On Generalization By Neural Networks

Subhash C. Kak
Department of Electrical & Computer Engineering
Louisiana State University
Baton Rouge, LA 70803-5901
Email: kak@ee.lsu.edu

March 24, 1998

Abstract

We report new results on the corner classification approach to training feedforward neural networks. It is shown that a prescriptive learning procedure where the weights are simply read off based on the training data can provide good generalization. The paper also deals with the relations between the number of separable regions and the size of the training set for a binary data network. Prescriptive learning can be particularly valuable for real-time applications.

1 Introduction

A new approach to training feedforward neural networks for binary data was proposed by the author [2, 3]. This is based on a new architecture that depends on the nature of the data. It was shown that this approach is much faster than backpropagation and provides good generalization. This approach, which is an example of prescriptive learning, trains the network by isolating the corner in the n-dimensional cube of the inputs represented by the input vector being learnt. Several algorithms to train the new feedforward network were presented. These algorithms were of three kinds. In the first of these (CC1) the weights were obtained upon the use of the perceptron algorithm. In the second (CC2), the weights were obtained by inspection from the data, but this did not provide generalization. In the third (CC3), the weights obtained by the second method were modified in a variety of ways that amounted to randomization and which now provided generalization. During such randomization some of the learnt patterns could be misclassified; further checking and adjustment of the weights was, therefore, necessitated. Various comparisons were reported in [6, 9, 7]. The comparisons showed that the new technique could be 200 times faster than the fastest version of the backpropagation algorithm with excellent generalization performance.

In this article we show how generalization can be obtained for such binary networks just by inspection. We present a modification to the second method so that it does provide generalization. This technique's generalization might not be as good as when further adjustments are made, but the loss in performance could, in certain situations, be more than compensated by the advantage accruing from the instantaneous training which makes it possible to have as large a network as one pleases. Experimental results in support of our method are presented.

Hidden Neurons

It is well known [8, 1] that the number of hidden neurons, H, needed to separate M number of regions in a $d-$dimensional space is given by

$$M(H,d) = \sum_{k=0}^{d} \binom{H}{k} \tag{1}$$

where

$$\binom{H}{k} = 0, \;\; H < k.$$

Let the number of regions that the hidden neurons separate be equal to C, where $C \leq M$. Since the number of classes at the output is only equal to 2, these C regions coalesce into the 2 classes at the output.

Let the input space dimension be d and let each dimension be quantized so that the total number of of binary variables is n. Not each dimension may require the same precision. If the average number of bits used per input dimension is q then $n = q \times d$.

If the number of training samples is T, then $M \leq T$. For $d = 1$, $H = M - 1$, for $d = 2$, $H = (\sqrt{(8M-7)} - 1)/2$, and for $d \geq H$, $H = log_2 M$.

When the data points are binary, as in our case, then these formulas require modification. The set of 2^n data points can now be separated by a total of n hidden neurons. But the outputs of these hidden neurons need to be combined using various logical operations to pass specific input patterns. This is not a desirable strategy to adopt if the learning is supposed to be decentralized with a cumulative response to all the training data.

Our network has H nearly equal to T (or M), therefore, our algorithms consider the data as effectively one-dimensional.

2 Prescriptive Training

We assume that the reader is familiar with the background papers [2, 3]. We consider the mapping $Y = f(X)$, where X and Y are $n-$ and $m-$dimensional binary vectors. But for convenience of presentation, it will be assumed that the output is a scalar, or $m = 1$. Once we know how a certain ouput bit is obtained, other such bits can be obtained similarly. We consider binary neurons that ouput 1 if and only if the sum of the inputs exceeds zero. To provide for effective non-zero thresholds to the neurons of the hidden layer an extra input $x_{n+1} = 1$ is assumed. In the earlier formulations of the rule we took the weights in the output layer all equal 1. Kun Won Tang has suggested that it is much better to take the weights as equal to 1 if the output value is 1 and -1 if the output value is 0. This amounts to learning both the "1" and the "0" output classes.

A hidden neuron is required for an input vector for each training sample, so that one might say that the hidden neuron 'recognizes' the training vector. Consider such a vector for which the number of 1's is s; in other words, $\sum_{i=1}^{n} x_i = s$. The weights leading from the input neurons to the hidden neurons are:

$$w_j = \begin{cases} h & if \;\; x_j = 0, \;\; for \;\; j = 1, ..., n, \\ +1 & if \;\; x_j = 1, \;\; for \;\; j = 1, ..., n, \\ r - s + 1 & for \;\; j = n + 1. \end{cases} \tag{2}$$

The values of h and r are chosen in various ways. This is a generalization of the expression in [2, 3] where w_j for $j = n + 1$ is taken to be $(r - s + 1)$ rather than $(1 - s)$, and where $h = -1$.

This change allows the learning of the given training vector as well as others that are at a distance of r units from it (for $h = -1$); in other words, r is the *radius of the generalized region.* This may be seen by considering the all zero input vector. For this $w_{n+1} = r$. Since, all the other weights are -1 each, one can at most have $(r-1)$ different $+1$s in the input vector for it to be recognized by this hidden neuron.

Choice of r

The choice of r will depend upon the nature of generalization sought.

If no generalization is needed then $r = 0$. For exemplar patterns, the choice of r defines the degree of error correction.

If the neural network is being used for function mapping, where the input vectors are equally distributed into the 0 and the 1 classes, then $r = \lfloor \frac{n}{2} \rfloor$. This represents the upper bound on r for a symmetric problem.

But the choice will also depend on the number of training samples.

Choice of h

The choice of h also influences the nature of generalization. Increasing h from the value of -1 correlates patterns within a certain radius of the learnt sequence.

This may be seen most clearly by considering a $2-$dimensional problem. The function of the hidden node can be expressed by the separating line:

$$w_1 x_1 + w_2 x_2 + (r - s + 1) = 0. \tag{3}$$

This means that

$$x_2 = \frac{-w_1}{w_2} x_1 + \frac{-(r - s + 1)}{w_2}. \tag{4}$$

Assume that the input pattern being classified is (0 1), then $x_2 = 1$. Also, $w_1 = h$, $w_2 = 1$, and $s = 1$. The equation of the dividing line represented by the hidden node now becomes:

$$x_2 = -h x_1 - r. \tag{5}$$

When $h = -1$ and $r = 0$, the slope of the line is positive and only the point $(0, 1)$ is separated. To include more points in the learning, $h > 0$, because the slope of the line becomes negative.

Relationship between h and r

Consider the all zero sequence (0 0 ... 0). After the appending of the 1 threshold input, we have the corresponding weights ($h\ h$ ... $h\ r{+}1$). Sequences at the radius of p from it will yield the strength of $ph + r + 1$ at the input of the corresponding hidden neuron. For such signals to pass through

$$ph + r + 1 > 0. \tag{6}$$

In other words, generalization by a Hamming distance of p units is achieved if

$$h > \frac{-(r+1)}{p}. \tag{7}$$

When $h = -1$; $p < r + 1$, or it may be taken to be equal to r. When $h = \mathit{positive}$, all the input patterns where the $0s$ have been changed into $1s$ will also be passed through and put in the same class as the training sample.

The network architecture for learning the XOR problem is given in Figure 1; this has four hidden neurons, one for each output. An example of a network which maps three 5-component input vectors into 2-component output vectors is given in Figure 2.

Figur3 provides the genralization obtained by using our method for training a 'spiral.' Notice that for $r = 4$ we have overgeneralization.

3 Training samples

The total number of sequences 2^n equals the number of classification classes M times the average number of members in each class.

Let the radius of the class i be r_i. The number of elements in this class will be

$$\sum_{k=0}^{r_i} \binom{n}{k}. \tag{8}$$

If all the classes are of the same size and each class is represented by a single training sample:

$$T \times \sum_{k=0}^{r} \binom{n}{k} = 2^n. \tag{9}$$

The following table gives the size of the training set for the example of $n = 10; 2^n = 1,024$.

Table I: Generalization and training set size

r	T
0	1,024
1	32
2	12
3	9

Since the probability that each training sample belongs to a different class could be small, the above numbers represent very rough estimates.

3.1 Experimental Results

Experiments were conducted on several kinds of time-series. Part of the time series was used for finding the weights; the rest was used to test the model. A window of w preceding points was used to predict the next point in the time series. All the analog values were quantized. In a variation of this method the weights were updated further as new data came in. For the time-series considered the best results were obtained when the radius of generalization r was about the same value as the window size w or, in other words, $r \approx w$.

Results on prediction for the Mackey-Glass (MG) time series are presented in Figure 4. The MG series is a commonly used benchmark because it is a chaotic time series which represents a difficult case for prediction. The data generated by the MG equation mimicks the nonlinear oscillations in physiological processes. The discrete time representation of the MG equation is given by

$$x(k+1) - x(k) = \frac{\alpha x(k-\tau)}{1 + x^{\gamma}(k-\tau)} - \beta x(k) \tag{10}$$

Another way to improve generalization is by varying the radius of generalization with the training sample. This may be done easily if each training sample can be characterized by a measure of quality which may be possible to do for certain situations.

Figure 5 gives prediction for the logistic map time series given by

$$x(k+1) = 4x(k)(1-x(k)) \tag{11}$$

The training window size for the cases of the prediction in Figures 4 and 5 is very small and the quantization is course which is why the learning is not perfect for the training period. The performance gets better as the quantization levels are increased and the window size gets larger. The results for such cases as well as techniques of supervised tuning will be published elsewhere.

4 Concluding Remarks

The generalized prescriptive rule presented in this paper makes it possible to train a neural network instantaneously. There is no need for any computations in determining the weights. This allows the building up of neural networks of unlimited size.

It may be useful to use a two-step strategy when the learning method described in this article is used. In the first step use a separate network to determine the mode of the data. For example, for stock market data one may define the outputs (1 0), (0 1) for bear and bull markets, respectively; and (0 0), (1 1) for two different kinds of stagnating markets. Once, the determination of the type of market has been made, the data can be used on specific networks, trained on that kind of data, to make the prediction. The logic behind this two-step approach is that, lacking any supervisory training, our networks are good only in separating a single class at any particular time.

But the strategy of two-step response is also biologically motivated. The brain reorganizes itself in response to an input; likewise, the artificial neural network chosen to deal with an input is based on the nature of the input.

The instantaneous method of neural network training described in this paper could be the method at work in biological systems. This learning could form part of a hierarchy of different languages of the brain [4, 5].

Acknowledgement

The author is grateful to Kun Won Tang for performing the experiments. This research was partly supported by NASA.

References

[1] G. Georgiou, Comments on ‘On Hidden Nodes.’ *IEEE Trans. on Circuits and Systems* 38, 1410 (1991).

[2] S.C. Kak, On training feedforward neural networks. *Pramana -J. of Physics*, 40, 35-42 (1993).

[3] S.C. Kak, New algorithms for training feedforward neural networks. *Pattern Recognition Letters*, 15, 295-298 (1994).

[4] S.C. Kak, Quantum neural computing. *Advances in Imaging and Electron Physics*, 94, 259-313 (1995).

[5] S.C. Kak, The three languages of the brain: quantum, reorganizational, and associative. In *Learning as Self-Organization*, K. Pribram and J. King (eds.). Lawrence Erlbaum, Mahwah, 1996, pp. 185-219.

[6] S.C. Kak and J. Pastor, Neural networks and methods for training neural networks. *U.S. Patent No. 5,426,721*, June 20, 1995.

[7] K.B. Madineni, Two corner classification algorithms for training the Kak feedforward neural network. *Information Sciences* 81, 229-234 (1994).

[8] G. Mirchandani and W. Cao, On hidden nodes for neural nets. *IEEE Trans. on Circuits and Systems* 36, 661-664 (1989).

[9] P. Raina, Comparison of learning and generalization capabilities of the Kak and the backpropagation algorithms. *Information Sciences* 81, 261-274 (1994).

Figure captions

Figure 1. Network architecture for learning the XOR function

Figure 2. Network architecture for the example below

Figure 3. Pattern classification for different values of r

Figure 4. Prediction of Mackey-Glass time series: "*" actual data, training to 379, testing 380-400 (marked with "o"); Window size=4, r=4, predict ahead by 1 point

Figure 5. Prediction of Logistic Map chaotic time series (seed=0.9): "*" actual data, training to 379, testing 380-400 (marked with "o"); Window size=4, r=4, predict ahead by 1 point