ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 19-24

Regulated Activation Weights Neural Network (RAWN)

H.A.B. te Braake[1], H.J.L. van Can[2], G. van Straten[3], H.B. Verbruggen
Control Laboratory, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands

***Abstract**: This paper considers the training of a feedforward neural network with one hidden layer. The proposed method splits the training problem into two separate parameter estimation problems. Each subproblem can be solved with standard least squares techniques. The training therefore is very fast compared to iterative training schemes. An examples is presented to show some of the properties or particularities of this algorithm.*

1 Introduction

Feedforward Neural Networks with a single hidden layer of neurons and a linear output layer are a convenient way to model a nonlinear input-output mapping. Commonly these networks are trained with backpropagation. Despite many improvements which were developed recently, such as acceleration, adaptive learning rate, momentum, special initialization, backpropagation is slow and has poor convergence properties. Several papers have addressed the training of static feedforward neural nets as a parameter estimation problem ([4]), ([3]). Also Newton and Quasi-Newton (e.g. Levenberg-Marquardt) methods have been developed for training. Significant speed and convergence improvements have been reported. However, in all these procedures, given off-line batch data many iterations through the data set are needed to obtain acceptable residual errors for each selected configuration (number of neurons and input data). This paper describes a fast way to obtain the weights in feedforward neural networks.

The basic idea of the proposed method is to split the training problem into two nearly independent subproblems. Both subproblems can be solved by using standard least squares methods, which leads to a fast computation of the weights. Various configurations can be tested in a relatively short time. So that afterwards a reasonable choice for a network configuration can be made.

In section 2 the basic structure of a feedforward neural network will be described. Section 3 is devoted to the calculation of the output weights and in section 4 the calculation of the so-called activation weights will be presented. In section 5 two examples will be worked out to demonstrate some properties of the training method. Finally some conclusions will be drawn.

2 Feedforward Neural Networks

To construct a feedforward neural network with one hidden layer, the input vector $x(k)$

1) email: h.a.b.tebraake@et.tudelft.nl
2) Kluyver Laboratory for Biotechnology, Delft University of Technology
3) Systems and Control Group, Wageningen Agricultural University, The Netherlands

and the bias vector can be grouped into a matrix, with one row for each event and one column for each input (including a column with ones for the biases) $\mathbf{X}(k) = (x_1(k) \,..\, x_i(k) \,..\, x_{N_i}(k)\ 1)$, with matrix $\mathbf{X} = (\mathbf{X}(1) \,..\, \mathbf{X}(k) \,..\, \mathbf{X}(N_e))^T$. Similarly, the output vectors can be grouped into a matrix $\mathbf{Y}(k) = (y_1(k) \,..\, y_l(k) \,..\, y_{N_o}(k))$, with $\mathbf{Y} = (\mathbf{Y}(1) \,..\, \mathbf{Y}(k) \,..\, \mathbf{Y}(N_e))^T$. Then the neural net can be expressed concisely by:

$$\begin{aligned} \mathbf{Z} &= \mathbf{X} \cdot \mathbf{W}^h \qquad [N_e \times N_h] \\ \mathbf{V} &= f(\mathbf{Z}) \qquad [N_e \times N_h] \\ \mathbf{Y} &= \mathbf{V}_b \cdot \mathbf{W}^o \qquad [N_e \times N_o] \end{aligned} \tag{1}$$

With input $\mathbf{X}$ ($\mathbf{X} \in X \subseteq \mathbb{R}^{N_e \times [N_i + 1]}$) and output $\mathbf{Y}$ ($\mathbf{Y} \in Y \subseteq \mathbb{R}^{N_e \times N_o}$). The function $f(.)$ is the activation function which, for this paper, is the *tanh*-function. This function does have a asymptote at -1 and one at 1. Matrices $\mathbf{Z}$ and $\mathbf{V}$ contain intermediate results. Matrix $\mathbf{V}_b \in V^{N_e \times (N_h + 1)}$ is equal to $\mathbf{V} \in V^{N_e \times N_h}$ except that one column with ones is added to express the output bias b^o. The set V is a real compact set with values between -1 and 1. The activation weights are grouped together in a matrix $\mathbf{W}^h \in \mathbb{R}^{[N_i + 1] \times N_h}$, and the output weights in a matrix $\mathbf{W}^o \in \mathbb{R}^{[N_h + 1] \times N_o}$.

3 Estimation of the Output Weights

The elements of the weight matrices $\mathbf{W}^h$ and $\mathbf{W}^o$ are parameters that must be found experimentally in order to obtain an acceptable fit to the available data. This process is called training, which is nothing else then finding the global or local minimum of the following criterion function:

$$J(\mathbf{W}^h, \mathbf{W}^o, \mathbf{X}, \mathbf{Y}) = \|\boldsymbol{e}\| \tag{2}$$

where $\boldsymbol{e}$ is a vector with modeling errors and $\|\,.\,\|$ is a suitable vector norm. Usually this function is minimized by means of gradient based iterative search algorithms, like back-propagation, Levenberg-Marquardt or SQP (Sequentially Quadratic Programming). To obtain a non-iterative training scheme, the proposal is to split the training problem into two subproblems which each can be solved separately. Of course this does not always imply that the overall problem can be solved optimally, but application of the proposed method to real-life processes, showed that satisfactory results can be achieved.

The first subproblem is the estimation of $\mathbf{W}^h$ and the second subproblem is the estimation of $\mathbf{W}^o$. The method to obtain $\mathbf{W}^h$ will be described later on. The estimation of $\mathbf{W}^o$ will be worked out in this section. To obtain $\mathbf{W}^o$, first assume that the weights $\mathbf{W}^h$ are already known, and therefore $\mathbf{V}_b$ is known, too. Suppose that the true output can be modeled by:

$$\mathbf{Y} = \mathbf{V}_b \mathbf{W}^o + \boldsymbol{e} \tag{3}$$

The vector $\boldsymbol{e}$ denotes the modeling error. A parameter estimation problem remains which is linear in the parameters. By minimizing the sum of squared modeling errors the well known least squares estimation of $\mathbf{W}^o$ then becomes:

$$\hat{\mathbf{W}}^o = [\mathbf{V}_b^T \mathbf{V}_b]^{-1} \mathbf{V}_b^T \mathbf{Y} \tag{4}$$

The 'hat' denotes the fact that the related variable is the estimation of that variable. The

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 19-24

matrix $\mathbf{V}_b^T\mathbf{V}_b$ must be nonsingular otherwise $[\mathbf{V}_b^T\mathbf{V}_b]^{-1}$ would not exist. This implies that $\mathbf{V}_b$ must have rank N_h+1.

4 Estimation of the Activation Weights

In the previous section it was demonstrated that by splitting up the training into two basic subproblems the elements of the matrix $\mathbf{W}^o$ can be calculated with eq. (4). Before this can be done, the matrix $\mathbf{W}^h$ must be obtained first. Te Braake and Van Straten ([1]) have shown that by taking the weights randomly good training results can be obtained. In this paper an extension to this approach will be described. Here a piecewise linear approximation will be used to choose the activation weights. In either method precautions on the choice of $\mathbf{W}^h$ must be made, because the matrix $\mathbf{Z}$ in eq. (1) must satisfy general conditions in order to guarantee that the least squares output weights estimation problem (the second subproblem) can be solved (see eq. (4)).

The basic idea is that the input data multiplied with the activation weights, i.e. matrix $\mathbf{Z}$ from eq. (1), must be correctly spread out over the relevant area of the nonlinear activation function. One can easily demonstrate that if $-c \leq z(k) \leq c$, where c is the point were $|df/dz| = \varepsilon$ (ε is small, real and nonzero scalar), and $z(k)$ is one element of $\mathbf{Z}$, $\tanh(z(k))$ is non-saturated. If $|z(k)| >> c$ for all j then $z(k)$ is in the saturated part of the activation function and then $f(|z(k)|) \approx f(|z(k) + \delta|) \approx 1$, with δ a small real number. The training therefore probably will fail to find a correct mapping between input and output. If $|z(k)| << c$ then the mapping is concentrated around $-\varepsilon < z(k) < \varepsilon$, with ε a small real number. Then $f(z(k))$ becomes an almost linear function of $z(k)$ which leads to an ill-conditioned matrix inversion $[\mathbf{V}_b^T\mathbf{V}_b]^{-1}$.

If $\mathbf{W}^h$ is of full rank, then the matrix product $\mathbf{X}\mathbf{W}^h$ causes that the rank of $\mathbf{Z}$ (see eq.(1)) is equal to the rank of $\mathbf{X}$. If the function $f(.)$ (as used in eq. (1)) is a linear function or a power function (e.g. $f(.)=z^a$), the columns of $\mathbf{V}_b$ still will not be of full rank. However, if the function $f(.)$ is a kind of exponential function (for example $f(.)=e^z$) $\mathbf{V}$ in general will have full rank (see [2]).

The estimation $\mathbf{W}^h$ is based on the assumption that a nonlinear function can be approximated by a sequence of linear functions based on a linear least squares estimation of the parameters of the linear parts. The parameters in these equations then can be used as the elements of $\mathbf{W}^h$. Each neuron is assigned to a specific subset of the complete data set and the network is built by sequentially assigning a neuron to each subset. The neuron number is the same as the number of the subset under consideration. For the partitioning of the data set several techniques can be used. In [2] a description of these techniques is given and also the details of the method are presented in that reference.

Suppose a feedforward neural network has to fit a certain (static) function $y = f(x)$, $x \in X \subseteq \mathbb{R}^{N_e \times N_i}$ and $y \in Y \subseteq \mathbb{R}^{N_e \times 1}$. Then, a data set Φ can be created with 'measured' input/output pairs, or $\Phi = [\mathbf{X}\ \mathbf{Y}]$. By partitioning the matrix Φ into N_h subsets, it is possible to construct a linear model for each subset Φ^n, with $\Phi^n \subset \Phi$ and $n \in \{1 \ldots N_h\} \subseteq \mathbb{N}$. The linear model is then given by:

$$\tilde{\mathbf{Y}}^n = \mathbf{X}^n\mathbf{W}_n^h + b_n^h \qquad (5)$$

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 19-24

The weights $\mathbf{W}_n^h$ ($\mathbf{W}_n^h \in W_n \subset \mathbb{R}^{N_i \times 1}$) and b_n^h can be calculated with OLS (ordinary least squares) resulting in:

$$\left[\mathbf{W}_n^h \; b_n^h\right]^T = (\tilde{\mathbf{X}}^{nT}\tilde{\mathbf{X}}^n)^{-1}\tilde{\mathbf{X}}^n\tilde{\mathbf{Y}}^n \tag{6}$$

In this equation $\tilde{\mathbf{X}}^n$ is the matrix $\mathbf{X}^n$ supplied with an extra column filled with ones, needed to express the biases b_n^h. $\tilde{\mathbf{Y}}^n$ is a scaled matrix $\mathbf{Y}^n$ with the following property: $\|\tilde{\mathbf{Y}}^n\|_\infty < c$, with c the value where $|df/dz| = \varepsilon$, (f is the activation function and ε is a small positive real scalar).

Every linear function describes the mapping between input and output for that particular subset n. Finally, after calculating N_h linear models, the matrix $\mathbf{W}^h$ is build as follows:

$$\mathbf{W}^h = \begin{bmatrix} \mathbf{W}_1^h & \mathbf{W}_2^h & \dots & \mathbf{W}_n^h & \dots & \mathbf{W}_{N_h}^h \\ b_1^h & b_2^h & \dots & b_n^h & \dots & b_{N_h}^h \end{bmatrix} \in \mathbb{R}^{(N_i+1)\times N_h} \tag{7}$$

If, after finishing the training phase, the trained neural network is used, then each submodel will be extrapolated out of the particular working range. The estimation of the output weights, with eq. (4), then fits the particular nonlinear submodels to the data.

5 Example

In this section an example of a static mapping will be given to show some particularities of the proposed method. To compare the results of the RAWN approach, the iterative training is performed with the Levenberg-Marquart algorithm (L-M). The gradients, needed to calculate the search direction, are calculated analytically. The Hessian is updated with the BFGS formula [3]. This training method is used instead of the backpropagation algorithm because it is faster, more stable, more robust and it gives better results. In Fig. 1 the result of the training with L-M is given. This result was not directly available.

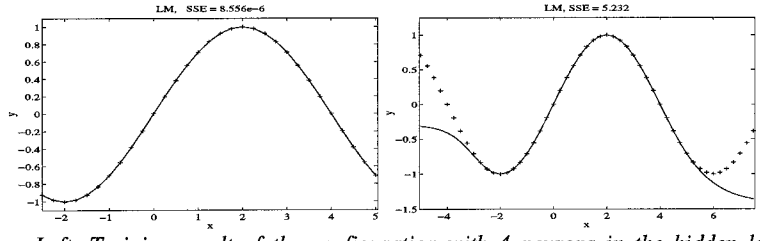

Fig. 1 Left: Training result of the configuration with 4 neurons in the hidden layer and trained with a combination of Backpropagation and Levenberg-Marquardt. '-'= network output, '+' = data. SSE=8.556e-6. Right: Result on the test set (extrapolation) SSE = 5.232

Several training sessions with different initial weights were necessary to obtain an acceptable fit. The result shown in Fig. 1 is obtained after 'initialization' of the weights by a few backpropagation runs. Note that only 32 points are used to train the network. Not much computation time is necessary for training. If more data was used then the time needed to train the network would increase rapidly with increasing size of the training set. The network shows a good performance on the training set. The result on a test set is shown in the same figure. It is clear that the network is only to a small extend able to extrapolate the

results.

The RAWN training procedure, is also applied to this example. The overall training result

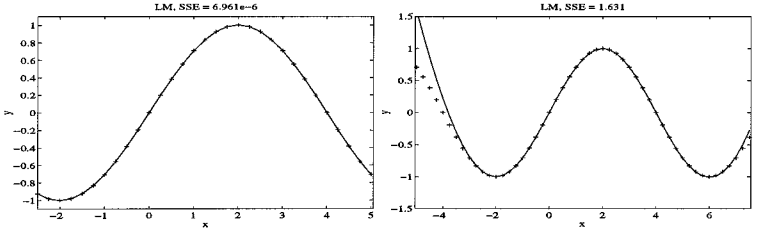

Fig. 2 *Left: Training with RAWN network with 4 neurons, trained Fuzzy Clustering and optimized scaling, SSE = 6.9e-6. Right: Extrapolation SSE = 1.631.*

is displayed in Fig. 2. The training result is slightly better than the result obtained with the L-M algorithm, the result on the test set shows the capability of the network to extrapolate (to some extent). In case of the sine example, it was found that using different data sets and different number of neurons, the extrapolation was always better then in case of the network trained with L-M. This is probably due to the linear submodels, causing the model to find the correct direction at the borders of the data set.

To compare the training methods in case of potential over fitting problems, both training procedures will now be tested on the same example but now zero mean white noise with a standard deviation of 0.1 is added to the output *y*. On the left hand side of the next figure the training result of the L-M training is showed and on the right side this training result is compared with the undisturbed output. The amount of training iterations is the same as in

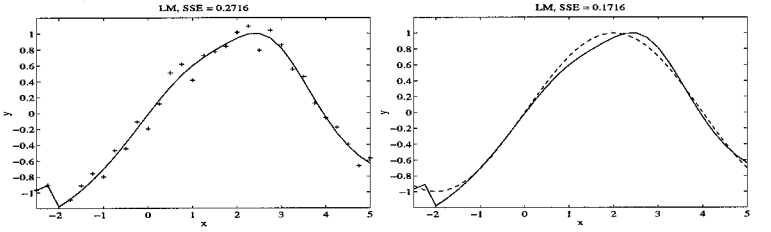

Fig. 3 *Left: L-M Training result of neural network with 4 hidden neurons. '+'= real, with noise disturbed, data, '-' = neural network output.SSE = 0.27. Right: Training result compared with the real undisturbed output. SSE = 0.17*

the undisturbed case. In Fig. 4 the same data set is applied to a network trained with the RAWN method. From this example it can be observed that the iterative L-M method suffers from over fitting . The RAWN training method does not have this drawback. Given the fact that both the activation weights and the output weights are calculated with an optimal linear estimator results in less problems with over fitting, i.e. the training algorithm works like a noise filter. However, note that if more neurons are used the over fitting problem also would occur. The computation time to estimate the weights in case of the RAWN method was less then training with L-M. If bigger data sets and more complex neural networks are used then the difference in necessary training time increases considerably.

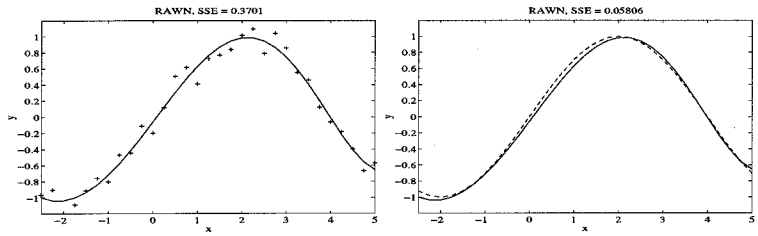

Fig. 4 *Left: RAWN Training result of neural network with 4 hidden neurons. '+'= real, with noise disturbed, data, '-' = neural network output. SSE = 0.37. Right: Training result compared with the real undisturbed output. SSE = 0.06.*

6 Conclusions and Discussion

Good results can be obtained with the RAWN-method. One reason for this is the fact that the problem is split in two subproblems and both subproblems are solved separately and optimally. The estimation of $\mathbf{W}^h$ is optimal in the sense that the obtained weights are an optimal solution to the stated criterion, which is the minimization of the modeling error of the submodels. This is also the case for the estimation of $\mathbf{W}^o$. So one can expect that the complete training problem is solved suboptimal.

It is clear that the RAWN is very fast compared to backpropagation and L-M. It only needs seconds to calculate the estimates for $\mathbf{W}^h$ and $\mathbf{W}^o$. The described method is very suitable and flexible for the identification of nonlinear processes with neural networks. The importance of the configuration and the training signal requests that a lot of configurations must be trained to see which one is the best. Thus fast training is required.

Both training methods, L-M and RAWN, can be compared to each other with various measures, e.g. calculation time for training, accuracy of the training, amount of necessary a-priori knowledge, flexibility and the sum of squared errors of the test set. Although it is impossible to extend the results of the example to a general conclusion about both training methods some remarks can be made. Application of the method to various real-life modeling problems did show that the method is worthwhile to consider as a interesting training algorithm for neural nets with one hidden layer. Moreover, although it was not shown, the proposed method is also applicable to model dynamic systems.

References

1 H.A.B. te Braake, and G. van Straten, (1995). Random Activation Weight Neural Net (RAWN) for fast non-iterative training. *Engineering Applications of Artificial Intelligence, 8, 71-80.*

2 H.A.B. te Braake. (1995). Two Step Approach in Training of Regulated Activation Weights Neural Networks (RAWN). *Internal Report R95.043. Control Laboratory, Delft University of Technology. October 1995.*

3 H. Demuth and M. Beale. **Neural Network Toolbox for use with Matlab**. The Mathworks, January 1994.

4 R.S. Scalero, and N. Tependelenlioglu, (1992). A fast new algorithm for training feedforward neural networks. *IEEE Transactions on Signal Processing, **40**, 202-210.*

5 S. Singhal, and L. Wu, (1989). Training feedforward networks with the extended Kalman algorithm. *IEEE Proceedings, 1187-1190.*