}$ is defined as:

$$\mathbf{O}_{\text{hidden}} = [F(\mathbf{M}_1{}^T\mathbf{X}), ...F(\mathbf{M}_n{}^T\mathbf{X}), 1]^T \qquad (7)$$

If the weights in the hidden layer $\mathbf{M}_i$ could be determined than $\mathbf{O}_{\text{hidden}}$ is fixed and only the right hand side of equation 6 must be optimized. The parameters **W** (containing all w_i's and $w_{\text{hidden+1}}$) are than optimized for a chosen set of weights $\mathbf{M}_i$. The right hand side of this equation is linear in its parameters which has the great advantage that the solution can be calculated immediately. The weight vector **W*** for which this equation is minimal is calculated as follows:

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

$$\mathbf{W}^* = \mathbf{R}^{-1}\mathbf{P} \qquad (8)$$

* Note that the Mn+1 is always multiplied by one and can be regarded as a threshold value.

$$\mathbf{R} = \sum_{\text{patterns}} O_{\text{hidden}} O^T_{\text{hidden}} \quad (9)$$

$$\mathbf{P} = \sum_{\text{patterns}} t\, O^T_{\text{hidden}} \quad (10)$$

Here **R** and **P** are the input correlation matrix and input-target correlation vector respectively of the output unit. The optimal weight vector **W*** is sometimes called the Fisher 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[11] page 19 or Duda and Hart[2] page 114.)

Experiments and Results

The weights experiments are compared with the standard back propagation method (see Rumelhart[7]) for low dimensional problems (pattern dimension is 2). The simulations are performed with custom written C code on a SUN 4 workstation. Simultaneously the experiments were verified with a Mathematica simulation also running on a SUN 4. There was no significant difference between these two simulations and therefore we only publish the results based on the programs written in 'C', because these are measured on more trials.

Table 1 Statistics of the low dimensional Gaussian data sets.

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)

The random weights experiments

The question which now arises is how to determine the weights $\mathbf{M}_i$. Our experiments indicate that these weights can be taken randomly, for a number of non trivial problems.

Training of the network is performed by the following steps.

1 Set the weights in the hidden units to uniform [-1..1] random values.
2 Determine the weights of all the output units by using the Fisher method

If the application domain is between -1000 and 1000 this method can be slightly optimized by setting $\mathbf{M}_i$ to random value in a more appropriate range, however this is not always needed. Our experiments show that for numerical stability (by solving the Wiener method with a Singular Value Decomposition and checking the range of the eigenvalues) this is sometimes preferred, however it does not improve the method dramatically.

Another possibility is to set the weights $\mathbf{M}_i$ to a random subset of the learning data set. If not enough data samples are present the remaining units are removed from the network. The weights connected to the threshold are set to one. This last method can be interpreted as setting the hidden units to act as 'correlators' for a number of samples in the learning data set.

For these experiments three different data sets are used. The three data sets are all based on the Gaussian probability distribution. These data sets represent two different classes which must be classified. In data set 1 the different classes have equal variance but different mean values. In data set 2 the different classes have equal means but different variances. In data set 3 the different classes have different means and different variances. These statistics are summarized in table 1.

For every data set a learning set of 100 samples per class was generated and a test set with 1000 samples per class was generated. The overlap in the first two data sets is 20% thus the optimal classifier for both data sets have a performance of 80%. The third data set has an overlap of about 5.5% and the optimal *linear* classifier has an error of about 15%.

In all the experiments 8 hidden units are used. This means 3x8=24 parameters in the hidden layer and 9 parameters in the output layer, which makes a total of 33 trainable parameters.

The weights in the hidden units are set to random values, uniformly distributed between -1..1 (called *unif random* in table 2) or they are set to the input values of a randomly selected subset of the learning set (called *data subset* in table 2). Both these methods are simulated and the average performance (measured on a test set) is calculated over 100 trials.

Furthermore these data sets are used to train a network with the standard back propagation rule (see Rumelhart e.a.[7]), where the initial weights are chosen randomly from a uniform distribution between -1..1. The learning coefficient and momentum term are set to 1.0 and 0.9

respectively and the network is trained for 500 epochs. The average performance (determined with an independent test set) is calculated over 100 trials.

Table 2 Simulation results of the low dimensional experiments

Data set	Method	Bayes Perf. (%)	Mean Perf. (%)	Std. dev. (%)
Data set 1	Unif random	80	78.6	0.27
	Data subset	80	78.4	0.25
	Back prop.	80	75.9	4.41
Data set 2	Unif random	80	77.0	1.81
	Data subset	80	76.3	1.79
	Back prop.	80	79.7	4.91
Data set 3	Unif random	94.5	86.6	1.61
	Data subset	94.5	85.3	1.64
	Back prop.	94.5	79.7	7.2

Conclusions

From table 2 it is clear that the random hidden layer procedure generates better networks for data set 1 and 3. Data set 2 is slightly worse, compared to the standard back propagation method. However the standard deviations in the experiments are much smaller for the random method than for the back propagation method.

From these experiments we may conclude that the output layer is significant more important than the weights found in the hidden layer. The 24 parameters in the hidden layer (3x8) are of less importance than the 9 weights found in the output unit. In the 33 dimensional parameter space of the complete network a sub space of 24 dimensions is full of combinations which may be regarded as *statistical acceptable* solutions.

If it is true that these experimental experiences may be expanded to larger networks, for example the Nettalk experiment, we can explain the successes of these experiments as follows. If again the weights found in the output units are of significant more importance, only 2106 (of the total of 18.629) weights need to be measured accurately. The approximately 5000 learning examples are used for this task and although this is still not a very over determined situation, it sounds more reasonable to be successful.

Acknowledgement

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

Furthermore the authors want to thank prof. Sarunas Raudys from the Institute of Mathematics and Informatics in Vilnius Lithuania, for his comments and discussions on this topic.

References

[1] J. Angel, P. Wizinowich, M. Lloyd-Hard and D. Sander, "Adaptive Optics for Array Telescope using Neural Networks Techniques", Letters to Nature, Vol 348 page 221, 15 Nov 1990.

[2] R. Duda and P. Hart, "*Pattern Classification and Scene Analysis*", John Wiley & Sons, 1973.

[3] R.P.W. Duin, "*On the Accuracy of Statistical Pattern Recognizers*", PhD Thesis June 1978, TU-Delft, The Netherlands, Dutch Efficiency Bureau Pijnacker, ISBN 90 6231 052 4.

[4] K. Funahashi, "On the Approximate Realization of Continues Mappings by Neural Networks.", Neural Networks, Vol 2 page 183-192, 1989.

[5] K. Hornik, "Multilayer Feedforward Networks are Universal Approximators", Neural Networks, Vol 2 page 359-366, 1989.

[6] A.M. Mood, F.A. Graybill and D.C. Boes, "*Introduction to the Theory of Statistics*", third edition, McGraw-Hill, 1974.

[7] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagation", in "*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.

[8] T.J. Sejnowski and C.R. Rosenberg, "NETtalk: A parallel network that learns to read aloud", The Johns Hopkins University Electrical Engineering and Computer Science Technical Report JHU/EECS-86/01, 32 pp.

[9] P. Sydenham, "*Handbook of measurement Science*", Vol 1, Chapter 8, John-Wiley & Sons, 1982.

[10] V.N. Vapnik, "Estimation of Dependences Based on Empirical Data", Springer Verlag 1982.

[11] B. Widrow and S.D.Stearns, "*Adaptive Signal Processing*", Prentice-Hall, Englewood Cliffs, New Jersey 1985.