Complex Systems **5** (1991) 371–380

Optimization of the Architecture of Feed-forward Neural Networks with Hidden Layers by Unit Elimination

Anthony N. Burkitt
Physics Department, University of Wuppertal,
Gauss-Straße 20, D-5600 Wuppertal 1, Germany

Abstract. A method for reducing the number of units in the hidden layers of a feed-forward neural network is presented. Starting with a net that is oversize, the redundant units in the hidden layer are eliminated by introducing an additional cost function on a set of auxiliary linear response units. The extra cost function enables the auxiliary units to fuse together the redundant units on the original network, and the auxiliary units serve only as an intermediate construct that vanishes when the method converges. Numerical tests on the Parity and Symmetry problems illustrate the usefulness of this method in practice.

1. Introduction

One of the central unresolved questions in the study of neural networks concerns the architecture and size of the network that is most appropriate to a particular problem. Learning procedures such as back-propagation [1] provide a means of associating pairs of input and output patterns in a feed-forward layered neural network, provided that there are enough hidden units. These learning algorithms are typically implemented on a chosen *fixed* architecture where the number of hidden units must be given and does not change during the learning procedure.

Deciding upon the number of hidden units in a network is somewhat of an art and matter of experience. It is clear that if the number of free parameters in a net is too large then the net will be able to fit the training data arbitrarily closely and essentially memorize the training patterns. However such networks are typically not able to generalize to new inputs. The folklore is that the number of weights in a network should be less than one-tenth the number of training patterns. The capacity of a network to generalize from the learned data to new inputs has been shown [2–4] to be most pronounced for a network containing a minimal number of units and connections. Moreover, the amount of computation both for forward computation and for learning grows with increasing size of the network, and it is therefore more efficient to keep the size of the network as small as possible.

Several researchers have proposed schemes to achieve near-minimal networks based upon either reducing oversize networks during training or explicitly building up networks until learning is achieved. The method presented here is such a network reduction (or *pruning*) algorithm but, in contrast to *weight reduction* algorithms (where the number of non-zero weights in the network is reduced), this method is based upon the elimination of redundant hidden units.

A number of weight reduction solutions have been proposed by adding terms to the standard sum squared error cost function:

$$E = \sum_{n} E_p = \frac{1}{2} \sum_{n,p} (t_{pn} - o_{pn})^2 \tag{1.1}$$

where the sum is over the n output units and p input/output training patterns, whose target output and network output on the output unit n are t_{pn} and o_{pn}, respectively. In the *weight decay* method [5, 6] an oversized net is taken and a minimization of the cost function is carried out in the usual way, but with an additional term that is equivalent to adding a quadratic cost term for every weight in the network. The weights that do not contribute to learning will then have an exponential time decay. In the *weight elimination* method [7] an additional term, which can be thought of as a complexity term, likewise encourages the reduction and eventual elimination of many of the connections. Another method [8] for eliminating superfluous weights is to estimate the sensitivity of the global cost function to each of the weights. This can be done by keeping track of the incremental changes to the weights during learning and by discarding connections with weights that have a low sensitivity. This method has the advantage that no modification of the cost function is required, and there is therefore no interference with the learning process while the sensitivities are being measured.

Alternative methods have been proposed [9–11] that focus on the internal representations (i.e., the states of the hidden units) rather than a cost function in weight space. In these methods the number of hidden units is not fixed in advance, but rather is determined by the algorithm itself. The *sequential learning algorithm* [9] uses the internal representation together with the perceptron algorithm to construct a network that is near-minimal and has acceptable learning times. The *tiling algorithm* [10] and related algorithms [11] proceed by including layers and units within layers until convergence is achieved. A further line of approach to this problem is by the application of *genetic algorithms* [12–15], in which the structure of the network is encoded in bit-strings (chromosomes) that can evolve to produce better network architectures.

The method presented here focuses on the units of the hidden layer(s) and essentially fuses together units that are redundant. Not only is this of practical use in reducing the size of a network, but by minimizing the number of hidden nodes we can hope to gain more insight into the behavior of the network, not only in terms of its possible description by a simple set of rules but also by considering the function of each of the hidden units. The general method is discussed in the next section, together with several variations. The third section contains results of the method as applied to the Parity and Symmetry problems with a single output unit and one hidden layer.

2. The unit elimination method

Consider an oversize feed-forward network with a single layer of N_H hidden units (H). A layer (S) of N_S auxiliary units is introduced *between* the original hidden layer and the output units (for simplicity it will be assumed here that $N_S = N_H$). There is a full set of connections now between this hidden auxiliary layer S and both the output units O and the original hidden layer H, whose weights are labeled by $w_{o_i s_j}$ and $\omega_{s_j h_k}$, respectively. There are no longer any direct connections between the hidden layer H and the output units O.

The weights of the connections between the auxiliary layer S and the original hidden layer H, which here are called auxiliary weights, are restricted to be excitatory and such that the *sum* of weights on each auxiliary node is unity:

$$\sum_{h_k=1}^{N_H} \omega_{s_j h_k} = 1, \quad \text{for each } s_j \in S \tag{2.1}$$

The auxiliary units have a *linear response*—their activity is simply the sum of their inputs:

$$a_{s_j} = \sum_{h_k=1}^{N_H} \omega_{s_j h_k} a_{h_k} \tag{2.2}$$

where a_{h_k} is the activity of the hidden unit h_k and there is no bias on the auxiliary units. The units in the hidden layer H and the output units have the usual sigmoid-function response.

We proceed now by carrying out training of the extended net using back-propagation [1] in the usual way by successively presenting the patterns to be learned. In this stage of the algorithm all the weights are iterated by carrying out a minimization of the standard error function (equation (1.1)). The constraint on the auxiliary weights of equation (2.1) is imposed at each iteration of the learning procedure by normalizing the iterated weights $\tilde{\omega}_{s_j h_k}$ explicitly:

$$\omega_{s_j h_k} = \tilde{\omega}_{s_j h_k} \Big/ \sum_{h_k=1}^{N_H} \tilde{\omega}_{s_j h_k} \tag{2.3}$$

When the network has found a first solution (i.e., the network output agrees with the target output—within the learning tolerance—for each training pattern) then the auxiliary weights $\omega_{s_j h_k}$ between the auxiliary layer S and the hidden layer H experience a cost term of the following form:

$$D(\omega) = \frac{1}{2}\beta N_S \sum_{s_j=1}^{N_S} (1 - d(\vec{\omega}(s_j))) \tag{2.4}$$

where the vector $\vec{\omega}(s_j)$ represents the weights $\{\omega_{s_j h_k}\}$ $(h_k = 1, \ldots, N_H)$, the coefficient β is a constant that can be adjusted, and the function $d(\vec{\omega}(s_j))$ is chosen as

$$d(\vec{\omega}(s_j)) = \frac{1}{2}\sum_{s_l=1}^{N_S} [\vec{\omega}(s_j) \cdot \vec{\omega}(s_l)]^2 = \frac{1}{2}\sum_{s_l=1}^{N_S} \left[\sum_{h_k=1}^{N_H} \omega_{s_j h_k}\omega_{s_l h_k}\right]^2 \tag{2.5}$$

This cost term has a two-fold effect. First, it causes each auxiliary unit to choose exactly *one* unit in the hidden layer H; that is, the weights $\omega_{s_j h_k}$ for unit s_j will have the value 1 for one hidden unit h_r and will vanish for all other h_k. This comes about by the term $s_j = s_l$ in the sum (2.5) above, which acts as a weight enhancement term that ensures that the weights iterate toward a solution in which each unit s_j chooses only *one* node h_r in the hidden layer H. Second, this cost term will cause the auxiliary units to *attract*; the auxiliary units will tend to choose the *same* hidden units if this is compatible with learning. The cost function chosen above is a unit-attracting potential that has the effect of maximizing those weight vectors $\vec{\omega}(s_j)$ that have large overlaps (i.e., point to the same units in the hidden layer H). The coefficient β can be adjusted to ensure optimal convergence of the procedure.

To ensure that the weights $\omega_{s_j h_k}$ remain excitatory $(\omega_{s_j h_k} \geq 0)$ it is useful to introduce the variables $\rho_{s_j h_k} = (\omega_{s_j h_k})^{1/2}$. The learning procedure can be carried out with the weights $\{\rho_{s_j h_k}\}$ in the usual way:

$$\Delta\rho_{s_j h_k}(t+1) = \alpha\Delta\rho_{s_j h_k}(t) + 2\epsilon\rho_{s_j h_k}(t)\begin{cases} \delta_{s_j} a_{h_k} & \text{back-prop} \\ \beta\sum_{s_l=1}^{N_S} \omega_{s_l h_k}(\vec{\omega}(s_j)\cdot\vec{\omega}(s_l)) & \text{otherwise} \end{cases} \tag{2.6}$$

with $\omega_{s_j h_k} = \rho^2_{s_j h_k}$, and where ϵ is the learning rate and α the momentum. In the implementation of this method it is also possible to include the unit-attracting term *throughout* the learning process, and results where this has been done will also be discussed in the next section.

A typical training of a network proceeds as follows. First the complete network is trained using standard back-propagation until the network finds a solution. The auxiliary units are then iterated further with the unit-attracting term and the ρ-parameterization of equation (2.6) until they either all choose a unit in the hidden layer or back-propagation is resumed in order to relearn some of the training patterns. The auxiliary weights are considered to have chosen a unit h_r in the hidden layer when the value of the associated weight $\omega_{s_j h_r}$ exceeds a value ω_{fix} (typically chosen between 0.7 and 0.95), at which point its value is fixed at unity (and all other components are fixed at zero). Training continues until the network *both* learns all the patterns *and* has each auxiliary unit choose one unit on the hidden layer H. In the course of training a typical network the auxiliary weights switch a couple of times between the back-propagation and the unit-attracting terms of equation (2.6) before a reduced network solution is reached.

When the weights $\omega_{s_j h_k}$ have all converged to values of 0 or 1, then it is straightforward to see that those units that are chosen by the auxiliary units constitute the new *reduced* layer of hidden units. Those units in the hidden layer with no non-zero weight connecting them to auxiliary units in S can be eliminated. The auxiliary units that choose the same unit in the hidden layer H, by virtue of their *linear* response, can simply be added together to give the new weights between the reduced hidden layer and the output units. This can be seen by considering the net input to the output nodes o_n:

$$net_{o_n} = \sum_{s_j=1}^{N_S} w_{o_n s_j} a_{s_j} = \sum_{s_j=1}^{N_S} w_{o_n s_j} \left(\sum_{h_k=1}^{N_H} \omega_{s_j h_k} a_{h_k} \right) = \sum_{r_i=1}^{N_R} \left(\sum_{s_r} w_{o_n s_r} \right) a_{r_i} \tag{2.7}$$

where the sum over $\{r_i\}$ is the sum over the reduced hidden layer—those nodes on the hidden layer that have a non-zero weight to the auxiliary layer—and the sum $\{s_r\}$ is over those units on the auxiliary layer that choose the hidden unit r. The resulting weight between a reduced unit and the output is therefore simply the sum of the weights over connected auxiliary units to the output, as seen in the bracket on the extreme right of equation (2.7).

There are a number of ways to see how and why this algorithm works. One way is to view the hidden units as hyperplanes that divide the patterns into two sectors (a picture that is, of course, strictly true only for threshold nets). The auxiliary units then act to eliminate hyperplanes that divide the input patterns in the same, or very nearly the same, manner. Suppose, for example, that two hidden units h_1 and h_2 have developed very nearly the same weights under back-propagation, so the weight vector components $\omega_{s_l h_1}$ and $\omega_{s_l h_2}$ will be almost identical. Assuming that either h_1 or h_2 is required for learning, the cost term $d(\vec{\omega}(s_l))$ in (2.5) on the auxiliary unit s_l will then cause only *one* of the two units to be choosen. The other unit will effectively be "turned off" as its weights are forced to zero by imposing the normalization condition (2.1) at each learning step. Another equivalent way of viewing the "turning off" of hidden units is as a variant of competitive learning, where the auxiliary units behave as regularity detectors. The condition (2.1) ensures that the *reinforcement* of a connection between an auxiliary unit and a particular hidden unit will effectively *inhibit* the connection to other hidden units. As learning further proceeds, such redundant hidden units are gradually decoupled from the output unit. In the implementation here, such units are explicitly turned off by setting their weights to zero when one unit has accumulated a weight of ω_{fix}, as discussed above.

3. Implementation on Parity and Symmetry problems

To test this method it was implemented on the Parity and Symmetry problems, in which there is a single output unit and where only one hidden layer is necessary to obtain solutions. The minimal number of hidden units necessary to give solutions for both these problems are known. The Parity function as implemented here gives an output +1 when the number of input units N_I with +1 is odd and an output 0 otherwise. The minimal number of hidden units N_H required for the Parity problem is $N_H = N_I$ [1]. In the symmetry problem the output is +1 if the pattern on the input units is symmetric about the center, otherwise the output is 0. The symmetry problem requires a minimum of two hidden nodes for its solution [1].

The training patterns were cycled through in order, and the weights were adjusted after each pattern. The back-propagation [1] algorithm was used with learning rate (ϵ) and momentum (α) parameters as defined in equation (2.6). The patterns were considered learned when the output was within a learning tolerance of 0.1 of the correct answer. To ensure that the network did not overlearn, back-propagation was not carried out on inputs whose output was within the learning tolerance [16]. The initial weights were chosen randomly on a scale $r = 2.5$ except for the auxiliary weights, which were all set to the value $\omega_{s_j h_k} = 1/N_H$.

A cutoff time was also introduced for the learning procedure, whereby if the training had not been completed within this number of iterations then the network was considered stuck in a local minimum. To give an indication of the performance of the method, both the success rate and the average training time are given. The success rate is simply the percentage of training runs that converged to a result before the cutoff time, and the average training time τ is given by [17]

$$\tau = \left(\frac{1}{n} \sum_{i=1}^{n} R_i \right)^{-1} \tag{3.1}$$

where the sum is over the training runs, and R_i is the inverse training time for successful runs and zero otherwise. Training times and success rates for the first solution with back-propagation (i.e., before the unit-attracting term was turned on) are given, as well as the training times and success rates for the complete node elimination algorithm (i.e., to find a reduced network).

Parity

The $N_I = 2$ Parity problem (XOR) was investigated starting with various numbers of nodes in the hidden layer: $N_H = 5, 8, 12$. The results are summarized in table 1, where N_R is the number of hidden nodes in the reduced net and the results represent an average over 1000 random starting network configurations. The percentages for the success rates of back-propagation and the node elimination algorithm are taken with respect to the total number of initial network configurations. The percentages for the various values of

		back-propagation		nodes		N_R (as %)		
N_H	β	τ_{bp}	success (%)	τ_{nodes}	success (%)	2	3	4
5	0.01	35	100	50	97	34	63	3
	0.001	27	100	48	100	11	71	18
	0.002	"	"	47	99	13	71	16
8	0.005	"	"	46	98	14	73	13
	0.01	"	"	46	96	17	72	11
	0.05	"	"	48	90	24	68	8
	0.1	"	"	50	85	28	65	7
12	0.01	25	99	52	88	13	75	12

Table 1: Results for the $N_I = 2$ Parity problem (XOR) with $\epsilon = 1.0$, $\alpha = 0.94$, and $\omega_{\text{fix}} = 0.7$. The network starts with N_H units in the hidden layer, and the method reduces the net to N_R hidden units.

N_R are taken with respect to the number of initial network configurations upon which the algorithm was successful. The learning rate was chosen to be $\epsilon = 1.0$, the momentum $\alpha = 0.94$, and the auxiliary weights were considered to have chosen a unit in the hidden layer when the corresponding weight exceeded $\omega_{\text{fix}} = 0.7$. A cutoff of 1000 presentations of the learning patterns was used throughout. The results show a dramatic reduction in the size of the network—the number of hidden nodes is reduced in almost all cases to between 2 and 4. It is interesting to note that the "grandmother cell" solution ($N_R = 4$) is only chosen in a small number of cases, although the initial oversize networks could easily allow it. The effect of varying the strength of the unit-attracting term β can also be seen. Increasing β tends to reduce the network size somewhat, but the success rate is also decreased.

Runs with higher values of ω_{fix} (e.g., 0.95) give longer training times but also tend toward smaller resultant networks, whereas reducing ω_{fix} decreases the training time but tends to give slightly larger resultant networks. Several trials were also made in which the auxiliary weights experience the unit-attracting term throughout the learning process. This generally gives substantially smaller resultant networks on average, but requires longer overall training times and has lower success rates.

In the case of the $N_I = 4$ Parity problem, 1000 initial network configurations were averaged over and a cutoff of 1000 presentations for the learning patterns was used. The network parameters took the values $\epsilon = 0.25$, $\alpha = 0.96$, and $\omega_{\text{fix}} = 0.7$. The results, shown in table 2, show a large reduction in the size of the network with the same qualitative behavior as that observed for the XOR network. The network tends to find the minimal solution ($N_R = 4$) more often for larger values of β and ω_{fix}, but the overall success rate decreases when these values increase. The observation that the network does not necessarily seek solutions with a minimal number of hidden units probably indicates that solutions with more hidden units are more numerous than the minimal solution(s), as discussed in section 10 of [2].

		back-propagation		nodes		N_R (as %)				
N_H	β	τ_{bp}	success (%)	τ_{nodes}	success (%)	4	5	6	7	8
8	0.001	142	98	200	94	9	63	26	2	–
	0.01	"	"	192	94	12	65	22	1	–
	0.1	"	"	192	91	14	67	19	–	–
12	0.01	95	100	136	99	2	45	41	11	1

Table 2: Results for the $N_I = 4$ Parity problem with $\epsilon = 0.25$, $\alpha = 0.96$, and $\omega_{\text{fix}} = 0.7$.

		back-propagation		nodes		N_R (as %)				
N_H	β	τ_{bp}	success (%)	τ_{nodes}	success (%)	2	3	4	5	6
5	0.01	50	93	60	90	19	55	25	1	–
8	0.001	37	99	49	98	3	31	49	16	1
	0.005	"	"	49	98	4	32	48	15	1
	0.01	"	"	49	98	4	34	48	13	1
	0.02	"	"	48	98	4	34	48	13	1
	0.1	"	"	49	98	4	34	48	13	1
12	0.01	45	85	64	84	1	15	43	32	9

Table 3: Results for the $N_I = 4$ Symmetry problem with $\epsilon = 0.5$, $\alpha = 0.9$, and $\omega_{\text{fix}} = 0.7$.

Symmetry

The Symmetry problem requires a minimum of $N_R = 2$ hidden units for its solution [1]. The results of the unit elimination algorithm for this problem with $N_I = 4$ input units are shown in table 3. The averages for the $N_H = 8$ networks are taken over 10000 randomly chosen initial network configurations, and for the $N_I = 5, 12$ networks the averages are over 1000 network configurations. The cutoff time for training was 1000 presentations of each of the sixteen training patterns, the learning rate was $\epsilon = 0.5$, momentum $\alpha = 0.9$, and $\omega_{\text{fix}} = 0.7$. The results show a distribution of reduced network sizes, ranging from the absolute minimum of 2 up to 6 hidden units, and that is relatively constant over a large range of values of β.

4. Discussion and conclusions

The algorithm presented here enables us to reduce the size of a network by the elimination of hidden units. Tests of the algorithm on some Boolean functions have shown that it produces a substantial reduction in the size of the network.

A number of possible variations of the algorithm are possible. In the examples given here the number of auxiliary units introduced is equal to that of the hidden units, and each auxiliary unit has a full set of connections to all units in the hidden layer. Both the number of auxiliary units and their connectivity can be varied, which is a feature that may be of considerable use in problems involving many hidden units. In such a case it would be possible to limit the number of auxiliary units to which each hidden unit is attached, or to associate groups of auxiliary units with groups of hidden units. This would reduce considerably the computational requirements for large problems but still enable substantial reductions in network size. It is also possible to use other types of unit-attracting terms, or even to implement the method by successively eliminating the smallest components of each vector $\vec{\omega}(s_i)$ until only one component remains on each auxiliary node.

Acknowledgments

I would like to thank P. Grassberger for helpful discussions. The partial support from the *Sonderforschungsbereich 262* and the Schwerpunkt Schi 257/2-2 is gratefully acknowledged.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," pages 318–362 in *Parallel Distrubuted Processing*, Volume 1, edited by D. E. Rumelhart and J. L. McClelland (Cambridge, MIT Press, 1985).

[2] J. Denker, D. Schwartz, B. Wittner, S. Solla, J. Hopfield, R. Howard, and L. Jackel, "Large Automatic Learning, Rule Extraction, and Generalization," *Complex Systems*, **1** (1987) 877–922.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Occam's Razor," *Information Processing Letters*, **24** (1987) 377–380.

[4] E. B. Baum and D. Haussler, "What Size Net Gives Valid Generalization?" *Neural Computation*, **1** (1989) 151–160.

[5] S. J. Hanson and L. Y. Pratt, "Comparing Biases for Minimal Network Construction with Back-Propagation," pages 177–185 in *Advances in Neural Information Processing Systems 1 (NIPS 88)*, edited by D. S. Touretzky (San Mateo, CA, Morgan Kaufmann, 1989).

[6] Y. Chauvin, "Generalization Performance of Overtrained Back-Propagation Networks," pages 46–55 in *Neural Networks. Proceedings of the EURASIP Workshop, Portugal 1990*, edited by L. B. Almeida and C. J. Wellekens (New York, Springer, 1990).

[7] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, in *Proceedings of the 1990 Connectionist Models Summer School*, edited by D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (San Mateo, CA, Morgan Kaufmann, 1990).

[8] E. D. Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks," *IEEE Transactions on Neural Networks*, **1** (1990) 239–242.

[9] M. Marchand, M. Golea, and P. Rujan, "A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons," *Europhysics Letters*, **11** (1990) 487–492.

[10] M. Mezard and J. P. Nadal, "Learning in Feedforward Networks: The Tiling Algorithm," *Journal of Physics A*, **22** (1989) 2191–2203.

[11] J. P. Nadal, "Study of a Growth Algorithm for a Feedforward Network," *International Journal of Neural Systems*, **1** (1989) 55–59.

[12] S. Harp, T. Samad, and A. Guha, "Towards the Genetic Synthesis of Neural Networks," pages 360–369 in *Proceedings of the International Conference on Genetic Algorithms*, edited by J. D. Schaffer (San Mateo, CA, Morgan Kauffman, 1989).

[13] G. Miller, P. Todd, and S. Hedge, "Designing Neural Networks Using Genetic Algorithms," pages 379–384 in *Proceedings of the International Conference on Genetic Algorithms*, edited by J. D. Schaffer (San Mateo, CA, Morgan Kauffman, 1989).

[14] H. Kitano, "Designing Neural Networks Using Genetic Algorithms with Graph Generation Systems," *Complex Systems*, **4** (1990) 461–476.

[15] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Using Genetic Search to Exploit the Emergent Behavior of Neural Networks," *Physica D*, **42** (1990) 244–248.

[16] T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, **1** (1987) 145–168.

[17] G. Tesauro and B. Janssens, "Scaling Relationships in Back-propagation Learning," *Complex Systems*, **2** (1988) 39–44.