PRAMANA
— journal of
physics

Vol. 40, No. 1,
January 1993
pp. 35–42

On training feedforward neural networks

SUBHASH KAK
Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803-5901, USA

MS received 16 July 1992; revised 18 September 1992

Abstract. A new algorithm that maps n-dimensional binary vectors into m-dimensional binary vectors using 3-layered feedforward neural networks is described. The algorithm is based on a representation of the mapping in terms of the corners of the n-dimensional signal cube. The weights to the hidden layer are found by a corner classification algorithm and the weights to the output layer are all equal to 1. Two corner classification algorithms are described. The first one is based on the perceptron algorithm and it performs generalization. The computing power of this algorithm may be gauged from the example that the exclusive-Or problem that requires several thousand iterative steps using the backpropagation algorithm was solved in 8 steps. Another corner classification algorithm presented in this paper does not require any computations to find the weights. However, in its basic form it does not perform generalization.

Keywords. Neural networks; nonlinear systems.

PACS Nos 87·30; 64·60; 75·10

1. Introduction

Feedforward neural networks map nonlinear transformations of importance in biophysics and computer science. The backpropagation algorithm [1] is used to train these networks but this algorithm is very slow and it does not guarantee convergence [2]. This paper describes a new efficient training algorithm for feedforward neural networks which is guaranteed to find a solution. We will develop a constructive procedure to realize a general mapping using a 3-layered feedforward network of binary neurons, where the input layer merely distributes data. This procedure also provides an upper bound on the number of hidden neurons; the number of input and the output neurons being n and m respectively.

2. The notation

Consider the mapping $Y = f(X)$, where X and Y are n and m dimensional binary vectors. Let the number of vectors be equal to k, so we can also write $Y^i = f(X^i)$, $i = 1,\ldots,k$.

The output of each neuron is 1 depending on whether the weighted sum of the signal to the neuron is greater than its threshold; otherwise it is 0. Mathematically, the output of the neuron j, o_j, is given by the formula:

$$o_j = \begin{cases} 1 \text{ if } \sum w_{ij} x_i > T \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

For convenience we will add an extra constant value input to each neuron and replace the threshold by the value zero. The weight of this additional input would then simulate any specific threshold value. If no threshold is specified it will be taken as zero. [For some limitations of this model see [3]].

If the k output sequences are written out in an array then the columns may be viewed as a sequence of m, k-dimensional vectors W_i.

DEFINITION

The weight of a binary vector is the number of elements of value 1 in it.

Note that the term weight is also used in a different sense later in the paper where it represents the strength of the synaptic interconnection.

DEFINITION

Let the weight of Y^i be α_i, of X^i be s_i, and let the weight of the vector $W_j = (y_{1j} y_{2j} \cdots y_{kj})$ be β_j.

3. The main algorithm

Consider the output y_1 for convenience; this output defines the vector W_1 when we run through all the input values. We will now show how this output can be obtained by a 3-layered network. Use of m such units will realize the entire mapping.

The y_1 values for the k samples represent the W_1 vector. There are exactly β_1 elements of value 1 in it. The vector W_1 can be written as a sum of β_1 vectors each one of which has a single 1 in it. This is the key step of the proposed algorithm. We wish to classify the input n-dimensional vectors that produce 1 individually. This should be contrasted with the perceptron algorithm [4, 2] where all the vectors that produce 1 are lumped together.

Now we need a single neuron to realize each of these vectors since the input X^i vectors constitute the corners of a n-dimensional cube and isolating precisely one of these corners is guaranteed upon the use of an appropriate hyperplane. In other words, a total of β_1 neurons are needed to isolate the β_1 corners corresponding to the vector W_1. These β_1 neurons in the hidden layer, connected to the n input lines through weights that define the correct hyperplanes, produce the output of 1 for the specified input vectors. Now the output of these neurons should go through a logical-Or operation to produce the correct response at the output layer neuron. Therefore, the hidden neuron outputs may be fed in the output layer to a single neuron, through links each with a weight of 1. The threshold of the output neuron should be less than 1 and greater than or equal to 0 to ensure a 1 for all the corners that were isolated by the β_1 hidden layer neurons. This is because a neuron produces an output of 1 only when it exceeds the threshold (equation 1). Although a zero threshold should suffice, we may do this by the use of a threshold which equals $1/\beta_1$, so as to be able to deal with the presence of noise on the links.

This may be done for each of the m output neurons. The corner classification procedure used in our structured method establishes that the realization of the mapping is guaranteed. The design of the feedforward network is as shown in figure 1.

But many of the hidden neurons in the design of figure 1 are duplicates. Consider

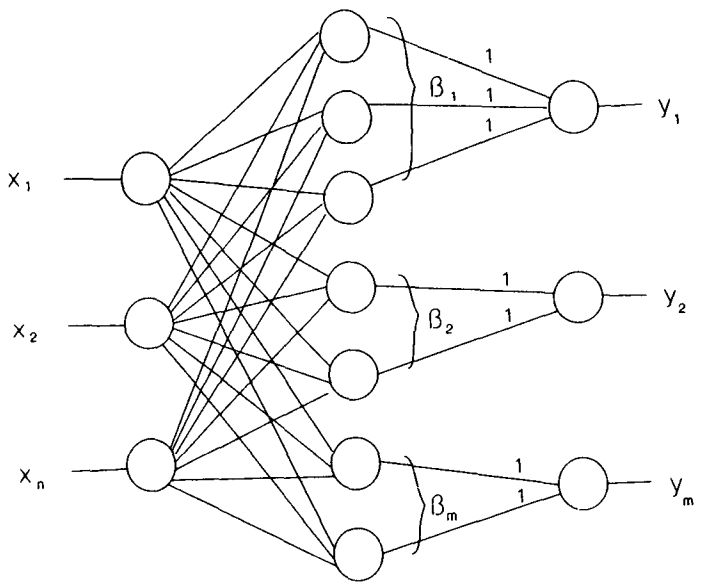

Figure 1. The general network design.

now the following functions:

$$\gamma_i = \begin{cases} \alpha_i - 1 \text{ if } \alpha_i \geqslant 1 \\ \alpha_i \text{ otherwise} \end{cases} \tag{2}$$

The number of hidden neurons can be reduced by the duplicating neurons that equal $\Sigma_{i=0}^{i=k} \gamma_i$. This result is summarized in the following theorem.

Theorem. *The number of hidden neurons required to realize the mapping* $Y^i = X^i$, $i = 1, 2, \ldots, k$ *is equal to* $\Sigma_{j=1}^{m} \beta_j - \Sigma_{i=1}^{k} \gamma_i$.

Since $\Sigma_{j=1}^{m} \beta_j = \Sigma_{i=1}^{k} \alpha_i$, we have the following corollary:

COROLLARY

The number of hidden neurons required to realize a mapping is at most k.

The next sections present two corner classification algorithms to implement the main algorithm.

4. Corner classification using perceptrons (CC1)

An algorithm to discover a successful set of weights to isolate a corner of the n-dimensional input vector cube can be devised in several ways. Remember that a single neuron is being used to isolate each corner. Whenever the neuron makes a mistake the weights are changed so as to make the mistake less likely. This is an application of the familiar perceptron algorithm and we know that it will converge.

Let the inputs and the weights be represented by the vectors $(x_1 x_2, \ldots, x_n)$ and $(w_1 w_2, \ldots, w_n)$. To learn the thresholds we assume an extra input with a constant value of 1. The algorithm may be restated as:

1. Start with a random initial weight vector. 2. If the neuron says no when it should say yes, add the input vector to the weight vector. 3. If the neuron says yes when it should say no, subtract the input vector from the weight vector. 4. Do nothing otherwise.

We now present a proof which is adapted from Papert [5].

Proof. Let W be the vector of weights and let X be the input vector. We would establish that the classification algorithm converges if there is a weight vector, W^+, such that $W^+ \cdot X > \delta$ for the corner that needs to be classified, and $W^+ \cdot X < -\delta$ otherwise.

Consider now the weight vector at the tth iteration, W_t. The cosine of the angle between W^+ and W_t is equal to the dot product of the two vectors divided by their lengths:

$$\cos\theta = \frac{W^+ \cdot W_t}{\| W^+ \| \, \| W_t \|}. \tag{3}$$

Consider an update of the weight vector when the neuron misclassifies. This would happen when $W_{t-1} \cdot X \leqslant 0$ when it should have been greater than zero and the other way round. Considering the first type of error leads to the following change to the weight vector:

$$W_t = W_{t-1} + X.$$

Thus the numerator of the cosine expression becomes

$$\begin{aligned} W^+ \cdot W_t &= W^+ \cdot (W_{t-1} + X), \\ &= W^+ \cdot W_{t-1} + W^+ \cdot X. \end{aligned}$$

Because W^+ always produces correct results, we know that $W^+ \cdot X > \delta$, given that the neuron is supposed to say yes. Thus, the numerator of the cosine expression may be rewritten as follows:

$$W^+ \cdot W_t > W^+ \cdot W_{t-1} + \delta.$$

We get the same inequality for the other type of misclassification. Repeating this process for t iterations leads to the following inequality for the cosine's numerator.

$$W^+ \cdot W_t > t\delta. \tag{4}$$

Now consider the cosine's denominator:

$$\| W^+ \| \, \| W_t \|.$$

Rewriting the square of W_t's length, $\| W_t \|^2$, as a dot product we have:

$$\begin{aligned} \| W_t \|^2 &= W_t \cdot W_t \\ &= \| W_{t-1} \|^2 + 2W_{t-1} \cdot X + \| X \|^2. \end{aligned}$$

Next, $W_{t-1} \cdot X$ must be 0 or less because the change was required by the neuron saying no when it should have said yes. Thus, we have:

$$\| W_t \|^2 \leqslant \| W_{t-1} \|^2 + \| X \|^2.$$

Exactly the same inequality is obtained for the other type of misclassification. Repeated substitution produces the following result:

$$\| W_t \|^2 \leqslant t \| X \|^2.$$

Since all the elements of X are either 0 or 1, $\| X \|^2$ cannot be larger than the weight of the input vectors, n. Its average value will be $(n+1)/2$. This yields:

$$\| W^+ \| \| W_t \| \leqslant \| W^+ \| \sqrt{t} \sqrt{n}. \tag{5}$$

Using (4) and (5) in (3), we have

$$\cos\theta > \frac{\delta\sqrt{t}}{\| W^+ \| \sqrt{n}}.$$

We see that the lower bound on the cosine of the angle between W^+ and W_t must increase with each change since it is proportional to $\sqrt{t}$. This establishes that the algorithm will converge.

It may be noted that the weights necessary to classify the corners may be precomputed for any problem of a fixed dimension and then used from a table look up. However, since the number of corners is 2^n and the corner classification algorithm is efficient, there is no need for this.

Example 1. The exclusive-Or mapping. A mapping such as exclusive-Or is not linearly separable, which is why it has served as one of the benchmarks to test training algorithms.

Sample	x_1	x_2	x_3	Output
1	0	0	1	0
2	0	1	1	1
3	1	0	1	1
4	1	1	1	0

This table has been made with $x_3 = 1$ to allow the hidden layer neurons to learn their thresholds.

For the exclusive-Or mapping $m = 1$, and $\Sigma\beta_i = 2$. So we know that we need 2 hidden layer neurons. Our training algorithm requires that the output vector (0 1 1 0) be split into two component vectors (0 1 0 0) and (0 0 1 0).

The input corners that we will now isolate are (0 1) and (1 0) respectively. Furthermore, since this mapping is a complete mapping, we will take the initial weights to be all zeros. Since the two corners are near the origin (0 0), the equations for the hyperplanes do not have to have a constant term in it. So for this case the weight component x_3 will be zero for each case, as we will see on the application of the corner classification algorithm. The first one implies realizing the transformation:

Sample	x_1	x_2	x_3	Output
1	0	0	1	0
2	0	1	1	1
3	1	0	1	0
4	1	1	1	0

This can be realized by the weights $(-1\ 1\ 0)$ obtained using the corner classification algorithm, which takes us through the sequence

$$\begin{matrix} 0\ 0\ 0 \\ 0\ 1\ 1 \\ -1\ 1\ 0 \end{matrix}$$

The weights are obtained in 2 steps. Note that these weights work because the threshold of zero for the neuron needs to be exceeded for the output 1. If one wished for the weights to be different so that there exists a separation between the neuron threshold and the actual signal input, one may continue with a few more passes at the samples with the classification algorithm.

Now consider the second basis transformation:

Sample	x_1	x_2	x_3	Output
1	0	0	1	0
2	0	1	1	0
3	1	0	1	1
4	1	1	1	0

Here the application of the corner classification algorithm gives us:

$$\begin{matrix} 0 & 0\ 0 \\ 1 & 0\ 1 \\ 0 & -1\ 0 \\ 1 & -1\ 1 \\ 0 & -2\ 0 \\ 1 & -2\ 1 \\ 1 & -2\ 0 \end{matrix}$$

We require a total of six iteration steps to obtain these weights. It so turns out that the thresholds of the hidden layer neurons for this example is zero since $w_3 = 0$ for each of the two neurons. Having obtained the weights of the input layer, the design is now complete since the weights in the output layer are all equal to 1. The threshold of the neuron in the output layer may be fixed at 1/2.

The total number of iterative steps required in this algorithm was 8. This may be contrasted with the 6,587 iterations required in the use of the backpropagation algorithm to solve the same problem [1].

5. A new corner classification algorithm (CC2)

An algorithm to discover a successful set of weights to isolate a corner of the n-dimensional input vector cube can be devised in several ways. In the previous section we used a perceptron algorithm to classify corners, and now we present a new formula for doing the same.

Consider that the input vector X^i needs to be isolated by a hidden neuron. We require the hidden neuron to produce a 1 corresponding to this input and a 0 corresponding to all other inputs. This can be achieved by use of weights that are positively correlated with X^i and negatively correlated with all other input vectors.

Consider that the inputs (that include an extra $x_{n+1} = 1$ to learn the thresholds) and the weights are represented by vectors $X^+ = (x_1 x_2, \ldots, x_n x_{n+1})$ and $W^+ = (w_1 w_2, \ldots, w_n w_{n+1})$ respectively. One requires to divide the input vectors into two cases to obtain appropriate classification. First, we assign $w_{n+1} = 1$ for the all-zero sequence. This is the least integer value of w_{n+1} since there are no non-zero values in this sequence to establish positive correlation. This choice now forces the w_{n+1} values for all other sequences to be zero or negative. For the other cases we assign $w_{n+1} = -(s_i - 1)$ where s_i is the weight of the sequence X^i; this ensures that the sequences with weights less than this are not classified. The weight of the all-zero sequence is 0, therefore the w_{n+1} required to classify it is $-(0-1)$ or 1 as chosen before. To establish negative correlation the weight w_i for $x_i = 0$ is taken to be -1. The increasingly more negative values of w_{n+1} for X^i's of larger weight defines hyperplanes that move further away from the origin. In other words the weights are given as follows:

$$w_j = \begin{cases} -1 & \text{if } x_j = 0 \\ 1 & \text{if } x_j = 1 \\ -(s_i - 1) & \text{for } j = n+1 \end{cases}$$

The value of $w_j = -(s_i - 1)$ implies that the threshold of the hidden neuron to separate this sequence is $(s_i - 1)$.

The above weights are just one set of solutions for the mapping problem. These weights represent the use of the smallest integer values.

Example 1. (*Contd.*) For the exclusive-Or mapping $m = 1$, and $\Sigma\beta_i = 2$. So we know that we need 2 hidden layer neurons.

Using our corner classification formula, the weights are the vectors $(-1\,1\,0)$ and $(1\,-1\,0)$. The w_3 being equal to 0 in each case implies that the threshold of both the hidden neurons is 0.

6. Generalization from training samples

One reason why neural networks have become popular is because they can generalize from training samples. Viewed in the space of input vectors, classification of each training sample also leads to a similar classification of its neighbours. Of the two corner classification algorithms presented in the paper, the first one (CC1) performs generalization, especially if the initial weights are randomly chosen. Since the algorithm CC2 has revealed that the weights can range over $(1, -n+1)$, we can pick numbers from this range as the initial weights to classify a corner.

The algorithm CC2 does not, in the basic form it was presented in the last section, perform generalization. This is because it classifies just the corners in the training samples. This limitation can be mitigated by modifications to CC2. For example, one may choose to learn facets from each training sample.

7. Conclusions

The new algorithm described in this paper is guaranteed to find the solution to any mapping problem. The effectiveness of this algorithm is due to the structured nature of the final design.

We have presented two corner classification algorithms, one of which (CC2) does not require any computations whatsoever. The corner classification based on the perceptron algorithm (CC1), must not be viewed as a generalization of that procedure. The perceptron algorithm lumps all vectors that produce 1 into the same class which is why it cannot be used to solve any nonlinear classification problem. CC1 algorithm is based on consideration of each corner separately. The algorithm is effective because we view the problem of mapping in a new way as a sum of corner classification sub-problems, and we use the vectors defined across the inputs to determine the design of the hidden layer. An improved version of the algorithm will be obtained if the hidden neurons are used to separate facets rather than corners.

CC1 can generalize from the training samples, the effectiveness of this generalization depends on how random the initial weights are. CC2 cannot generalize in the basic form.

The new algorithm is computationally extremely efficient compared to the back-propagation algorithm. It is also biologically plausible since the computations required to train the network are extremely simple.

References

[1] D E Rumelhart *et al Parallel distributed processing*, (The MIT Press, Cambridge, 1986) Vol. 1
[2] M L Minsky and S A Papert, *Perceptrons*, (The MIT Press, Cambridge, 1988)
[3] S C Kak, *Pramana – J. Phys.*, **38**, 271 (1992)
[4] Rosenblatt F, *Principles of neurodynamics*, (Spartan, New York, 1962)
[5] S A Papert, Some mathematical models of learning. *Proceedings of the fourth London symposium on information theory* (1961)