POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

Simple Neural Networks for Pattern Classification

K. Ming Leung

Abstract: A simple neural network capable of classifying patterns into two categories is introduced. It is trained using Hebb's learning rule.

Directory

Table of Contents

1. Simple NN for Pattern Classifications

We consider here the simplest type of NN capable of performing pattern classifications into two categories. For example, each input character must be classified as capital letter "X" or capital letter "O".

Since the output of the NN has only two possible values (a value for each of the two categories), we can choose an output layer containing only a single neuron of binary or bipolar value. For simplicity we assume an input layer with n binary or bipolar input neurons, $X_i, i = 1, \cdots, n$. We also assume that we do not need any hidden layers. Thus we are interested in the NN as shown in the figure.

For $i = 1, \cdots, n$ the activation of neuron X_i is denoted by x_i. Here we assume that the neurons all have bipolar values, and thus $x_i = \pm 1$. The case for neurons having binary values can obtained by straightforward modifications of the results here.

The input signal can then be represented by a vector $\mathbf{x} = [x_1 x_2 \ldots x_n]$. In the case of character recognition, this vector can be obtained for example, by discretizing the input character on a rectangular grid, with black dots representing by 1 and white dots by -1. The input

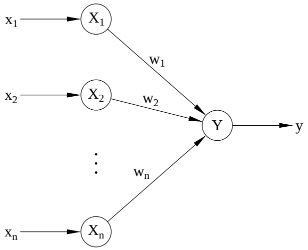

Figure 1: *A feedforward NN having N input and 1 output neurons.*

vector $\mathbf{x}$ is then obtained by concatenating those values column-wise or row-wise.

We assume that input neuron X_i, $i = 1, \cdots, n$ is connected to the output neuron, Y, with a weight of w_i. We denote the output of neuron Y by y, and so $y = \pm 1$.

The total input received by neuron Y is then given by

$$y_{\text{in}} = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n = \sum_{i=1}^{n} x_i w_i = \mathbf{x} \cdot \mathbf{w}.$$

We assume that the transfer function, f, is given by the bipolar step function with threshold θ, that is

$$f_\theta(x) = \begin{cases} +1, & \text{if } x \geq \theta, \\ -1, & \text{if } x < \theta. \end{cases}$$

Therefore the network output is

$$y = f_\theta(y_{\text{in}}) = \begin{cases} +1, & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta, \\ -1, & \text{if } \sum_{i=1}^{n} x_i w_i < \theta. \end{cases}$$

Thus the output y can be computed for any given input $\mathbf{x}$ provided that the weights $\mathbf{w}$ and the threshold, θ are known.

The above equation can be rewritten as

$$y = f_\theta(y_{\text{in}}) = \begin{cases} +1, & \text{if } -\theta + \sum_{i=1}^n x_i w_i \geq 0, \\ -1, & \text{if } -\theta + \sum_{i=1}^n x_i w_i < 0. \end{cases}$$

In the above expression, the term $-\theta$ can be considered as a bias $b = -\theta$. Therefore the threshold can be eliminated completely if we introduce an additional input neuron, X_0, whose value is always given by $x_0 = 1$, and is connected to the output neuron Y with a weight of $w_0 = b = -\theta$. Thus the above equation can be re-written as

$$y = f(y_{\text{in}}) = \begin{cases} +1, & \text{if } \sum_{i=0}^n x_i w_i \geq 0, \\ -1, & \text{if } \sum_{i=0}^n x_i w_i < 0, \end{cases}$$

where the transfer function is just the bipolar step function (with zero threshold):

$$f(x) = \begin{cases} +1, & \text{if } x \geq 0, \\ -1, & \text{if } x < 0. \end{cases}$$

This trick will often be used to remove the presence of a threshold or bias in a neuron.

Of course in order to compute the output we need to know the weights (and the bias). We will consider here the case of supervised learning first. This means that a set of learning patterns are given together with their corresponding target output. This is referred to as the training set:

$$\{\mathbf{s}^{(q)}, t^{(q)}\}, \qquad q = 1, 2, \ldots, Q.$$

For $q = 1, 2, \ldots, Q$, $\mathbf{s}^{(q)}$ is one of the training patterns, and $t^{(q)}$ is its corresponding targeted output value.

The NN has to be trained using the training set before it can be used to solve problems. During the training process, each of the training vector is presented to the NN as input, and the weights and bias(es) are then adaptively adjusted so that the NN correctly classifies all (or nearly so) the training patterns.

There are a few possible supervised training algorithms of interest here:

1. Hebb rule

2. perceptron learning rule
3. delta (or least mean squares) rule

We will consider the Hebb rule in this chapter. The perceptron learning rule and the delta rule will be considered in subsequent chapters.

But before we introduce the Hebb rule, we want to define what is meant by a decision boundary, and consider the important idea of linear separability.

2. Linearly-Separability and Decision Boundary

We define what is known as the decision boundary and introduce a very important concept called linear separability.

For a given weight vector $\mathbf{w} = [w_1 w_2 \ldots w_n]$ and bias b, the decision boundary is a hyperplane of dimension $n-1$ given by points $\mathbf{x} = [x_1 x_2 \ldots x_n]$ which obey the equation

$$b + \sum_{i=1}^{n} x_i w_i = b + \mathbf{x} \cdot \mathbf{w} = 0.$$

We move b to the right-hand side and divide the equation by the magnitude of $\mathbf{w}$ to obtain (recall that $b = -\theta$):

$$\hat{\mathbf{w}} \cdot \mathbf{x} = \frac{\theta}{w},$$

where $\hat{\mathbf{w}}$ is a unit vector (of unit magnitude) pointing in the direction of $\mathbf{w}$, and $w = |\mathbf{w}|$ is the magnitude of $\mathbf{w}$. Thus we see that this hyperplane is perpendicular to $\mathbf{w}$, cutting it at a distance of $\frac{\theta}{w}$ from the origin. If b is positive, then θ is negative, in that case the hyperplane cuts the vector $\mathbf{w}$ at a distance of $\frac{-\theta}{w}$ on the opposite side of $\mathbf{w}$ from the origin. It is clear that each given set of weights and bias determine a unique decision boundary.

For a given training set, if the training vectors whose target values are equal to $+1$, can be separated from those training vectors whose target values are equal to -1 by a hyperplane, then the problem is said to be linearly-separable. Clearly we can hope to be able to properly train a NN only if the problem is linearly-separable.

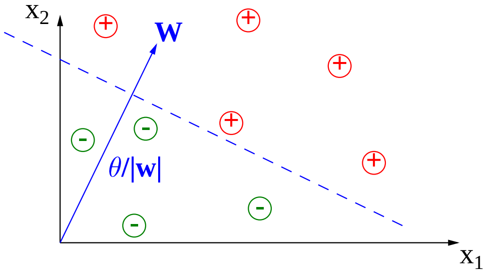

Figure 2: *A decision boundary (hyperplane) separating input vectors belonging to the two different classes.*

3. Hebb Rule

Hebb learning occurs by modification of the synapse strengths (weights) in a way that if 2 interconnected neurons are both ”on” (or both ”off”), then the weight should be further increased. For bipolar neurons, the change in the weight w_i for any i is given by

$$\Delta w_i = w_i^{\text{new}} - w_i^{\text{old}} = x_i y.$$

for the bias, since it can be replaced by a neuron whose input value is always fixed at 1, the updating rule for it is

$$\Delta b = b^{\text{new}} - b^{\text{old}} = y.$$

The Hebb rule is:

1. Initialize all weights and bias to zero (or some random values).
2. For each input training vector $\mathbf{s}^{(q)}$ and target $t^{(q)}$ pairs, go through the following steps
   (a) Set activations for input vector $\mathbf{x} = \mathbf{s}^{(q)}$.
   (b) Set activation for the output unit $y = t$.
   (c) Adjust weights and bias:

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + \mathbf{x}y,$$

$$b^{\text{new}} = b^{\text{old}} + y.$$

Note that we go through the training set in a single pass. The ordering of the training vectors in the training set does not matter. In fact if the initial weight vector and bias are given by $\mathbf{w}(0)$ and $b(0)$ respectively, the Hebb rule goes through the loop:

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{s}^{(k)}t^{(k)}, \qquad b(k) = b(k-1) + t^{(k)}$$

for $k = 1, \ldots, Q$ simply gives the following final results

$$\mathbf{w} = \mathbf{w}(0) + \sum_{k=1}^{Q} \mathbf{s}^{(k)} t^{(k)}, \qquad b = b(0) + \sum_{k=1}^{Q} t^{(k)}.$$

Every step in the Hebb's learning rule tends to move the decision boundary in such a way to better classify the particular training vector presented to the NN. It is easier to understand how the rule works especially if the bias is removed (absorbed). In that case, we have at the k-th step

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{s}^{(k)} t^{(k)},$$

thus

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{s}^{(k)}, \mathrm{if} t^{(k)} = +1,$$

and

$$\mathbf{w}(k) = \mathbf{w}(k-1) - \mathbf{s}^{(k)}, \mathrm{if} t^{(k)} = -1.$$

The following diagram explains the reasoning behind the Hebb's rule.

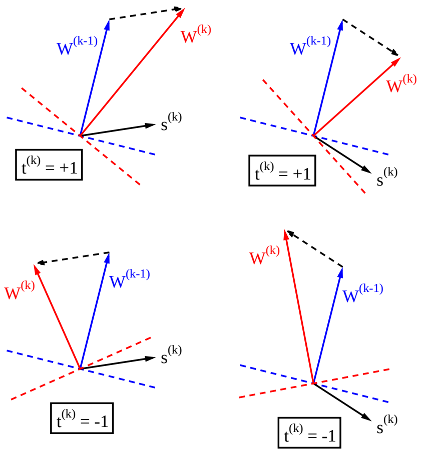

Figure 3: *One step in Hebb's rule, assuming either the threshold is absent or absorbed in the zeroth component.*

It should be remarked that Hebb's rule should not be used in its present form with binary output neurons, since the NN cannot learn any training vector whose target output is 0.

4. Applications

We will apply the Hebb rule to 2 NNs. We will use matrix notation to represent the vectors. Input and training vectors are row vectors, and the weight vector is a column vector. Therefore we write

$$y_{\text{in}} = b + \mathbf{x}\mathbf{w}$$

and

$$\mathbf{w} = \mathbf{w}(0) + \sum_{k=1}^{Q} t^{(k)} \mathbf{s}^{(k)T}, \qquad b = b(0) + \sum_{k=1}^{Q} t^{(k)}.$$

where superscript T represents the transpose.

4.1. Bipolar Logical Function: AND

The training set is given by the following table:

We assume that the weights and bias are initially zero. Therefore

$$\mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$

and

$$b = 1 - 1 - 1 - 1 = -2.$$

q	$\mathbf{s}^{(q)}$	$t^{(q)}$
1	[1 1]	1
2	[1 -1]	-1
3	[-1 1]	-1
4	[-1 -1]	-1

The first thing one must do after training is to see how the NN performs on the training data itself.

For $q = 1$, we have

$$y_{\text{in}} = b + \mathbf{s}^{(1)}\mathbf{w} = -2 + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 2.$$

Since $y_{\text{in}} \geq 0$, the network output is $y = 1$, which is the correct result. Similarly,

$$y_{\text{in}} = b + \mathbf{s}^{(2)}\mathbf{w} = -2 + \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = -2 \quad \Rightarrow y = -1,$$

$$y_{\text{in}} = b + \mathbf{s}^{(3)}\mathbf{w} = -2 + \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = -2 \quad \Rightarrow y = -1,$$

$$y_{\text{in}} = b + \mathbf{s}^{(4)}\mathbf{w} = -2 + \begin{bmatrix} -1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = -6 \quad \Rightarrow y = -1,$$

Thus the NN has learnt to perform perfectly for the training patterns.

Next we want to see how well the NN perform on patterns that it has not seen before. For the present example involving $n = 2$ bipolar vectors, we have no other vectors besides the training vectors. However from the weights and bias, we know that the decision boundary is given by

$$-2 + 2x_1 + 2x_2 = 0.$$

This gives the straight line

$$x_2 = -x_1 + 1,$$

having a slope of -1 and passing through the point [10]. Clearly not only does this decision boundary work for this training set, and it is actually the best solution in the sense that the ANN will perform well even in the presence of substantial noise.

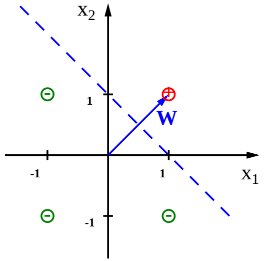

Figure 4: *For zero initial weights and bias, Hebb's rule finds the best decision boundary for the* `AND` *function.*

4.2. Character Recognition

We will use the Hebb rule to train the NN to distinguish between pattern "X" and the pattern "O". These two patterns are discretized on a 5×5 grid.

```
# . . . #          . # # # .
. # . # .          # . . . #
. . # . .          # . . . #
. # . # .          # . . . #
# . . . #          . # # # .
```

We assign each "#" the value 1, and each "." the value -1. The two-dimension patterns are converted to input vectors by concatenating the rows.

Thus the training set includes

$\mathbf{s}^{(1)} =$

$$[1\ {-1}\ {-1}\ {-1}\ \ 1\ {-1}\ \ 1\ {-1}\ \ 1\ {-1}\ {-1}\ {-1}\ \ 1\ {-1}\ {-1}\ {-1}\ \ 1\ {-1}\ \ 1\ {-1}\ \ 1\ {-1}\ {-1}\ {-1}\ \ 1]$$

with $t^{(1)} = 1$, and

$\mathbf{s}^{(2)} =$

$$[-1\ 1\ 1\ 1\ -1\ 1\ -1\ -1\ -1\ 1\ 1\ -1\ -1\ -1\ 1\ 1\ -1\ -1\ -1\ 1\ -1\ 1\ 1\ 1\ -1]$$

with $t^{(2)} = -1$.

From Hebb's rule, we obtain

$$\mathbf{w} =$$

$$[2\ -2\ -2\ -2\ 2\ -2\ 2\ 0\ 2\ -2\ -2\ 0\ 2\ 0\ -2\ -2\ 2\ 0\ 2\ -2\ 2\ -2\ -2\ -2\ 2]$$

and $b = 0$.

Again we first check to see if the trained NN perform well with the training data. For pattern 1, one finds that $y_{\text{in}} = 42$, and so $y = 1$, and for pattern 2, $y_{\text{in}} = -42$, and so $y = -1$. Thus the NN works well with the training data.

It is important to realize that the NN will give reasonable response even with patterns that it has not seen before. In particular, this will happen even when it is presented with input patterns that are imperfect. There are two important types of imperfection:

1. one or more components of the input vectors have their signs reversed.

2. one or more components of the input vectors are missing (for

example not measured by a scanner). These missing values may represent a 1 or a -1 but we are not sure, so we sometimes assign them a value of 0 (for bipolar vectors).

5. Remarks on Hebb's Learning Rule

There is no proof that Hebb's learning rule will always gives a set of weights and bias that allows the NN to correctly classify all the patterns in the training set. In fact it is easy to come up with examples where Hebb's learning rule fails to train a NN for even patterns in the training set.

Consider the following training set:

q	$\mathbf{s}^{(q)}$	$t^{(q)}$
1	[1 1 1]	1
2	[1 1 -1]	-1
3	[1 -1 1]	-1
4	[-1 1 1]	-1

Assuming zero initial weights and bias, Hebb's rule gives $\mathbf{w} = [000]$

and $b = -2$. Therefore for any input vector

$$y_{\text{in}} = b + \mathbf{x}\mathbf{w} = -2$$

always! The NN will therefore fails even with the training patterns. However, if one graphs the training set in a three-dimension plot, it is clear that this problem is linearly separable and therefore has solutions, that is, there are (actually infinitely number of) weights and biases that will enable the NN to correctly classify all the training patterns. However Hebb's rule fails to find a set of weights and bias that works (unless we happen to choose a correct set of starting values for the weights and bias).

Suppose we absorb the bias, and take the initial weights to be zero, the Hebb's rule gives

$$\mathbf{w} = \sum_{k=1}^{Q} t^{(k)} \mathbf{s}^{(k)T}.$$

We want to find the response of the NN when one of the training

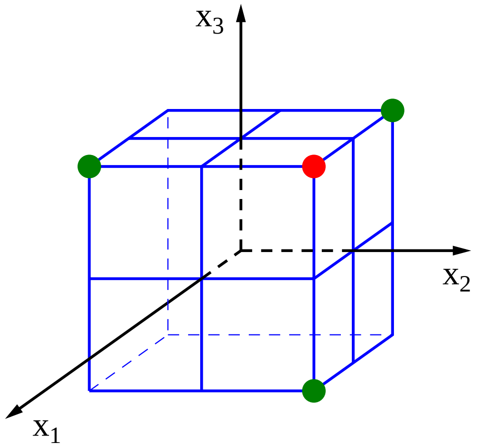

Figure 5: *This problem is clearly linearly separable, however Hebb's rule fails to find an acceptable decision boundary.*

vector $\mathbf{s}^{(m)}$ is presented to it. We find that

$$y_{\text{in}} = \mathbf{s}^{(m)}\mathbf{w} = \mathbf{s}^{(m)}\sum_{k=1}^{Q} t^{(k)}\mathbf{s}^{(k)T} = |\mathbf{s}^{(m)}|^2 t^{(m)} + \sum_{k\ =m}^{Q} t^{(k)}\mathbf{s}^{(m)}\mathbf{s}^{(k)T}.$$

The last term vanishes if the training vectors are mutually orthogonal. In that case, we have

$$y = f(y_{\text{in}}) = f(|\mathbf{s}^{(m)}|^2 t^{(m)}) = f(t^{(m)}) = t^{(m)}$$

which is the desire correct result. Therefore Hebb's rule always finds a set of weights and bias that correctly classify the training vectors if they are mutually orthogonal. Otherwise there is no guarantee that it will do so.

For training vectors that are not mutually orthogonal, the last term involving dot-products is not expected to be zero. It is due to "cross-talk" between the different patterns.

References

[1] See Chapter 2 in Laurene Fausett, ”Fundamentals of Neural Networks - Architectures, Algorithms, and Applications”, Prentice Hall, 1994.