Letters

A Simple Procedure for Pruning Back-Propagation Trained Neural Networks

EHUD D. KARNIN

Abstract—**One possible method of obtaining a neural network of an appropriate size for a particular problem is to start with a larger net, then prune it to the desired size. Training and retraining the net under all possible subsets of the set of synapses will result in a prohibitively long learning process; hence some methods that avoid this exhaustive search have been proposed. Here we estimate the *sensitivity* of the global error (cost) function to the inclusion/exclusion of each synapse in the artificial neural network. We do it by introducing "shadow arrays" that keep track of the incremental changes to the synaptic weights during (a single pass of) back-propagating learning. The synapses are then ordered by decreasing sensitivity numbers so that the network can be efficiently pruned by discarding the last items of the sorted list. Unlike previous approaches this simple procedure does not require a modification of the cost function, does not interfere with the learning process, and demands a negligible computational overhead.**

Introduction and Background

When constructing an artificial neural network the designer is often faced with the problem of choosing a network of the right size for the task. In theory at least, if a problem is solvable with a network of a given size, it can also be solved by a larger net which imbeds the smaller one, with (hopefully) all the redundant connections, or synapses, having a zero strength. However, the learning algorithm will typically produce a different structure, with nonvanishing synaptic weights spreading all over the net, thus obscuring the existence of a smaller size neural net solution.

We reiterate the advantages of using the smaller network:

- The cost of computation, measured by the number of arithmetic operations, grows (almost) linearly with the number of the synaptic connections. Hence a smaller net is more efficient in both forward computations and learning.
- Neural net learning is usually based on a finite (often small) set of training patterns. A network which is too large will tend to memorize the training patterns and thus have a poor generalization ability. (This phenomenon is also known in classification theory as "tuning to the noise," and it occurs whenever the number of the free parameters of the classifier is large relative to the training data.)
- There is always the hope that a smaller net will exhibit a behavior that can be described by a simple set of rules. One such example is given in [1], where a paradigm for the solutions to the *parity problem*, which was discovered by the learning system, is described.

Nevertheless, one would rather overestimate the network size than underestimate it. A network which is too small may never solve the problem (a simple example being the XOR function which cannot be implemented with a single neuron, see, e.g., [1]), while a larger net may even have the advantage of a faster learning rate. Thus it makes sense to start with a large net and then reduce its size. Several researchers have proposed methods to accomplish this reduction, and the same approach, i.e., *net pruning*, is pursued here too.

Manuscript received January 18, 1990.

The author is with IBM Science and Technology, Technion City, Haifa, 32000, Israel.

IEEE Log Number 9035521.

Before presenting the methods we introduce some notation, Let o_{pi} denote the output of unit (neuron) i upon presentation of pattern p. Unit j is connected to unit i via a synapse of strength w_{ij}, and the ith unit computes

$$o_{pi} = f\left(\sum_j w_{ij} o_{pj}\right). \tag{1}$$

The activation function f most frequently used is the logistic function, or its symmetric version

$$f(x) = \tanh(x) \tag{2}$$

that we will use here. The desired output for unit i, upon presentation of pattern p, will be denoted by t_i. The global error of the net is defined as

$$E = \sum_p \sum_i (o_{pi} - t_{pi})^2 \tag{3}$$

where the inner summation is over all neurons that are considered as output units of the net, and the outer sum is over patterns of the training set.

From (1) and (3) it becomes apparent that for a given training set, E is a function of all w_{ij} weights. We refer to *learning* as the process of modifying the weights in order to decrease E (and hopefully to achieve its minimum with respect to w_{ij}). The celebrated back-propagation learning algorithm updates the weights after each presentation of a subset of the training patterns (where the subset may range from a single pattern to the whole set) according to

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{4}$$

for some gain factor η. This technique is a variant of the steepest descent optimization method, see, e.g., [2]. In the case of layered feedforward networks the partial derivatives in (4) are computed by the chain rule, leading to the *generalized delta rule*, as described in [1].

Assuming that the network has learned, then an arbitrary setting of a w_{ij} weight to zero (which is equivalent to eliminating the synapse that goes from neuron j to neuron i), will typically result in an increase of the error E. In view of the discussion above, efficient *pruning* means finding the subset of weights that, when set to zero, will lead to the smallest increase in E.

Theoretically, this can be done by training the net under all possible subsets of the set of synapses. However, this exhaustive search is computationally infeasible, unless dealing with a very small net and few training patterns. Sietsma and Dow [3] have analyzed pruning of such networks by examining all units under the presentation of the entire training data. They removed each unit (along with its synaptic connections) that did not change state, or replicated another unit. Thus they found a subset of the net which has the same performance (i.e., no increase of E) as the complete network, for the given training data. Unfortunately, scaling up of this technique to large nets and many patterns will result is a prohibitively long learning process.

Several researchers have studied the idea of *weight decay* (see, e.g., [4]). The synaptic weight update of (4) is modified to

$$w_{ij}(n+1) = -\eta \frac{\partial E}{\partial w_{ij}}(n) + \beta w_{ij}(n) \tag{5}$$

for some positive $\beta < 1$. The idea is that weights that do not have much influence on decreasing the error while learning, i.e., have

$\partial E/\partial w_{ij} \simeq 0$, will experience an exponential time decay:

$$w_{ij}(n) \simeq \beta^n w_{ij}(0).$$

Weight decay is equivalent to adding a penalty term to the error function, changing it into

$$E' = E + \beta' \sum_i \sum_j w_{ij}^2 \tag{6}$$

where E was defined in (3). Thus the global cost E' is a weighted sum of two terms, with the relative weighting β' being yet another free parameter to be determined. Hanson and Pratt [4] have extensively experimented with various forms of weight decay, and concluded that these penalty terms do not seem to minimize the number of units.

Mozer and Smolensky [5] have introduced the idea of estimating the sensitivity of the error function to the elimination of each unit. In terms of (possible) connection elimination S_{ij}, the sensitivity with respect to w_{ij} will be defined here as

$$S_{ij} = E(w_{ij} = 0) - E(w_{ij} = w_{ij}^f) \tag{7}$$

where w_{ij}^f is the (final) value of the connection upon the completion of the training phase. In [5] S_{ij} is estimated by

$$\hat{S}_{ij} = -\left.\frac{\partial E}{\partial w_{ij}}\right|_{w_{ij}^f} w_{ij}^f.$$

The problem with this estimate, as pointed out in [5], is that the partial derivative tends to zero when the error decreases. While this is a desirable property of the gradient descent learning method, it results in poor sensitivity estimation. Mozer and Smolensky suggested changing the cost function from the quadratic measure of (3) into

$$E'' = \sum_p \sum_i |o_{pi} - t_{pi}|$$

and demonstrated the usefulness of their estimation procedure on some examples.

In the subsequent paragraphs we will pursue the idea of evaluating sensitivities for connections. However, a different approach for their estimation will be taken that does not require any change of the error function. Thus the two tasks of training the network and evaluating of candidates for pruning are completely separated, which means that the learning process is not interfered with by any extraneous process.

The Pruning Procedure

Our approach to pruning is to estimate the sensitivity of the error function to the exclusion of each synapse (connection), then prune the low sensitivity connections. The sensitivity S_{ij}, defined in (7), can be rewritten as

$$S = -\frac{E(w^f) - E(0)}{w^f - 0} w^f \tag{8}$$

where $w = w_{ij}$ and E is expressed as a function of w, assuming that all other weights are fixed (at their final states, upon completion of learning).

A typical learning process does not start with $w = 0$, but rather with some small (often randomly chosen) initial value w^i. Fig. 1 depicts an example of the decrease in E, as a function of the weight w, during the training phase.

Since we do not know $E(0)$, we will approximate the slope of $E(w)$ (when moving from 0 to w^f) by the average slope measured between w^i and w^f, namely

$$S \simeq -\frac{E(w^f) - E(w^i)}{w^f - w^i} w^f. \tag{9}$$

The initial and final weights, w^i and w^f, respectively, are quantities that are readily available during the training phase. However, for the numerator of (9) it was implicitly assumed that only one

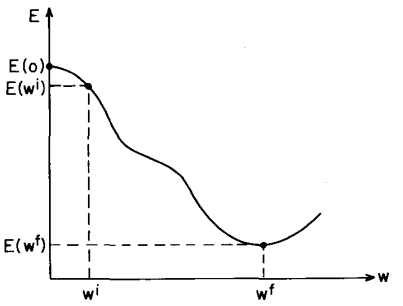

Fig. 1. The error as a function of one weight.

weight, namely w, had been changed, while all other weights remained fixed. This is clearly not the case during normal learning. To elaborate, consider the example of a network with only two weights, denoted u and w (the extension to more weights will become obvious). For this case the numerator in (9) is

$$E(u^f, w^f) - E(u^f, w^i) \tag{10}$$

i.e., only the contribution (or influence) due to the changes in w is taken into account. Fig. 2 clarifies the situation.

The error $E(u, w)$ is illustrated by constant value contours. The initial point in the weight space is designated by I in Fig. 2, and the learning path is the dashed line from I to F, the final point. For a precise evaluation of S, the numerator of (8) can be evaluated as

$$E(w = w^f) - E(w = 0) = \int_A^F \frac{\partial E(u^f, w)}{\partial w}\, dw.$$

The integral is along the line from point A, which corresponds to $w = 0$ (while all other weights are in their final states), to the final weight state F. However, the training phase starts at point I (rather than A), so we have to compromise on an approximation to integral above, namely, we will use

$$E(w = w^f) - E(w = 0) \simeq \int_I^F \frac{\partial E(u, w)}{\partial w}\, dw. \tag{11}$$

This expression will be further approximated by replacing the integral by summation, taken over the discrete steps that the network passes while learning. Thus the estimated sensitivity to the removal of connection w_{ij} (cf. (9) above) will be evaluated as

$$\hat{S}_{ij} = -\sum_0^{N-1} \frac{\partial E}{\partial w_{ij}}(n)\, \Delta w_{ij}(n)\, \frac{w_{ij}^f}{w_{ij}^f - w_{ij}^i} \tag{12}$$

where N is the number of training epochs.

The above estimate for the sensitivity uses terms that are readily available during the normal course of training. (Indeed, this was the main motivation for deriving such an approximation). Obviously the weight increments w_{ij} are the essence of every learning process, so they are always available. Also, virtually every optimization search (e.g., steepest descent, conjugate gradient, Newton's method) uses gradients to find the direction of change, so the partial derivatives $\partial E/\partial w_{ij}$, which are the components of the gradient, are available (cf. [2]). Therefore, the only extra computational demand for implementing our procedure is the summation in (12). This (negligible) overhead merely calls for maintaining a "shadow array" (of the same size as the number of connections in the network) that keeps track of the accumulated terms that build up S_{ij} in (12).

For the special case of back-propagation, weights are updated according to (4), hence (12) reduces to

$$\hat{S}_{ij} = \sum_0^{N-1} [\Delta w_{ij}(n)]^2 \frac{w_{ij}^f}{\eta(w_{ij}^f - w_{ij}^i)}. \tag{13}$$

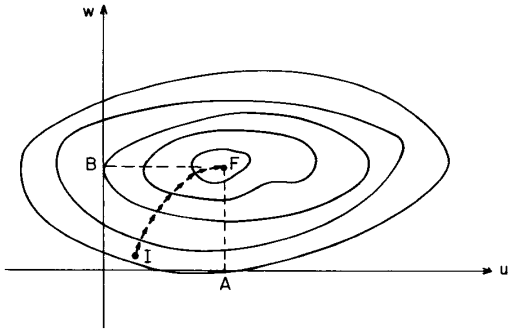

Fig. 2. Learning as motion on an error function surface.

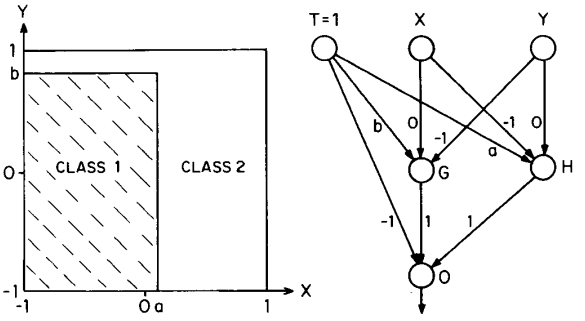

Fig. 3. A pattern classification problem.

(For back-propagation with momentum (cf. [1]), the general form of (12) should be used.)

Upon completion of training we are equipped with a list of sensitivity numbers, one per each connection. They were created by a process that runs concurrently, but without interfering, with the learning process. At this point a decision can be taken on pruning the synapses of the smallest sensitivity numbers, as demonstrated in the next section.

Some Examples

Sample problems of sensitivity estimation and its application to network pruning are discussed below. We have initialized the network with small random weights, then applied back-propagation (with momentum) for supervised learning of a training sequence. The output of each unit, as well as the inputs to the net, were in the range $[-1, 1]$, and the activation function of (3) was used.

Example 1

Consider the pattern classification problem in the two-dimensional feature space depicted in Fig. 3. The feature vector (X, Y) is uniformly distributed in $[-1, 1] \times [-1, 1]$, and the two nonoverlapping classes are equally probable (i.e., $b = 2/(a + 1) - 1$, see Fig. 3).

The first task is to train a neural net classifier to recognize the patterns in class 1 (i.e., the output should be 1 for patterns in class 1 and -1 for class 2). An "ideal" two input, two hidden units, and one output network (a 2-2-1 net, for short) is shown in Fig. 3. Upper case characters denote units (or neurons), where T stands for the constant 1 input, used to set the threshold for all internal and output units.

We have examined a case study with $0 < a \ll 1$, where it is clear that X is a more important feature than Y, as far as a minimum decision error is desired. Hence we would expect the sensitivity analysis to indicate this fact.

For the particular case of $a = 0.1$, one experiment yielded the following weights (after training with $\eta = 0.05$, momentum coefficient $\alpha = 0.5$, weights update every 24 presentations of random patterns, and 10 000 training epochs):

	First Layer Weights		
	T	X	Y
G	2.96	0.57	−4.43
H	1.56	−7.39	0.80

	Second Layer Weights		
	T	G	H
0	−4.32	5.44	5.31

Similar results were obtained by other experiments. One hidden unit H roughly learned to produce the decision line $X = a$, while the decision of the other hidden unit was mainly based on Y. Now, suppose we would like to eliminate one of the hidden units by pruning its connection to the output. The sensitivities we obtained are:

	Sensitivities		
	T	G	H
0	0.67	0.15	0.33

The connection from G to the output has the lower sensitivity, hence w_{OG} is to be pruned. As expected, the result is a network that produces a decision boundary which is roughly a vertical line through $X = a$ in Fig. 3.

Example 2

This example is adopted from [5], who termed it "the rule-plus-exception problem." They used it to demonstrate that their estimation method fails to find correct sensitivities when used in conjunction with the quadratic error function, while being successful with the linear cost function. We will demonstrate that our method can live with the quadratic function.

The problem is to learn the Boolean function $AB + \bar{A}\bar{B}\bar{C}\bar{D}$. A four input, two hidden units, and one output network (4-2-1 net) will be trained to learn this function. The output unit should be on (i.e., equals 1) when both A and B are on, which is the "rule." It should also be on when the "exception" $\bar{A}\bar{B}\bar{C}\bar{D}$ occurs. Clearly, the "rule" is more important than the "exception," since by itself it accounts for 15 (out of 16) correct decisions. A proper sensitivity analysis should reveal this fact, and helps prune the net accordingly.

The architecture of the network is illustrated in Fig. 4. For the example shown we have trained the net with $\eta = 0.1$, momentum $\alpha = 0.8$, 3000 training epochs, each one consisting of the full 16 possible binary vectors. The resulted weights were:

	First Layer Weights				
	T	A	B	C	D
X	3.70	1.45	1.44	1.40	1.40
R	−2.66	3.45	3.41	0.56	0.56

	Second Layer Weights		
	T	X	R
0	3.55	−4.98	4.57

It is apparent that the unit R roughly computes AB, the "rule" (notice the small weights w_{RC}, w_{RD} compared to w_{RA}, w_{RB}), while unit X computes $\bar{A}\bar{B}\bar{C}\bar{D}$, the "exception." The sensitivities that

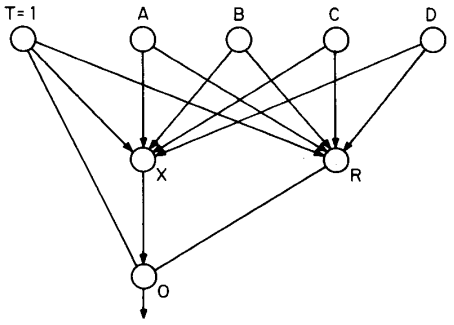

Fig. 4. A network for the "rule-plus-exception" problem.

were estimated during course of learning were:

First Layer Sensitivities					
	T	A	B	C	D
X	0.15	0.09	0.05	0.02	0.02
R	0.16	0.42	0.27	0.06	0.03

Second Layer Sensitivities			
	T	X	R
0	0.39	0.22	0.72

If one hidden unit has to be removed, we have to compare S_{OX} to S_{OR}, to realize that unit X, the "exception" is the one to be eliminated. Further, we look at the sensitivities of the connection from the inputs to the remaining unit R. Among them S_{RC} and S_{RD} are the smallest, so w_{RC} and w_{RD} can be pruned. This leaves a small size network that can compute the "rule" (AB) only, and it is the best two input approximation to the problem at hand.

Summary and Comments

We have devised a simple procedure that takes advantage of pieces of data (namely, the gradient and increments to the weights) that are available during the normal course of neural net training. Learning remains intact, and the sensitivity of the (unchanged) error function to the elimination of each synapse is concurrently evaluated by a "shadow process" that demands only a negligible computational overhead. A decision on the connections that should be pruned is made only after the completion of the training phase.

The advantages of this method over previous approaches have been described and also demonstrated by examples. However, one should not overlook the fact that this procedure, like previous approaches, is only a heuristic that provides some reasonable estimates of the true sensitivities (as defined in (7) above). As we explained in the derivation of (12), we are restricted to the learning path (the dashed line in Fig. 2), otherwise we have to specially train and retrain for each synapse that is a candidate for elimination. This would result in a prohibitively long training which our simple procedure avoids.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1, D. E. Rumelhart and J. L. McCleland, Eds. Cambridge, MA: M.I.T. Press, 1986, pp. 318–362.

[2] D. G. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984.

[3] J. Sietsma and R. J. F. Dow, "Neural net pruning—Why and how?," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1 (San Diego, CA), 1988, pp. 325–332.

[4] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in Neural Information Processing I*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 177–185.

[5] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in Neural Information Processing I*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 107–115.

Neural Networks for Control Systems

PANOS J. ANTSAKLIS, SENIOR MEMBER, IEEE

***Abstract*—This letter describes 11 papers from the April 1990 Special Issue of the IEEE CONTROL SYSTEMS MAGAZINE on Neural Networks in Control Systems.**

Introduction

Ever-increasing technological demands of our modern society require innovative approaches to highly demanding control problems. Artificial neural networks with their massive parallelism and their learning capabilities offer the promise of better solutions, at least to some problems. By now, the control community has heard of neural networks and wonders if these neural networks can be used to provide better control solutions to old problems or perhaps solutions to control problems which have withstood its best efforts.

Control Technology Demands

The use of neural networks in control systems can be seen as a natural step in the evolution of control methodology to meet new challenges. Looking back, the evolution in the control area has been fueled by three major needs: the need to deal with increasingly complex systems, the need to accomplish increasingly demanding design requirements, and the need to attain these requirements with less precise advanced knowledge of the plant and its environment; that is, the need to control under increased uncertainty. Today the need to better control increasingly complex dynamical systems under significant uncertainty has led to a reevaluation of the conventional control methods, and it has made apparent the need for new methods. It has also led to a more general concept of control, one which includes higher level decision making, planning, and learning, which are capabilities necessary when higher degrees of system autonomy are desirable. These ideas are elaborated upon in [1]. In view of this, it is not surprising that the control community is seriously and actively searching for ideas to deal effectively with the increasingly challenging control problems of our modern society. Need is the mother of invention and this has been true in control since the times of Ktesibios and his water clock with its feedback mechanism in the third century B.C.[2], the earliest feedback device on record. So the use of neural networks in control is rather a natural step in its evolution. Neural networks appear to offer new, promising directions toward better understanding and perhaps even solving some of the most difficult control problems. History, of course, has made clear that neural

Manuscript received December 19, 1989.

The author is with the Department of Electrical and Computer Engineering, University of Notre Dame, Notre Dame, IN 46556.

IEEE Log Number 9034405.