ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 51-56

Cascade Learning for FIR–TDNNs

M. Diepenhorst, J.A.G. Nijhuis and L. Spaanenburg
Rijksuniversiteit Groningen, Dept. of Computing Science,
P.O. Box 800, 9700 AV Groningen (The Netherlands)

Abstract. Time–Delay neural networks are well–suited for prediction purposes. A particular implementation is the Finite Impulse Response (FIR) neural net. As illustrated by some applications, a major design problem exists in establishing the optimal order of such filters. Here, a constructive solution based on Cascade Learning is outlined.

I. Introduction

The handling of temporal characteristics in the training of neural nets has been researched in the past for its biological plausibility and for its predictive potential. In a standard Time–Delay Neural Network (TDNN), as introduced by Lang and Hinton (1988) and Waibel et al. [1], temporal characteristics are imbedded in a feedforward ANN through synaptic delays. In its most simple format, a memory–less feedforward network with a tapped delayline on its inputs constitutes already a TDNN. A more complicated scheme arises when the delayline is repeated on the inputs of every neuron. By structural transformation, any delayline can be replaced by a fanout–tree of parallel delays.

So typically, two neurons in a TDNN will be interconnected by N synapses, each having a different delay. If these delays can be expressed in discrete time as multiples of a unit time–delay, the set of N synapses between two neurons can be replaced by a adaptable synaptic Finite Impulse Response (FIR) filter [2]. In the synaptic filter, the delays are placed in the synapses; hence the filter coefficients are also the weights. We study here the TDNN filter, where the delays are placed within the neuron to separate the learning of the weights from that of the filter characteristics and order.

In the following, we will first review the determination of the weight values from training. Then we show how the filter characteristics can be determined as well and relate some practical experience. Finally, we give a constructive procedure to train for the order of the filters.

II. The synaptic filter

The output $y(n)$ of an N'th order adaptable FIR filter in response to an input $x(n)$ is described by: $y(n) = \sum_{i=0}^{N} h_i(n)x(n-i)$

Here, $h_i(n)$ denotes the i'th coefficient of the filter at time n. If $\mathfrak{I}_j$ denotes the set of neurons that are connected to neuron j by synaptic FIR filters, the output of this neuron in discrete time is described by:

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 51-56

$$x_j(n) = f\Big(-\theta_j(n) + \sum_{k\in\mathfrak{I}_j}\Big[\sum_{i=0}^{\text{orde filter k}} h_{jki}(n)x_k(n-i)\Big]\Big) \qquad (1)$$

If all synaptic filters $H_{jk}(n)$ in (1) may be described as the product of a basic filter H_j and a factor $v_{jk}(n)$, i.e. $h_{jki}(n) = v_{jk}(n) \cdot h_{ji}$ (2)
all filters will be of the same order and (1) can be rewritten to:

$$x_j(n) = f\Big(-\theta_j(n) + \sum_{k\in\mathfrak{I}_j}\Big[\sum_{i=0}^{N} h_{ji}v_{jk}(n)x_k(n-i)\Big]\Big)$$

$$= f\Big(-\theta_j(n) + \sum_{i=0}^{N} h_{ji}\Big[\sum_{k\in\mathfrak{I}_j} w_{jk}x_k\Big](n-i)\Big) \qquad (3)$$

where $w_{jk}(n)$ is defined according to:

$$w_{jk}(n-i) = v_{jk}(n) \qquad 0 \leqq i \leqq N \qquad (4)$$

This procedure, that reduces the total number of weights in a TDNN, is described by Shamma [3] in the context of continuous time. Effectively, (3) and (4) describe a TDNN in which the synaptic filters have been replaced by intra–neural filters. Shamma argues that the choice of the form of h_{ji} depends on the amount of detail required. However, determining the exact order and characteristics that each H_j should have to reach a maximum performance seems a difficult task, that requires a thorough understanding of the problem at hand. Training a TDNN is usually accomplished with a modified error back–propagation algorithm as in [1] and [3]. Here, the adapted backprop algorithm will be extended in order to determine the characteristics of the filters H_j.

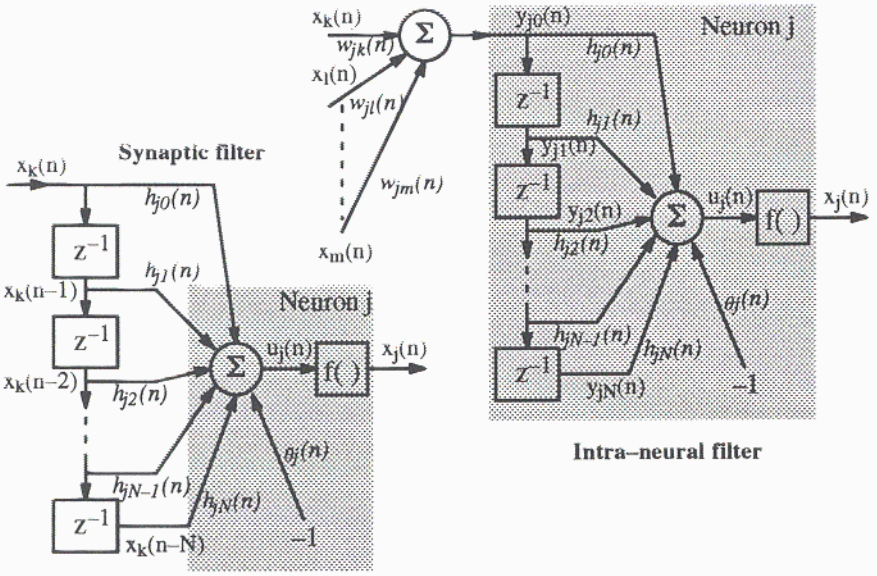

Fig.1 Two arrangements for a FIR–filter neural network.

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 51-56

First, recognize that from a mathematical point of view the decomposition of $h_{jki}(n)$ in (2) is not unique. Equally well, we may write:

$$h_{jki}(n) = v_{jk}(n) \cdot h_{ji}(n) \tag{5}$$

Using (4) together with (5) and substituting the result in (3) leads to:

$$x_j(n) = f\Big(-\theta_j(n) + \sum_{i=0}^{N} h_{ji}(n)[\sum_{k\in\mathfrak{I}_j} w_{jk}x_k](n-i)\Big)$$

$$= f\Big(-\theta_j(n) + \sum_{i=0}^{N} h_{ji}(n)y_{ji}(n)\Big) \tag{6}$$

Here, $y_{ji}(n)$ is defined as:

$$y_{ji}(n) = [\sum_{k\in\mathfrak{I}_j} w_{jk}x_k](n-i) \tag{7}$$

Together, (6) and (7) describe the output of a neuron with an adaptable intra–neural filter (see fig. 1) in response to the activation level of other neurons in the network.

III. Updating weights and filter coefficients

In the standard error back–propagation, weights are adjusted according to:

$$\Delta w_{jk}(n) = \alpha\Delta w_{jk}(n-1) - \eta\frac{\partial\mathfrak{E}(n)}{\partial w_{jk}(n)} \tag{8}$$

$\mathfrak{E}(n)$ denotes instantaneous sum of squared errors. If $\mathfrak{O}$ denotes the set of output neurons and $t_j(n)$ the target output of such a neuron at time n, $\mathfrak{E}(n)$ can be defined according to:

$$\mathfrak{E}(n) = \frac{1}{2}\sum_{j\in\mathfrak{O}}(t_j(n) - x_j(n))^2 = \frac{1}{2}\sum_{j\in\mathfrak{O}} e_j(n)^2$$

When the neurons are equiped with internal adaptable filters, two distinct kinds of weights appear and in addition to (8) the equation

$$\Delta h_{ji}(n) = \beta\Delta h_{ji}(n-1) - \mu\frac{\partial\mathfrak{E}(n)}{\partial h_{ji}(n)} \tag{9}$$

has to be solved. In case neuron j is an output neuron the partial derivative in (9) can be computed as:

$$\frac{\partial\mathfrak{E}(n)}{\partial h_{ji}(n)} = \frac{\partial\mathfrak{E}(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial x_j(n)} \cdot \frac{\partial x_j(n)}{\partial u_j(n)} \cdot \frac{\partial u_j(n)}{\partial h_{ji}(n)}$$

$$= e_j(n)\ (-1)\ f'(u_j(n))\ y_{ji}(n) \tag{10}$$

If we define $\delta_j(n)$ as $\delta_j(n) = e_j(n)\ f'(u_j(n))$ (11)

and use this definition while substituting (10) in (9), we obtain:

$$\Delta h_{ji}(n) = \beta\Delta h_{ji}(n) + \mu\delta_j(n)y_{ji}(n) \tag{12}$$

This equation is very similar to the one describing the weight adjustments in the standard backprop algorithm. Using the definition of $y_{ji}(n)$ (7), the partial derivative in (8) can be calculated as:

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 51-56

$$\frac{\partial \mathcal{E}(n)}{\partial w_{jk}(n)} = \frac{\partial \mathcal{E}(n)}{\partial u_j(n)} \cdot \frac{\partial u_j(n)}{\partial y_{j0}(n)} \cdot \frac{\partial y_{j0}(n)}{\partial w_{jk}(n)} = -e_j(n)\, f'(u_j(n))\, h_{j0}(n)\, x_k(n) \qquad (13)$$

If we define $\gamma_j(n)$ as $\gamma_j = \delta_j(n) h_{j0}(n)$ (14)

substitution of (13) in (8), while using this definition, produces

$$\Delta w_{jk}(n) = \alpha \Delta w_{jk}(n-1) - \eta \gamma_j(n) x_k(n) \qquad (15),$$

which describes the update for the synaptic weights. When neuron j is a hidden neuron, (14) still can be used to compute $\gamma_j(n)$. In this case, however, $\delta_j(n)$ can not be computed with (12) for we cannot directly associate an error–signal with a hidden neuron. Fortunately, a recursive definition for $\delta_j(n)$ can be derived. As this derivation is essentially the same as that for the backprop algorithm (see e.q. [4]) the result will be presented without further ado:

$$\delta_j(n) = f'(u_k(n)) \sum_k \gamma_k(n) w_{kj}(n) \qquad (16)$$

IV. Some fixed–order experiences

The research reported sofar arose from the experience gained in a number of projects, that were concerned with the prediction of natural dynamic processes. In the bio–chemical control for pharmaceutical purposes, it is clear that not only an analytical model for the controled process is not available, but also that a number of unknown phenomena can have a severe impact on the measurement data. Besides numerous electrical effects arising from the sensory set–up, also the body movements play an important role.

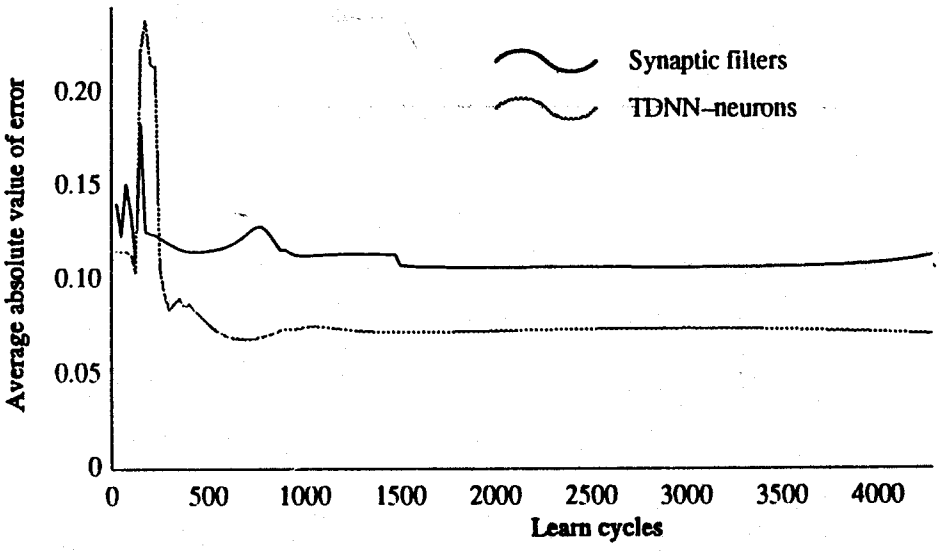

Fig.2 Observed learning curves for the synaptic and the TDNN network.

A first attempt to extract the time–dependent non–linear behaviour from the measurement data was performed by recursive neural networks. Even when the

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 51-56

recursion was integrated in the individual neurons, an acceptable fit was not found. Therefore the focus shifted towards synapse filters and time–delay nets. It was found that, though the fit was remarkably more close, the development effort was immense. The main reason was the effort spent on determining the order of the filters.

This observation was confirmed in a second series of experiments, that were concerned with the consumption of energy. In order to prepare for the movement of energy to individual households it must be brought in a transportable form. For economic reasons, this has to be done at the last moment; hence long–term and short–term prediction is required. Except for the influence of the weather, this experiment could be performed in a more controled manner. Though basically the synaptic and the TDNN approach could give similar results, the experiments show a consistent shorter and higher quality learning for the TDNN (Fig.2).

Apparently, both networks are trained in 1000 to 1500 learn cycles. A major difference, however, is present in their generalisation capabilities. The synaptic filter was still very sensitive to the individual inputpatterns, while the TDNN has very good generalisation properties. Whether this observation has a general significance or is due to the individual training parameters remains unclear sofar. As shown in Fig.3, the dynamic characteristics are adequately predicted, but the order of the filter is not optimal.

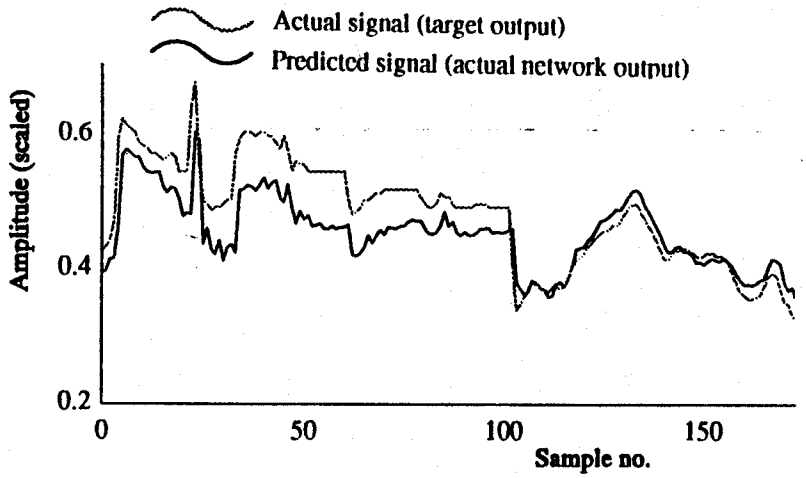

Fig.3: Prediction by a trained TDNN of a testpattern.

V. Determination of the filter–order

To determine the order of the filters, a technique similar to cascade–correlation learning [5] is used. Initially, each neuron in the network is equiped with a zero'th order filter (i.e. no delay elements are present) and the network is trained to

minimize the error–signal. After each pass over the entire training set, the improvement in performance is considered. If the error decreases less than by $b\%$, a delay element must be added to one of the neurons.

To determine which filter must be enlarged, the following procedure is adopted. Each neuron is assigned an additional candidate delay element. For the time being, the weighted output

$$a_j(n) = h_{jN+1}(n) y_{jN+1}(n) \tag{17}$$

of this candidate is not connected to the network. Furthermore, all weights except those associated with the newly created set of dangling connections are frozen and each $h_{jN+1}(n)$ is initialized randomly. Now, an additional p passes over the entire training set are made, during which the weights $h_{jN+1}(n)$ are adapted according to (12). After p passes, the most suitable candidate is selected. Its weighted output is connected to the network and all other candidates are removed.

Determination of the suitability of a candidate is accomplished by inspecting the contribution of its output ($y_{jN+1}(n)$) to the total error of the network. Therefore, if T denotes the size of the training set, the value of $\sum_{n=0}^{T} \left| \frac{\partial \mathcal{E}(n)}{\partial y_{jN+1}(n)} \right|$ (18)

has to be evaluated for each candidate. Again, the partial derivative in (18) can be expanded and using (11) we obtain:

$$\sum_{n=0}^{T} \left| -e_j(n) f'(u_j(n)) h_{jN+1}(n) \right| \quad \textit{and} \quad \sum_{n=0}^{T} \left| -\delta_j(n) h_{jN+1}(n) \right| \tag{19}$$

The candidate with the smallest value for (19) is considered to be the most suitable one. The use of cascade–correlated learning largely offsets the negative effect of the increased network size on the learning speed; furthermore the iterative search for the optimal filter order is removed and learning becomes more complete.

References

[1] A.Waibel, T.Hanazawa, G.Hinton, K.Shikano, K.J.Lang, *Phoneme recognition using time–delay neural networks*, IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 37 (1989), No. 3.

[2] E.A.Wan, *Temporal backpropagation for FIR neural networks*, IEEE International Joint Conference on Neural Networks, Vol.1 (1990), pp. 575–580

[3] S.Shamma, *Spatial and temporal processing in central auditory networks*, Methods in neuronal modeling (C.Koch & I.Segev, eds.), pp. 247–289

[4] S. Haykin, *Neural networks*, pp. 142–147, Macmillan College Publishing Co, Inc (1994)

[5] S.E. Fahlman, C. Lebiere, *The cascade–correlation learning architecture*, In: Advances in neural information processing systems 2 (D.S Touretzky, ed.), Morgan Kaufman Publishers (1990)