ESANN'2005 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 27-29 April 2005, d-side publi., ISBN 2-930307-05-6.

Initialisation improvement in engineering feedforward ANN models.

A. Krimpenis and G.-C. Vosniakos

National Technical University of Athens, School of Mechanical Engineering, Manufacturing Technology Division, Heroon Polytehneiou 9, 157 80 Zografou, Athens, Greece

Abstract
Any feedforward artificial neural network (ANN) training procedure begins with the initialisation of the connection weights' values. These initial values are generally selected in a random or quasi-random way in order to increase training speed. Nevertheless, it is common practice to initialize the same ANN architecture in a repetitive way in order for satisfactory training results to be achieved. This is due to the fact that the error function may have many local extrema and the training algorithm can get trapped in any one of them depending on its starting point based on the particular initialisation of weights. This paper proposes a systematic way for weight initialisation that is based on performing multiple linear regression on the training data. Experimental data from a metal cutting process were used for ANN model building to demonstrate an improvement on both training speed and achieved training error regardless of the selected architecture.
Keywords: Feedforward ANNs, initialisation, engineering data, multiple linear regression

1. Introduction

The connection weights between neurons are where the ANN stores information to describe the problem at hand and to quantify interdependencies between its inputs and outputs. The goal of the training procedure is to calculate the values of the weights that correspond to the minimum value of the error function. In the case of feedforward ANNs, this function is usually a sum of squares of the differences between the actual data and those that are calculated by the ANN, the weights being the unknown parameters. The initial values of weights determine the starting point of the training algorithm and directly affect training speed and training error. If these initial values are not close to the global minimum or are close to an area with many local minima of the error function, the training algorithm may be trapped in one of them and therefore training will be slow and/or produce bad results [1]. To avoid such problems, the initialisation is done in a random or quasi-random way. This, in turn, results in the need to train the same ANN multiple times, effectively using different initialisations, to ensure that the performance of the ANN model is primarily dependent on its architecture and not on the initial values of the weights. Consequently, the practitioner is involved in a repetitive process that requires more development time as well as experience and intuition.

Different initialisation methods have been proposed to deal with this problem. Nguyen and Widrow [2], propose an initialisation in the interval [-0.5,0.5] using a

uniform distribution. In this way, the active regions of the layer's neurons will be distributed approximately evenly over the input space. The advantages of this method compared to purely random initialisations are that few neurons are wasted and training works faster since each area of the input space has associated neurons. Yam and Chow [3] minimize the norm $\left\| A^l \cdot W^l - S^l \right\|_2$, where l = 1, 2, …, L-1 (L being the number of layers of the ANN), A^l are the inputs to the l-th layer, W^l are the weight values and S^l are the inverse of the activation function. When applied to function approximation the results correspond to a 46.1% in the number of required epochs. An extension of this method is described in [4], where the Cauchy inequality is introduced and two new methods using uniform and normal distribution initialisation respectively are proposed. Francois uses orthogonal arrays in [5] to linearly correlate the input and hidden neurons in order to improve the generalization ability of the ANN. A partial least-squares (PLS) algorithm is used in [6] together with the back-propagation algorithm to calculate both the initial weight values and the optimal number of hidden neurons. The PLS structure is viewed as a simplified 3-layered ANN and its basic function is to reduce the number of input variables. A much different approach is adopted in [7] for function approximation. Since the function is known, its local extrema can be calculated and then these results can be used to initialize the weights. In this way, very fast training is achieved even in the case of multivariate functions. Pre-processing of data has received a lot of attention by researchers and two very thorough investigations concerning the different methods used and considerations that must be made are presented in [8] and [9]. Another very interesting work is described in [10] where k-nearest neighbour filtering is employed to remove noise from the training data. Ivanova and Kubat [11] employ decision-tree generators to initialize and train ANNs. After constructing decision-trees from the training examples, they transform the rules using the neurons as logical operators and set the initial weights so that the ANN approximates the decision-tree classifications.

In all of the described methods, the initialisation of the weights is based on mathematical criteria and analytical equations. It is clear that most of these approaches are not generic, but rather strongly case-, or even, data-dependent and this is why they have been applied in focused problems such as function approximation. On the other hand, in the majority of engineering applications, the correlations between the different parameters are unknown and there is no analytical description due to the complex nature of the underlying phenomena. Therefore, an initialisation approach that combines a data-dependent model with a random initialisation scheme is being presented in this paper.

2. The approach

The initialisation method has been developed based on the following simplifications: it only applies to feedforward ANNs, with one hidden layer of neurons and a single neuron in the output layer. The activation function of the output layer is the identity function. These assumptions do not limit the generality of the proposed method because on one hand these are also valid for the majority of the ANN models that are usually developed for engineering applications and on the other hand, the method can

be easily extended for more than one hidden layers. An ANN that fulfils these assumptions is given in Figure 1, with n input and m hidden neurons. The mathematical notation used is as follows:

x_i: activation of the i-ith input neuron (i = 1, 2, …, n),
k_j: activation of the j-ith hdden neuron (j = 1, 2, …, m),
y: activation of the output neuron (i.e., the response of the ANN)
$IW_{j,i}$: weight between the i-ith input neuron and the j-th hidden neuron
b_j: bias of the j-th hidden layer
$LW_{1,j}$: weight between the j-ith hidden neuron and the output neuron
b_y: bias of the output neuron
tansig(x): hyperbolic tangent function

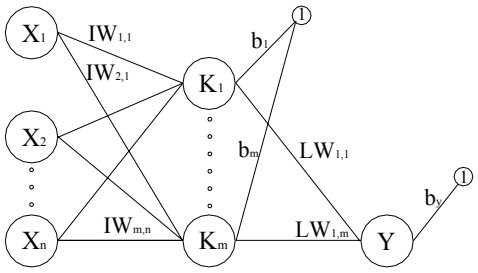

Figure 1. Feedforward ANN.

The ANN's response is given by:

$$y = \sum_{j=1}^{m} LW_{1,j} \cdot k_j + b_y \tag{1}$$

where

$$k_j = \tan sig\left(\sum_{i=1}^{i=n} IW_{j,i} \cdot x_i + b_j\right) \tag{2}$$

During training, the only known magnitudes in equations (1) and (2) are y and x_i, $IW_{j,i}$, b_{kj}, $LW_{1,j}$ and b_y are initialized and then their values are determined by the training algorithm used. These equations analytically correlate the input with the output parameters of the ANN, which in this case is nothing more than a complex non-linear model.

If a multiple linear regression on the training data were conducted, the result would be:

$$y = a_o + a_1 \cdot x_1 + a_2 \cdot x_2 + \ldots + a_n \cdot x_n = \sum_{i=1}^{n} a_i \cdot x_i + a_o \tag{3}$$

where y is the dependent variable (output parameter) and x_i are the independent variables (input parameters) and the coefficients α_i and α_o are all known.

By comparing equations (2) and (3) it is easily concluded that:

$$b_j = a_o, \; IW_{j,i} = a_i$$

In this way, the results for the coefficients of the multiple linear regression model become the initial values of the first layer of weights and of the biases of the

hidden layer of neurons. As for the second layer of weights, the initial values can still be obtained using a random or semi-random scheme allowing for a better search of the solution space in case of consecutive trainings of the same ANN architecture. Note that, if the data is split into training and testing subsets, the multiple linear regression should be performed using the training subset alone.

3. Results and Discussion

In order for the proposed method to be tested, a comparison was made to the Nguyen – Widrow (N-W) initialisation method, in terms of training speed and achieved training error, using experimental data. The Nguyen-Widrow method is generally superior to purely random initialisations, as discussed earlier, and this is why it has been selected for this comparison. Furthermore, this is the default initialisation method for the feedforward ANNs that are created through the ANN toolbox of the MATLAB software package, which was used to train the ANN models. The multiple linear regression (MLP) initialisation method was also implemented using MATLAB programming.

The experimental data derived from a turning cutting process of a metal bar with the input parameters being the depth of cut (mm), feed (mm/rev), spindle speed (RPM), the ratio of the workpiece length to its diameter (L/D) and the ratio of the distance of the cutting point to the workpiece's length (L_i/L). The output parameter was the deviation of the actual from the desired depth of cut. A total of 40 different cuts were made.

Three different architectures were used, using one hidden layer with 3, 6 and 10 neurons respectively, in order to evaluate the method's performance for different network sizes. For each architecture, there were 10 training procedures using the N-W method and 10 training procedures using the proposed method. For each training, the history of the mean squared error (MSE) in relation to the number of epochs was recorded. The detailed results are given in Tables 1, 2 and 3, for each architecture respectively.

Using the smaller network, one can see that the two methods are almost equivalent in terms of training speed and performance. Although different limits for the maximum number of epochs were used, the training error was practically constant after the first 3000 epochs for all trainings. Regardless of the initialisation method, the achieved training error is not very good due to the low number of neurons in the hidden layer.

By doubling the number of neurons in the hidden layer, the results are very different. Training stops in much fewer epochs and the training error is considered very good. By observing the N-W results, especially for training runs 3, 6 and 10, it may be concluded that the training algorithm was trapped in a local minimum. Instead, using the proposed method training is completed in almost any case comparatively faster and there does not seem to be any indication of convergence to local minima.

Increasing the number of hidden neurons even more allows the N-W method to avoid local minima. However, the comparison of the two methods indicates that the proposed method is superior and produces more consistent initialisations.

Training no.	Epochs	MSE Training Error	Initialization method	Epochs	MSE Training Error	Initialization method
1	5000	1,26E-06	N-W	6393	1,03E-06	MLP
2	5000	1,60E-06	N-W	6576	1,03E-06	MLP
3	1675	2,69E-06	N-W	6510	1,03E-06	MLP
4	5000	6,90E-07	N-W	10000	6,94E-07	MLP
5	5000	1,06E-06	N-W	10000	8,28E-07	MLP
6	5000	8,38E-07	N-W	6752	1,03E-06	MLP
7	612	8,10E-07	N-W	6990	1,03E-06	MLP
8	5000	7,69E-07	N-W	10000	8,28E-07	MLP
9	5000	9,13E-07	N-W	10000	8,28E-07	MLP
10	5000	1,60E-06	N-W	7030	1,03E-06	MLP

Table 1. Training results using the two different initialisation methods for architecture 5x3x1.

Training no.	Epochs	MSE Training Error	Initialization method	Epochs	MSE Training Error	Initialization method
1	2325	1,15E-29	N-W	2351	4,97E-27	MLP
2	3441	2,07E-28	N-W	1473	1,25E-30	MLP
3	10000	2,45E-07	N-W	2294	2,81E-29	MLP
4	9377	1,95E-26	N-W	1666	3,74E-30	MLP
5	1397	3,79E-24	N-W	957	3,73E-29	MLP
6	10000	3,91E-08	N-W	2147	1,09E-30	MLP
7	2565	1,37E-26	N-W	926	6,32E-28	MLP
8	2612	1,92E-28	N-W	996	7,54E-31	MLP
9	1701	6,85E-28	N-W	1926	2,50E-30	MLP
10	10000	3,23E-07	N-W	1019	8,82E-29	MLP

Table 2. Training results using the two different initialisation methods for architecture 5x6x1.

Training no.	Epochs	MSE Training Error	Initialization method	Epochs	MSE Training Error	Initialization method
1	1466	1,43E-25	N-W	915	5,61E-31	MLP
2	734	1,54E-28	N-W	686	1,72E-31	MLP
3	670	1,78E-29	N-W	750	1,67E-28	MLP
4	618	2,24E-25	N-W	1328	8,47E-26	MLP
5	1753	8,30E-26	N-W	1004	5,54E-31	MLP
6	765	7,63E-28	N-W	985	1,86E-26	MLP
7	983	5,90E-24	N-W	1032	2,12E-26	MLP
8	2155	1,11E-27	N-W	899	2,52E-28	MLP
9	256	1,27E-31	N-W	903	1,04E-28	MLP
10	1139	4,74E-24	N-W	1860	3,71E-31	MLP

Table 3. Training results using the two different initialisation methods for architecture 5x10x1.

4. Conclusions

Feedforward ANNs have been widely used to model the complex interdependencies and phenomena that appear in engineering applications. In order to facilitate the ANN model development procedure a new method for the initialisation of the weight values has been proposed. This method involves an initial estimate for the weight values that is based on a first order approximation (multiple linear regression) of the training data. Based on the results, it is clear that there is an improvement for both the number

of required epochs and the achieved training error. Thus, the proposed method results in faster and more accurate training of the ANN model. It must also be noted that the improvement is proportional to the size of the network, i.e. for larger networks the improvement is also larger, which is very desirable since these networks usually take more time to train.

Acknowledgements

This work was partly funded by the PENED01 program (Measure 8.3 of the Operational Program Competitiveness, of which 75% is European Commission and 25% national funding). It was also partly funded by the Basic Research program of the National Technical University of Athens Thales 2001.

References

[1] D. Partridge. Network generalization differences quantified. *Neural Networks*, 9(2):263-271, 1996.

[2] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *Proceedings of the International Joint Conference on Neural Networks*, 3:21-26, 1990.

[3] Y.F. Yam, T.W.S. Chow and C.T. Leung. A new method in determining initial weights of feedforward neural networks for training enhancement. *Neurocomputing*, 16:23-32, 1997.

[4] Y.F. Yam and T.C. Chow. A weight initialisation method for improving training speed in feedforward neural networks. *Neurocomputing*, 30:219-232, 2000.

[5] B. Francois. Orthogonal considerations in the design of neural networks for function approximation. *Mathematics and Computers in Simulation*, 41:95-108, 1996.

[6] T.-C.R. Hsiao, C.-W. Lin and H.K. Chiang. Partial least-squares algorithm for weights initialisation of backpropagation network. *Neurocomputing*, 50:237-247, 2003.

[7] X.M. Zhang, Y.Q. Chen, N. Ansari and Y.Q. Shi. Mini-max initialisation for function approximation. *Neurocomputing*, 57:389-409, 2004.

[8] A.C. Tsoi and A. Back. Static and dynamic preprocessing methods in neural networks. *Engineering Applications of Artificial Intelligence*, 8(6):633-642, 1995.

[9] W.S. Sarle. Neural Network FAQ, part 2 of 7: Learning. Periodic posting to the Usenet newsgroup comp.ai.neural-nets, 1997, URL: ftp://ftp.sas.com/pub/neural/FAQ.html

[10] P.L. Rosin and F. Fierens. The effects of data filtering on neural network learning. *Neurocomputing*, 20:155-162, 1998.

[11] I. Ivanova and M. Kubat. Initialisation of neural networks by means of decision trees. *Knowledge-Based Systems*, 8(6):333-344, 1995.