SLAC-PUB-6516
June 1994
(T)

Neural Network Construction via Back-Propagation*

THOMAS T. BURWICK

Stanford Linear Accelerator Center

Stanford University, Stanford, California 94309

ABSTRACT

A method is presented that combines back-propagation with multi-layer neural network construction. Back-propagation is used not only to adjust the weights but also the signal functions. Going from one network to an equivalent one that has additional linear units, the non-linearity of these units and thus their effective presence is then introduced via back-propagation ('weight-splitting'). The back-propagated error causes the network to include new units in order to minimize the error funtion. We also show how this formalism allows to escape local minima.

Talk presented at Stanford PDP Research Meeting, June 23, 1994

* Work supported by U.S. Department of Energy under contract DE-AC03-76SF00515.

1. Introduction Backpropagation has proved to be a most powerful tool for training multi-layer neural networks to fit a given set of input/output patterns. The error is back-propagated through the network to adjust the weights such that in successive steps the network turns into the desired function-approximator [1]. It is well known, however, that apart from adjusting the weights, the choice of the network architecture itself plays a crucial role for the performance. It is quite difficult, in general impossible to guess an optimal architecture from the begining. Instead, there are two alternatives. Either one begins with too many units and by pruning [2] or weight decay removes unnecessary units. Using back-propagation weight decay is achieved by adding penalty terms to the error function [3]. Alternatively, one may begin with not enough units and include additional units until the network is able to perform the required task (see for example [4], or more recently [5]). Although the latter possibility - neural network construction - is generally seen as being more elegant, the drawback is that so far there seems to be no natural way to embedd network construction into a back-propagation procedure. Nevertheless, given the success of back-propagation, it is highly desireable to find ways to combine back-propagation with network construction. In this paper we show how this may indeed be achieved by introducing what we call 'weight-splitting'.

How could a back-propagated error cause the creation of an additional unit? It seems rather impossible that discrete steps like adding units to a network could be implemented via back-propagation. What will help is that every network is equivalent to an infinite number of networks with additional linear units. Such units may be described as 'transparent' and thus 'invisible' to the forward propagating input. Suppose that in a first step we go to one of these equivalent networks – a trivial task – and in a second step we make the signal function subject to back-propagation. In order to minimize the error function back-propagation may then turn the linear into a non-linear unit: the additional unit becomes 'visible'. Thus it is the back-propagated error that causes the effective presence of an additional unit. Gradient descent gives a measure of how much is gained when introducing this unit.

In the following this procedure will be discussed in more detail. We will see that it will allow for such a variety of possible strategies that only experiments may tell what strategy is best. Given the vast number of possible ways to implement the method, we will restrict this paper to setting the stage for future experiments by explaining the theoretical framework. In section 2 we explain how signal functions can be varied and optimized by using a straightforward extension of back-propagation. In section 3 we show how this can be used to add new units to a network. This describes 'weight-splitting'. In section 4 we comment on different strategies how weight-splitting can be applied. Section 5 contains the summary.

2. Variable Signalfunctions and Extended Backpropagation Let us introduce the basic notation. Given the layers $n = 0, 1, ..., H+1$, the layer $n = 0$ is the input layer, $n = 1, ..., H$ are hidden layers and $n = H+1$ is the output layer. There are N_n units $j_n = 1, 2, ..., N_n$ in layer n. The signal of unit j_n in layer n is

$$S^n_{j_n} = g_{\alpha^n_{j_n}}(I^n_{j_n}) \quad , \quad I^n_{j_n} = \sum_{j_{n-1}} w^{n-1}_{j_n j_{n-1}} S^{n-1}_{j_{n-1}} \tag{1}$$

for $n = 1, ..., N$, while $S^0_{j_0} = \xi_{j_0}$ are the input signals. The weights w^{n-1} give the weighted input I^n to the units in layer n. We also write $S^{H+1}_{j_{H+1}} = O_{j_{H+1}}$ for the output signals. We use variable signal functions. A particular choice could be

$$g_\alpha(x) = \alpha \tanh(x) + (1-\alpha)x \tag{2}$$

In (1) we allow for different parameters α at each unit. It is essential for the following that the dependence on the parameter α is such that for a certain value the signal function reduces to a linear one. In the following signal functions $g_\alpha(x)$ are assumed to be linear for $\alpha = 0$.

It is straightforward to extend back-propagation to adjust not only the weights w but also the signal functions, i.e. the parameters α. We have to pick a certain error function, say

$$E[w, \alpha] = \sum_{j_{H+1}} (\zeta^\mu_{j_{H+1}} - O^\mu_{j_{H+1}})^2, \tag{3}$$

where (ξ^μ, ζ^μ), $\mu = 1, ..., p$, are the p input/output patterns that the network has to learn. Forward propagation is performed using (1). For back-propagation we also need

$$S'^n_{j_n} = g'_{\alpha^n_{j_n}}(I^n_{j_n}) \quad , \quad \dot{S}^n_{j_n} = \dot{g}_{\alpha^n_{j_n}}(I^n_{j_n}) \tag{4}$$

where

$$g'_\alpha(x) = \frac{d}{dx} g_\alpha(x) \quad , \quad \dot{g}_\alpha(x) = \frac{d}{d\alpha} g_\alpha(x) \tag{5}$$

For example (2) we have $g'_\alpha(x) = 1 - \alpha \tanh(x)^2$ and $\dot{g}_\alpha(x) = \tanh(x) - x$. Chosing a learning rate η the error function (3) is then minimized if the weights and signal functions are adjusted according to the gradient descent method:

$$\begin{aligned} \Delta w^n_{kl} &= -\eta \frac{\partial E}{\partial w^n_{kl}} = \eta \sum_\mu \Delta^{n+1,\mu}_k \cdot S'^{n+1,\mu}_k \cdot S^{n,\mu}_l \\ \Delta \alpha^{n+1}_k &= -\eta \frac{\partial E}{\partial \alpha^{n+1}_k} = \eta \sum_\mu \Delta^{n+1,\mu}_k \cdot \dot{S}^{n+1,\mu}_k \end{aligned} \tag{6}$$

The output error enters into

$$\Delta^{H+1,\mu}_{j_{H+1}} = \zeta^{\mu}_{j_{H+1}} - O^{\mu}_{j_{H+1}}. \tag{7}$$

For different error functions (3) only (7) will change, not the inner derivatives. These are obtained from the back-propagation rule:

$$\Delta^{m,\mu}_{j_m} = \sum_{j_{m+1}} \Delta^{m+1,\mu}_{j_{m+1}} S'^{m+1}_{j_{m+1}} w^m_{j_{m+1}j_m} \quad , \quad m = 1, ..., H \tag{8}$$

This completes describing the back-propagation procedure with varying signal functions. In fact, the procedure is easily generalized to signal functions that depend on several parameters. The form of the signal function maybe under complete control if we allow for an arbitrary number of parameters. For the purpose of this paper and the sake of clarity it is sufficient to work with signal functions like (2) that have only one parameter. A study of more complex varying signal functions and their effect on speeding up convergence can be found in [6].

3. Weight-Splitting and Neural Network Construction We will now show how (6) can be used to add new units to a network. The first step is trivial and involves going from a given network architecture to another with additional units that are linear. The effective presence of the new units is only introduced in a second step where back-propagation changes the new signal functions to non-linear ones.

Take for example the situation shown in fig. 1. We choose our notation such that the new units are $j = 1, ..., N$, the signals are V_j and the corresponding parameters α_j. The new weights that replace the w are u, v. Both networks are the same if

$$\sum_{j_n} w^n_{j_{n+1}j_n} S^n_{j_n} = \sum_j v_{kj} g_{\alpha_j}(\sum_{j_n} u_{jj_n} S^n_{j_n}). \tag{9}$$

for all possible signals $S^n_{j_n}$. Thus the signal functions of the new units indeed have to be linear, $\alpha_j = 0$, and the the choice of the new weights is only restricted by

$$w^n_{j_{n+1}j_n} = \sum_j v_{j_{n+1}j} u_{jj_n} \tag{10}$$

Notice that (10) may be written as a matrix equation $W = VU$ where V is a $N_{n+1} \times N$ and U a $N \times N_n$ matrix. A simple choice would be U (V) diagonal, $N = N_n$ (N_{n+1}), giving $V = WU^{-1}$ ($U = V^{-1}W$) immediately.

The new units begin to play a role only after they turn non-linear and this is where extended back-propagation enters. It is easily seen how introducing the new units via back-propagation helps to improve the performance. For that purpose we introduce also the notation

$$\Delta_j^\mu = \sum_{j_{n+1}} \Delta_{j_{n+1}}^{n+1,\mu} \, S'^{n+1,\mu}_{j_{n+1}} \, v_{j_{n+1}j} \tag{11}$$

for the error that is back-propagated to the new units. This is analog to (8). Using gradient descent the change in the error function due to changes in u, v and α is given by (η small)

$$\Delta E \simeq \frac{-1}{\eta} \sum \left[(\Delta v_{j_{n+1}j})^2 + (\Delta u_{jj_n})^2 + (\Delta \alpha_j)^2 \right] \tag{12}$$

where the extended back-propagation rules (6)-(8) give ($\alpha_j = 0$)

$$\begin{aligned}
\Delta v_{j_{n+1}j} &= \eta \sum_\mu \Delta_{j_{n+1}}^{n+1,\mu} \, S'^{n+1,\mu}_{j_{n+1}} \, S_j^\mu \\
&= \sum_{j_n} u_{jj_n} \, \Delta w^n_{j_{n+1}j_n} \qquad (13) \\
\Delta u_{jj_n} &= \eta \sum_\mu \Delta_j^\mu \, S'^\mu_j \, S^{n,\mu}_{j_n} \\
&= \beta \sum_{j_{n+1}} v_{j_{n+1}j} \, \Delta w^n_{j_{n+1}j_n} \qquad (14) \\
\Delta \alpha_j &= \eta \sum_\mu \Delta_j^\mu \, \dot{S}_j^\mu \\
&= \eta \sum_{j_{n+1}} v_{j_{n+1}j} \sum_\mu \Delta_{j_{n+1}}^{n+1,\mu} \, S'^{n+1,\mu}_{j_{n+1}} \, \dot{S}_j^\mu \qquad (15)
\end{aligned}$$

In (14) we used $S'^\mu_j = \beta$ since $g_0(x) = \beta x$. For example (2) this is $\beta = 1$. Let us now assume that upon varying the weights w we arrived at a minimum so that

$$\Delta E = \frac{-1}{\eta} \sum (\Delta w^n_{j_{n+1}j_n})^2 = 0 \quad \text{and} \quad \Delta_{j_{H+1}}^{H+1,\mu} \neq 0 \tag{16}$$

for some μ. This may be a global minimum (for the given architecture) due to a lack of units so that the network cannot learn the required p patterns. For the equivalent network varying u, v at $\alpha_j = 0$ corresponds to varying w and so (13), (14) and (16) give $\Delta u = \Delta v = 0$. However, the $\Delta \alpha_j$ in (15) maybe non-zero,

leading to $\Delta E < 0$ in (12). Moreover, once that $\alpha_j \neq 0$ the relations (13), (14) will no longer hold and in the next back-propagation steps also u, v will be changed. The network begins to 'roll down' the error function first in the α_j and then also the weight directions. This is how introducing a new unit via back-propagation will cause the network to leave the minimum of the old architecture.

Notice that (16) may describe a local minimum. In that case (15) helps to escape the local minimum at the cost of introducing a new unit - another possible application of weight-splitting.

4. Possible Implementations of Weight-Splitting We mentioned before that for every network there is an infinite number of equivalent networks with additional linear units. As a consequence there is a variety of ways that may be taken to apply weight-splitting. To illustrate this point we will now first present a particular algorithm and then discuss possible modifications.

Using a notation slightly different from section 2, we take H to be the maximal number of hidden layers that we want to allow, and introduce $h \leq H$ as the number of layers that the weight-splitting algorithm introduced up to a certain back-propagation step. A unit j_n in layer n is said to be active iff $\alpha^n_{j_n} \neq 0$. A layer is said to be (completely) active if some (all) of its units are active. Introducing a new unit refers to activating this unit. We assume that the hidden layers $n = h+1, ..., H$ are not active, i.e., their units are linear, $\alpha^n_{j_n} = 0$.

We define every hidden layer to have the same number of hidden units, $N_n = N$, $n = 1, ..., H$. The number of active units in layer h is N'. The signal function of the output unit is fixed.

We also use the notion of active weights. A weight $w^n_{j_{n+1}j_n}$ is said to be active if $n = H$, or if the unit j_{n+1} in layer $n+1$, $n = 0, ..., H-1$, is active. Only active weights will be varied by back-propagation.

The following algorithm will search for an error-minimum using the current architecture. Whenever it reaches a minimum that is non-vanishing it will introduce, i.e. activate, an additional hidden unit.

0) Initialize the weights w^H to small random values. Set $h = 0$, $N' = N$, and $w^n_{j_{n+1}j_n} = \delta_{j_{n+1}j_n}$ for $n = 0, ..., H-1$.*

1) Vary active weights and active signal functions using back-propagation (6) until the error (3) reaches a minimum. Notice that standard techniques may be applied (e.g. simulated annealing) to assure that the minimum is global for the given architecture.

---

* δ_{kl} is the Kronecker symbol: $\delta_{kk} = 1$ and $\delta_{kl} = 0$ if $k \neq l$.

2) If this minimum is smaller than some predefined tolerance terminate the procedure with success.

3) Activate a new unit. If $N' < N$ reset $N' \to N'+1$. If $N' = N$, $h < H$, reset $h \to h+1$, $N' = 1$. If $N' = N$, $h = H$ terminate the procedure without success (the N, H may have been predefined too small).

4) Return to step 1.

A most simple example is shown in fig. 2. It is essential that forward and backward propagation for non-activated signal functions and weights need no computation. This is why units are effectively not present until they are activated.

There are numerous possibilities to modify the above procedure. Instead of establishing a new unit only when a non-vanishing error minimum is reached, one could make all signal functions subject to back-propagation from the beginning ($h = H$). In that case one should shift the error function

$$E \to E + P[\alpha] \tag{17}$$

with a term, $P[\alpha] \sim \alpha^2$, that penalizes the presence of every new unit. Alternatively and returning to the above procedure of introducing units only when an error-minimum is reached, one may choose $P[\alpha] \sim (\alpha - 1)^2$. This would accelerate the back-propagation steps of introducing the new unit, $g_\alpha(x) \to \tanh(x)$ for example (2). Both approaches may be combined in

$$P[\alpha] \sim (\alpha - \frac{1}{2})^4 - \frac{1}{2}(\alpha - \frac{1}{2})^2 + \frac{1}{16} \tag{18}$$

This is minimal for absence, $\alpha = 0$, or presence, $\alpha = 1$, of a unit. Small α are penalized, but once the gain in decreasing the error overcomes the penalty, the term (18) will support introducing this new unit. Of course, there is no need to continue varying the signal function once a unit is established, i.e., $\alpha = 1$.

Following the above algorithm, in step 3 units of a new layer are always activated in the same order. Alternatively, one may use (15) to establish criteria that decide what unit should be activated next. Also, there is freedom of choosing how many neurons should be activated, and along what architecture (see fig. 3). One could allow for $N_n \neq N$ in hidden layers. Moreover, instead of activating only one unit whenever step 3 was reached, there may be cases where many hidden units are needed so that more units should be activated at once. Then step 3 could be modified to activate for example a whole layer, combined with a penalty term as in (17) with $P[\alpha] \sim \alpha^2$.

5. Conclusions and Outlook The triviality of linear units in a feed-forward network can be turned into an advantage if applied to network construction. Back-propagation can then be used to introduce new units in a two step procedure. The first step is simple and consists of going from one network architecture to an equivalent one with additional linear units. This network will behave exactly as the original architecture. In a second step, however, we make the parameters of the signal function subject to back-propagation. The back-propagated error will then turn the linear into non-linear units, thereby establishing their effective presence ('weight-splitting'). This method may also be applied to escape local minima at the cost of introducing an additional unit (which may be removed again by pruning and weight-decay).

There is an infinite number of equivalent networks with additional linear units, and as a consequence a lot of freedom for implementing the first step of weight-splitting. Also, there is a freedom of chosing at what step of back-propagation the new units should be established. Moreover, one should notice that there is another way to combine back-propagation with network construction. It consists of going from one network architecture to another with additional non-linear units that are effectively not present due to vanishing weights. The back-propagated error will then create non-vanishing weights. Such a trivial approch, however, cannot create a multi-layer network like the one shown in fig. 2, where the signals of each layer are mapped exclusively to the next layer. Nevertheless, combining both approaches is possible. It is also possible to combine weight-splitting with other construction algorithms. Comparing all the different ways to apply weight-splitting will require extensive numerical studies. Using the algorithm that we described in section 4 together with a shift (17) where $P[\alpha] \sim (\alpha - 1)^2$ accelerates introducing the new unit, $g_0(x) = x \to g_1(x) = \tanh(x)$ for example (2), seems to be the most conservative approach and will evidently lead to the right architecture. But more efficient ways to apply weight-splitting are still to be established and we leave their studies to future investigations.

Acknowledgements It is a pleasure to thank David Rumelhart for valuable discussions.

REFERENCES

[1] A.E. Bryson and Y.-C. Ho (1969), 'Applied Optimal Control', New York: Blaisdell;

P. Werbos (1974), 'Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences', Ph.D. Thesis, Harvard University;
D.B. Parker (1985), 'Learning Logic', Technical Report TR-47, Center for Computational Research in Economics and Management Science, Massachussetts Institute of Technology, Cambridge, MA;
D.E. Rumelhart, G.E. Hinton, and R.J. Williams (1986), 'Learning Representations by Back-Propagating Errors', *Nature* **323**, 533-536; 'Learning Internal Representations by Error Propagation', in D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing*, vol. 1, chap. 8, Cambridge: MIT Press.

[2] J. Sietsma and R.J.F. Dow (1988), 'Neural Net Pruning - Why and How', in *IEEE International Conference on Neural Networks*, San Diego 1988, vol. I, 325-333, New York: IEEE.

[3] G.E. Hinton (1986), 'Learning Distributed Representations of Concepts', in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst 1986, 1-12. Hilldale: Erlbaum;
R. Scalettar and A. Zee (1988), 'Emergence of Grandmother Memory in Feedforward Networks: Learning with Noise and Forgetfulness', in *Connectionist Models and Their Implications. Readings from Cognitive Science*, eds. D. Waltz and J.A. Feldman, 309-332, Norwood: Ablex;
A.H. Kramer and A. Sangiovanni-Vinventelli (1988), 'Efficient Parallel Learning Algorithms for Neural Networks', in Advances in Neural Information Processing Systems I (Denver 1988), ed. D.S. Touretzky, 40-48, San Mateo: Morgan Kaufmann;
S.J. Hanson and L. Pratt (1988), 'A Comparison of Different Biases for Minimal Network Construction with Back-Propagation', in Advances in Neural Information Processing Systems I (Denver 1988), ed. D.S. Touretzky, 177-185, San Mateo: Morgan Kaufmann;
Y. Chauvin (1988), 'A Back-Propagation Algorithm with Optimal Use of Hidden Units', in Advances in Neural Information Processing Systems I (Denver 1988), ed. D.S. Touretzky, 519-526, San Mateo: Morgan Kaufmann.

[4] M. Mézard and J.-P. Nadal (1989), 'Learning in Feedforward Layered Networks: The Tiling Algorithm', *Journal of Physics* **A22**, 2191-2204;
M. Marchand, M. Golea, and P.Ruján (1990), 'A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons', *Europhysics Letters* **11**, 487-492;
M. Frean (1990), 'The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks', *Neural Computation* **2** (1990), 198-209.

[5] S.G. Romaniuk and L.O. Hall (1993), 'Divide and Conquer Neural Networks', *Neural Networks*, vol. 6, 1105-1116; and references therein.

[6] J.A. Drakopoulos (1994), 'Multi-Sigmoidal Neural Networks', Technical Report, Department of Computer Science, Stanford University, CA; and references therein.

FIGURE CAPTION

FIG.1 Starting from a given architecture (a) N new units are introduced by weight-splitting (b). Both networks are equivalent if the new units are linear, and the weights obey eq.(10). Only the back-propagated error will change the new units to non-linear ones, thereby establishing their effective presence.

FIG.2 The XOR-problem is the simplest that needs hidden units. Applying the algorithm described in section 4 with $H = 2$, one may start with no hidden units, $h = 0$, (a) and after activating a first hidden unit, $h = 1$, $N' = 1$, (b) and a second, $N' = 2$, (c) one arrives at an architecture that solves the problem. (Broken circles and lines correspond to non-activated units and weights.)

FIG.3 There are many possible ways how to apply weight-splitting. The figure demonstrates for a simple situation how connections could be splitted into one, two, three, etc. new units.

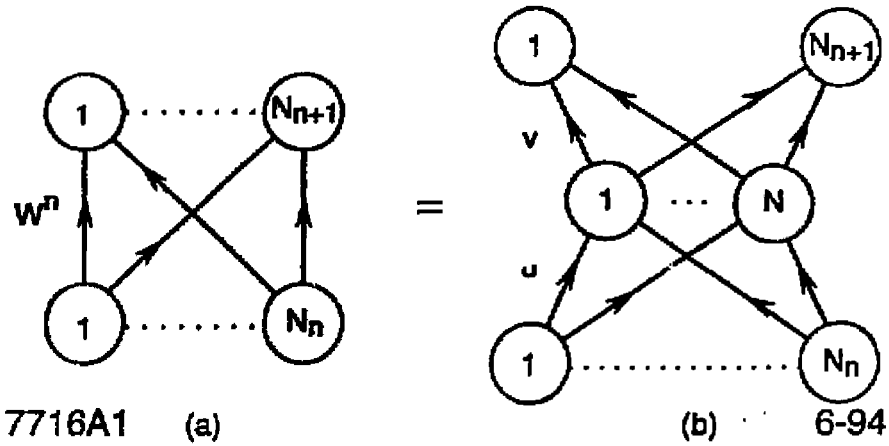

Fig. 1

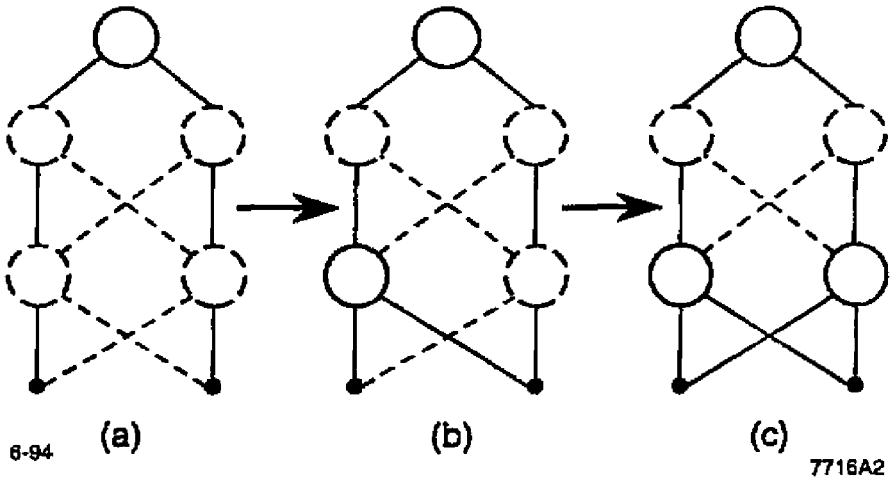

Fig. 2

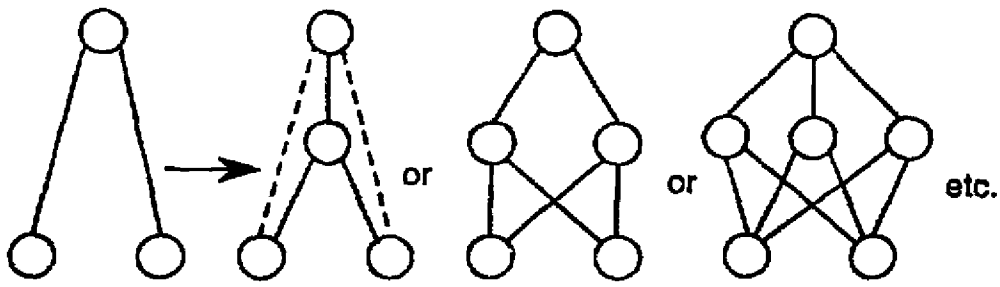

Fig. 3