1 McCulloch–Pitts Neuron

We start by introducing an extremely simplified model of a biological neuron, based on the early work of McCulloch and Pitts. This model is variously referred to as McCulloch-Pitts Neuron, Threshold Neuron, Threshold Logic Unit (TLU) and perceptron.

What follows is a caricature that describes the working of nervous systems: A nervous system is an organized network of neurons. A typical neuron consists of three parts: the dendrites, the cell body, and the axon (also called the nerve fiber). The dendrites carry nerve signals toward the cell body, while the axon carries the signal away from the cell body. Neural circuits are formed from groups of neurons arranged with the end branches of the axon lying close to the dendrites of another neuron. In the human brain, each neuron can interact with thousands of other neurons. The point of contact between the components of two neurons is called a synapse. A small microscopic gap between the two neurons exists at a synapse. It is known that the ease of neural signal transmission across the synapse is altered by activity in the nervous system — a possible mechanism for learning. When a neuron receives input from other neurons, the electro-chemical processes involved cause its voltage to increase. When the voltage exceeds a certain threshold, it results in a volley of nerve impulses that travel down the axon (thereby stimulating other neurons and so on). Or, the neuron is said to fire.

In McCulloch-Pitts model, numerical parameters called weights $w_1, \ldots, w_n$ model the strength of synaptic coupling between neurons. $x_1, \ldots, x_n$ model the inputs. $T$ models the threshold. For mathematical convenience, $T$ is replaced by a weight $-w_0$ and a fictitious input $x_0$ which is always constant (typically equal to unity) is introduced.

$$y = 1 \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i > T$$

$$\text{or} \quad \sum_{i=1}^{n} w_i x_i - T > 0$$

$$\text{or} \quad \sum_{i=0}^{n} w_i x_i > 0$$

$$\text{or} \quad W \cdot X > 0;$$

$$y = 0 \quad \text{otherwise}$$
Figure 1: McCulloch-Pitts Neuron. $W = [w_0, \ldots, w_n]^t$ stands for the weight vector; $X = [x_0, \ldots, x_n]^t$ stands for the input vector; where $A^t$ stands for the transpose of vector $A$.

2 Computational capabilities of a TLU

2.1 Connection with boolean logic

Suppose $x_i \in \{0, 1\}$ (i.e., the inputs are binary), and $x_0 = 1$. (A threshold neuron with binary inputs is called a binary threshold neuron). Suppose $w_1 = w_2 = \cdots = w_n = 1$. Consider a special case with $n = 2$. As evident from figure 3, this neuron functions as a 2-input AND gate.

\[
\begin{array}{cccc}
  x_1 & x_2 & W \cdot X & y \\
  0 & 0 & -1.5 & 0 \\
  0 & 1 & -0.5 & 0 \\
  1 & 0 & -0.5 & 0 \\
  1 & 1 & 0.5 & 1 \\
\end{array}
\]

Figure 2: A threshold neuron that functions like an AND gate.

Similarly, it is possible to select weight values so that an $n$-input threshold neuron functions as an $n$-input AND gate or as an $n$-input OR gate. It is also straightforward to select weights so that a 1-input threshold neuron (i.e., $n = 1$) functions as a NOT gate. Since any boolean function can be expressed in terms of AND, OR, and NOT functions, we have the following theorem:

**Theorem 1.** A sufficiently large network of binary threshold neurons can compute any arbitrary boolean function.

**Proof.** Omitted.

McCulloch and Pitts also went on to prove that arbitrary finite state automata can be realized using binary threshold logic units with unit time delay (between input and output). Since it is known from the theory of computation (developed by Turing, Kleene, Church, Rosser, Post, and others) that any arbitrary algorithm can be realized by a sufficiently large finite state machine, we can assert that there exist networks of threshold neurons that can compute any computable function.
Although any boolean function can be computed by an appropriately designed network of binary threshold neurons, there are functions that cannot be computed by a single binary threshold neuron. Exclusive OR (EXOR) is one such function. To see this, consider the constraints on weights that need to hold in order for a neuron to compute a 2-input EXOR:

\[
\begin{array}{ccc}
  x_1 & x_2 & y \\
  0 & 0 & 0 & \rightarrow & w_0 \leq 0 \\
  0 & 1 & 1 & \rightarrow & w_0 + w_2 > 0 \\
  1 & 0 & 1 & \rightarrow & w_0 + w_1 > 0 \\
  1 & 1 & 0 & \rightarrow & w_0 + w_1 + w_2 \leq 0 \\
\end{array}
\]

Clearly, there is no set of weights that simultaneously satisfy each of these 4 inequalities. Thus we have the following theorem:

**Theorem 2.** There exist logical functions that cannot be realized by a single binary threshold neuron.

**Definition 1.** The class of boolean functions that can be realized by a single binary threshold neuron are called boolean threshold functions.

In the light of the above, it is natural to ask what fraction of the number of possible \( n \)-input boolean functions can be realized by an \( n \)-input binary threshold neuron. To this day, we do not have an exact closed-form formula that provides an answer to this question. However, the number of possible threshold functions have been enumerated for \( n \leq 6 \). Let us consider the case for \( n = 2 \). There are \( 2^4 = 16 \) boolean functions of 2 inputs. (There are 4 binary patterns: 00, 01, 10, and 11.

A boolean function partitions the set of patterns into 2 classes and there are \( \binom{4}{0} + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} = 16 \) ways to do this). Of these, 14 can be shown to be threshold functions. Before you jump to the conclusion that the number \( N_T(n) \) of \( n \)-input boolean threshold functions is a relatively large fraction of the number \( N_B(n) \) of \( n \)-input boolean functions, it must be noted that as \( n \) increases, threshold functions form a vanishingly small subset of possible boolean functions. In fact, it can be shown that \( N_T(n) \leq 2^n - 2 \).

Recall that \( N_B(n) = 2^{2^n} \). It follows that the limit of \( \frac{N_T(n)}{N_B(n)} \) approaches 0 as \( n \) approaches infinity.

### 2.2 Connection with Geometry

Consider 2-input boolean threshold neurons. The input patterns can be thought of as points in 2-d Euclidian space.

Suppose we draw a line given by \( x_2 = -x_1 + 0.5 \) in this space. The equation of the line can be written as \( w_1 \cdot x_1 + w_2 \cdot x_2 + w_0 = 0 \) where \( w_1 = 1, w_2 = 1, w_0 = -0.5 \). Thus, a 2-input threshold neuron partitions a
2-dimensional Euclidean space into two regions (of half-spaces). One one side of the line are points for which the neuron produces an output of 1. On the other side are points for which the neuron produces an output of 0. We have used the convention that if a point falls on the line, its output is 0. (But other variations on this leave the computational model essentially unchanged).

Can this be generalized into \( n \)-dimensional space?

As shown in the figure, a 3-input threshold neuron realizes a 2-D plane that partitions a 3D Euclidean space into two half-spaces. We can generalize this example to threshold neurons with an arbitrary, but finite number of inputs \((n)\). In general, an \( n \)-input threshold neuron with weight vector \( \mathbf{W} = [w_0 \cdots w_n]^t \) implements an \( n-1 \) dimensional hyperplane that divides the \( n \)-dimensional Euclidean space into two half-spaces. The equation of the plane is given by: \( \mathbf{W} \cdot \mathbf{X} = 0 \) (for some choice of \( \mathbf{W} \)), \( \mathbf{X} = [1, x_1 \cdots, x_n]^t \) defines the axes of the \( n \)-dimensional space. The values of the components of \( \mathbf{W} \) define the location and orientation of the hyperplane. The hyperplane has two sides: a positive (+ve) side and a negative (-ve) side.

Example 1. Assume \( \mathbf{W} = [w_0 \cdots w_3]^t = [1 1 \; 1 1]^t \) and \( \mathbf{X}_1 = [1 0 1 0]^t \). What side of the hyperplane defined by \( \mathbf{W} \cdot \mathbf{X} = 0 \) does \( \mathbf{X}_1 \) fall?

\( \Rightarrow \) On the plane since \( \mathbf{W} \cdot \mathbf{X}_1 = 0 \).

Note that although our examples have used binary inputs, in general, threshold neurons can have multi-valued inputs or real-valued inputs.

2.2.1 A Little Geometry

Because of the close relation between \( n \)-input threshold neurons and \( n \)-dimensional Euclidean space, we will digress to review some geometry.

- The equation of \( n-1 \) dimensional hyperplane in \( n \)-dimensional space is given by
  \[ w_0 + w_1 x_1 + \cdots + w_n x_n = 0 \]

- The orientation of the plane is governed by \((w_1 \cdots w_n)\) and its location by its distance from the origin is governed by \( w_0 \).
• The magnitude of the perpendicular vector distance of the hyperplane (1.1) from the origin is

$$\frac{|w_0|}{\sqrt{w_1^2 + \cdots + w_n^2}}$$

• The distance of an arbitrary point given by \((x_1, \ldots, x_n)\) from the plane (1.1) is

$$\frac{|w_1x_1 + \cdots + w_nx_n + w_0|}{\sqrt{w_1^2 + \cdots + w_n^2}}$$

![Figure 4: Hyperplanes and Direction of Unit Normal Vectors](image)

**Example 2.** See Fig. 4.

In summary, a threshold neuron is a highly simplified computational model of a neuron. As we have seen, even such a simplified model has fairly interesting computational properties. In particular, a threshold neuron, given an appropriate choice of the weights, is capable of computing any threshold function. This raises the question as to whether such a device can be \textit{taught} a desired (but a-priori unknown) threshold function from examples of the desired inputs and the corresponding outputs. If this could be done, as Rosenblatt and other early researchers in neural computation realized, threshold neurons could be used to build trainable pattern classifiers. Such pattern classifiers can potentially be used to solve pattern recognition problems that arise in diagnosis, handwriting recognition, etc. Clearly, since the exact function computed by a threshold neuron depends on the choice of weights, an obvious possibility for learning is to modify the weights so as to correctly classify a given training set. If the training set is \textit{representative} of the domain of interest, one can hope that the resulting classifier would perform quite reasonably on novel input patterns. More on this later. In the next section, we consider the task of learning threshold functions from examples.

### 3 Learning threshold functions from examples

As we saw in the previous sections, an \(n\)-input McCulloch-Pitts neuron can compute an interesting subset of the set of functions \(\{\phi : \mathbb{R}^n \rightarrow \{0, 1\}\}\). Since the set of functions that are computable by an \(n\)-input threshold neuron include many functions of practical interest (including boolean threshold functions) which find applications in pattern classification tasks (e.g., diagnosis), a natural question to ask is whether it is possible to \textit{learn} particular threshold functions from \textit{examples} of the desired input-output mapping. This chapter explores a simple algorithm — the so-called \textit{perceptron algorithm} due to Rosenblatt (among others) that can be used to train a threshold neuron to compute a function implicitly specified using a set of training examples.

We start with a few definitions.

A training example \(\mathcal{E}_k\) is an ordered pair \((X_k, c_k)\) where \(X_k = [x_{0k}, \ldots, x_{nk}]\) is a pattern vector (\(\forall k x_{0k} = 1\)) and \(c_k = 0\) or \(1\) (depending on the desired output of the neuron for input pattern \(X_k\)). A \textit{training set} \(\mathcal{E}\) is simply a set of training examples \(\mathcal{E} = \{\mathcal{E}_k\}\).
Learning Task: Given a training set $\mathcal{E}$, find a weight vector $W_* = [w_0, \ldots, w_n]$ such that

$$\forall x_k \in S^+, W_* \cdot x_k > 0 \quad \text{and} \quad \forall x_k \in S^-, W_* \cdot x_k \leq 0.$$ 

It goes without saying that we can hope to find such a weight vector $W_*$ only if one exists, that is, iff $\mathcal{E}$ implicitly defines a threshold function.

### 4 Perceptron Algorithm

Having defined the learning task, we now examine a relatively simple algorithm – more specifically, the fixed correction rule, popularly known as the perceptron algorithm, for training a threshold neuron.

Intialize $W \leftarrow [0 \ldots 0]$

until a complete pass through the training set results in no weight updates do:

1. Select an example $\mathcal{E}_k = (x_k, c_k)$ and compute $W \cdot x_k$, the output of the neuron $o_k = 1$ if $W \cdot x_k > 0$, or $o_k = 0$ otherwise
2. $W \leftarrow W + \eta(c_k - o_k)x_k$  

Example

The following example illustrates the working of the perceptron algorithm using a 2-input threshold neuron and a set of training examples for the boolean OR function.

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$c_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$X_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$X_3$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$X_4$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Assume that patterns are selected by cyclic order (i.e., $X_1, X_2, X_3, X_4, X_1, \cdots$) and $\eta = 1$ always.

initialize: $W = [0\ 0\ 0]$

1. $W \cdot X_1 = [0\ 0\ 0] \cdot [1\ 0\ 0] = 0 \Rightarrow o_1 = 0 = c_1$ (no change in $W$)
2. $W \cdot X_2 = [0\ 0\ 0] \cdot [1\ 0\ 1] = 0 \Rightarrow o_2 = 0 \neq c_2$
3. $W = [0\ 0\ 0] + (1 - 0)[1\ 0\ 1] = [1\ 0\ 1]$
4. $W \cdot X_3 = [1\ 0\ 1] \cdot [1\ 1\ 0] = 1 \Rightarrow o_3 = 1 = c_3$ (no change in $W$)
5. $W \cdot X_4 = [1\ 0\ 1] \cdot [1\ 1\ 1] = 2 \Rightarrow o_4 = 1 = c_4$ (no change in $W$)

initialize: $W = [1\ 0\ 1] + (0 - 1)[1\ 0\ 0] = [0\ 0\ 1]$

1. $W \cdot X_2 = [0\ 0\ 1] \cdot [1\ 0\ 1] = 1 \Rightarrow o_2 = 1 = c_2$ (no change in $W$)
2. $W \cdot X_3 = [0\ 0\ 1] \cdot [1\ 1\ 0] = 0 \Rightarrow o_3 = 0 \neq c_3$
3. $W = [0\ 0\ 1] + (1 - 0)[1\ 1\ 0] = [1\ 1\ 1]$
4. $W \cdot X_4 = [1\ 1\ 1] \cdot [1\ 1\ 1] = 3 \Rightarrow o_4 = 1 = c_4$ (no change in $W$)

initialize: $W = [1\ 1\ 1] + (0 - 1)[1\ 0\ 0] = [0\ 1\ 1]$

1. $W \cdot X_2 = [0\ 1\ 1] \cdot [1\ 0\ 1] = 1 \Rightarrow o_2 = 1 = c_2$ (no change in $W$)
2. $W \cdot X_3 = [0\ 1\ 1] \cdot [1\ 1\ 0] = 1 \Rightarrow o_3 = 1 = c_3$ (no change in $W$)
3. $W \cdot X_4 = [0\ 1\ 1] \cdot [1\ 1\ 1] = 2 \Rightarrow o_4 = 1 = c_4$ (no change in $W$)

initialize: $W = [0\ 1\ 1] \cdot [1\ 0\ 0] = 0 \Rightarrow o_1 = 0 = c_1$ (no change in $W$)
Thus, a complete pass through the training set in the last four pattern presentations shown above results in no changes in \( W \). So the process terminates with the final weight vector \( W_\star = [0 \ 1 \ 1] \) which implements the function specified by the training set.

## 5 Perceptron Convergence Theorem

As we saw in the preceding example, the algorithm appears to successfully perform the learning task. But it is not obvious as to whether it would work in general. In particular, it is not obvious that weight changes that correct the output for some of the patterns do not for ever cause some other previously correct outputs to be changed (as a result of the weight changes). Fortunately however, it turns out that we can prove (after Novikoff) that the perceptron algorithm is guaranteed to find a solution to the learning task outlined above whenever a solution exists. Obviously, no algorithm can be expected to find a solution that does not exist. In particular, if the function to be learned is not a threshold function (e.g., exclusive OR), it cannot be computed by a threshold neuron and so no algorithm can train a threshold neuron to compute it. We will consider more powerful learning algorithms that work with networks of neurons to deal with this problem later. But first, we prove the perceptron convergence theorem.

**Theorem:**

Given a training set consisting of +ve and -ve samples (\( S^+ \) and \( S^- \) respectively) the perceptron learning algorithm is guaranteed to find a weight vector \( W_\star \) such that \( \forall X_k \in S^+ (c_k = 1), W_\star \cdot X_k \geq \delta \) and \( \forall X_k \in S^- (c_k = 0), W_\star \cdot X_k \leq -\delta \), for some \( \delta > 0 \), whenever such a \( W_\star \) exists.

**Proof**

The basic idea behind the proof is to exploit the geometric fact that the cosine of the angle between any two vectors is at most 1. In particular, this has to hold for the solution vector \( W_\star \) and the weight vector being manipulated by the perceptron algorithm. We will use this fact, along with some observations regarding the weight updates performed by the perceptron algorithm, to prove the convergence theorem.

First, without loss of generality, assume that \( W_\star \cdot X \) (the solution hyperplane) passes through the origin of the pattern space spanned by the co-ordinates \( x_1 \cdots x_n \). In other words, \( w_{0_\star} = 0 \). This simplifies the proof considerably without sacrificing its generality for the following reason: If there is a solution hyperplane which does not pass through the origin, we can make it pass through the origin simply by moving the origin by an appropriate amount. Second, we simplify matters by transforming the learning task so that we only have to consider positive samples:

\[
\text{Let } Z_k = X_k \text{ if } c_k = 1 (W_\star \cdot X_k > \delta \Leftrightarrow W_\star \cdot Z_k \geq \delta) \Rightarrow \neg X_k \text{ if } c_k = 0 (W_\star \cdot X_k \leq -\delta \Leftrightarrow W_\star \cdot Z_k \geq \delta)
\]

So the modified training set will consist of only +ve examples \( \{(Z_k, 1)\} \). Let \( Z = \{Z_k\} \). In order to prove the theorem, we need to show that the perceptron algorithm is guaranteed to find a \( W_\star \) such that \( \forall Z_k \in Z W_\star \cdot Z_k \geq \delta \).

Let \( W_t \) = weight vector after \( t \) weight updates. (If the weights do not change as a result of a pattern presentation, then it does not qualify as an update). Let \( W_{t+1} \) = weight vector after \( t + 1 \) weight updates. Let \( W_0 = [0 \cdots 0] \). (It is easy to extend the proof so that it works for the case where \( W_0 \) is initialized to an arbitrary weight vector).

\[
W_{t+1} = W_t + \eta (c_k - o_k) Z_k
\]

always 1

\[
W_\star \cdot W_{t+1} = W_\star \cdot (W_t + \eta Z_k)
\]

\[
= W_\star \cdot W_t + \eta \underbrace{W_\star \cdot Z_k}_{\geq \delta}
\]

\[
\geq W_\star \cdot W_t + \eta \delta
\]

It follows that:

\[
W_\star \cdot W_t \geq \eta \delta
\]
Now let us examine how the length of the weight vector changes as a result of weight updates:

\[
\|W_{t+1}\|^2 = W_{t+1} \cdot W_{t+1} = (W_t + \eta Z_k) \cdot (W_t + \eta Z_k) = W_t \cdot W_t + 2\eta W_t \cdot Z_k + \eta^2 Z_k \cdot Z_k = \|W_t\|^2 + 2\eta W_t \cdot Z_k + \eta^2 \|Z_k\|^2 \leq 0
\]

Now we note that given any finite training set, the length of the training patterns is bounded. That is, \(\forall Z_k, \|Z_k\| \leq L\). It follows that:

\[
\|W_t\|^2 \leq t\eta^2 L^2 \tag{2}
\]

We use the above results to show that the number of updates, \(t\), is bounded. Consider:

\[
t\eta \delta \leq \|W_*\| \|W_t\| \cos \theta \leq \|W_*\| \|W_t\|.
\]

Substituting for \(\|W_t\|\) we have:

\[
t \eta \delta \leq \|W_*\| \sqrt{t} \eta L
\]

Thus, \(t\) is bounded. This proves the theorem.

Remarks:

- Note that the learning rate does not appear in the bound we just derived for the number of weight updates. So the convergence theorem holds for any bounded learning rate \(\eta > 0\).

- The bound that we have established is not useful in terminating the algorithm in practice (since \(\|W_*\|\) is not known until we have found a solution.).

The perceptron algorithm is quite robust as a learning model. It can be shown that the algorithm converges even when \(\eta\) is allowed to fluctuate arbitrarily over time as long as \(0 < \eta_t \leq B\) where \(B\) is the upper bound on the learning rate.

Corollary: Suppose the training patterns are binary: \(X_k = [X_{0k}, \ldots, X_{nk}]\) where \(X_{ik} = 0\) or \(1\). Thus we have: \(L = \sqrt{n + 1}\). If the patterns can be separated (i.e., \(W_* \cdot Z_k \geq \delta\) where \(\delta > 0\)) then there exists a \(W_*\) that separates patterns with \(\delta = 1\). It follows that:

\[
t \leq \left(\frac{\|W_*\| L}{\delta}\right) \quad \text{or} \quad t \leq \|W_*\|^2 (n + 1).
\]

6 Variants of the Perceptron Algorithm

Perceptron algorithm offers a simple, provably convergent procedure for finding a weight vector that corresponds to a separating hyperplane whenever the training set is linearly separable. A number of variants of the perceptron algorithm that converge faster to a solution than the simple algorithm discussed in this chapter are available. One such algorithm is based on the so-called fractional correction rule. It is better understood in terms of an alternative representation of a threshold neuron called the weight space representation. Any choice of weights say \(W_p = [W_{0p}, \ldots, W_{np}]\) can be thought of as a point in an \((n + 1)\)-dimensional “weight space” with coordinate axes \((w_0, \ldots, w_n)\). In weight space, a pattern vector \(X_k = [X_{0k}, \ldots, X_{nk}]\) defines a hyperplane given by \(W \cdot X_k = 0\) where \(W = [w_0, \ldots, w_n]\).

Thus, in the weight space representation is the dual of the pattern space representation.
Weights are fixed after training, but learning modifies the weights. The idea is to find the weight coordinates that lie on the correct side of all the pattern-defined hyperplanes. Thus, starting from some initial location, the point that defines the weights moves in the weight space as weights are updated according to some learning rule.

If a weight vector $\mathbf{W}_t$ incorrectly classifies a pattern $\mathbf{X}_k$ (corresponding to pattern hyperplane $\mathbf{W}_t \cdot \mathbf{X}_k = 0$) all we need to do is to change $\mathbf{W}_t$ so that it moves across the hyperplane. The quickest way to move a point across a pattern hyperplane defined by a pattern that is incorrectly classified is along the normal to that hyperplane. $\mathbf{X}_1$ is the normal to the hyperplane $\mathbf{W} \cdot \mathbf{X}_1 = 0$

The learning rate $\eta$ in the perceptron learning rule controls how far $\mathbf{W}$ is moved. The perceptron algorithm discussed earlier changes $\mathbf{W}_t$ as follows:

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t + \eta \mathbf{X}_k$$

What if we allow the learning rate to vary? Suppose we choose $\eta$ such that at each weight update, we correct for the entire error in one shot.

$$\mathbf{W}_{t+1} \cdot \mathbf{X}_k = (\mathbf{W}_t + \eta \mathbf{X}_k) \cdot \mathbf{X}_k$$

We then choose $\eta_t$ such that

$$\mathbf{W}_{t+1} \cdot \mathbf{X}_k > 0 \text{ if } \mathbf{W}_t \cdot \mathbf{X}_k < 0 \text{ and } c_k = 1.$$ 

So

$$\eta_t = \left[ \frac{\mathbf{W}_t \cdot \mathbf{X}_k}{\mathbf{X}_k \cdot \mathbf{X}_k} \right]$$

This gives the so-called absolute correction rule. Alternatively, we could choose $\eta$ to be

$$\lambda \frac{\mathbf{W}_t \cdot \mathbf{X}_k}{\mathbf{X}_k \cdot \mathbf{X}_k}$$

where $0 < \lambda \leq 2$

This gives the fractional correction rule. Both absolute correction and fractional correction rules are provably convergent. We omit the proofs here but the interested reader is referred to [Nilsson, 1965].

7 Discussion

The perceptron algorithm is poorly behaved on training sets that are not *linearly separable*. A number of better-behaved variants of the perceptron algorithm have been proposed to address this problem so that we can find a reasonably good solution when a perfect solution (i.e., a separating hyperplane) does not exist.
There are clearly practical pattern classification problems that are not amenable to solution using a single threshold neuron. Thus, we need to consider algorithms for constructing and/or training networks of neurons. Alternatively, given some prior knowledge about the pattern classification task, one might be able to transform the patterns so that they become linearly separable.

8 Multi-Category Perceptrons

So far, we have considered the use of threshold neurons as 2-category pattern classifiers. Many practical pattern classification tasks involve multiple categories. A simple-minded extension of 2-category classifiers to $M$-category classifiers involves the training of each of the $M$ neurons to separate the patterns corresponding its assigned category from the rest of the patterns in the training set using linear hyperplanes. However, as we shall see shortly, this falls short of exploiting the full computational capabilities of a group of $M$ threshold neurons.

In a WTA network, each of the neurons computes its net input as the dot product of its weight vector with the input pattern. The neuron with the highest net input outputs a 1, and all other neurons output 0. If there is a tie, all of the neurons output 0. This mode of operation is inspired by cortical circuits of neurons which laterally inhibit other neurons in their neighborhood. This inhibition is thought to play a role in a variety of functions from contrast enhancement of visual input to learning. We will examine the detailed processes underlying such competitive interactions among neurons and their implications in terms of information processing functions of the nervous systems later. For now, a simplified algorithmic description of the result of such interaction as outlined above is adequate for our purposes.

Consider a simple network of 3 neurons shown in the figure.

$W_1 = \begin{bmatrix} 1 & -1 & -1 \end{bmatrix}$

$W_2 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$

$W_3 = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix}$
Hyperplane for 3-input OR function

\[ w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0 \]

Figure 7: An Example of WTA Network

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( W_1 \cdot X )</th>
<th>( W_2 \cdot X )</th>
<th>( W_3 \cdot X )</th>
<th>( O_1 )</th>
<th>( O_2 )</th>
<th>( O_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>+1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>+1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Upon closer scrutiny, we see that the third neuron (with output \( O_3 \)) is computing the exclusive-OR function. Of course, we know that exclusive-OR function is not a threshold function. Thus, this example establishes that WTA networks can compute a richer class of functions than a single threshold neuron. We will see that they still fall short of the computing abilities of arbitrary networks of threshold neurons, but nevertheless, they offer interesting designs for multi-category pattern classifiers. We can define the class of functions computed by the WTA groups (linearly separable functions) as follows:

Let \( S = \{X_1, \ldots, X_p\} \) be a set of pattern vectors.
Suppose each \( X_k \in S \) belongs to exactly one of \( M \) classes: \( C_1, \ldots, C_m \). Where \( S = C_1 \cup C_2 \cup \ldots \cup C_m, C_i \cap C_j = \emptyset \) \( \forall i \neq j \).
The set \( S \) is said to be linearly separable if \( \exists \) weight vectors \( W_1, \ldots, W_m \) such that \( \forall \) classes \( C_j \), we have \( \forall X_k \in C_j, W_j \cdot X_k > W_i \cdot X_k \) \( \forall i \neq j \).
Note that 2-neuron WTA groups are computationally equivalent to a single threshold neuron. Thus, WTA groups are interesting only when the number of neurons in the group is at least 3.

9 Traing WTA networks

The learning algorithm for WTA networks is similar to Perceptron learning. Let: \( W_i = \) weight vector of neuron \( i \); \( X_k = k^{th} \) input pattern, \( X_{k0} = 1 \); \( O_{ik} = \) output of neuron \( i \) for pattern \( X_k \); \( O_{ik} = 1 \) iff \( W_i \cdot X_k > W_j \cdot X_k \) \( \forall j \neq i \), otherwise \( O_{ik} = 0 \).
As in the case of single neuron training, we cycle through the patterns in the training set one at a time until all the patterns are correctly classified by the winner take all group. However, we modify the weights of the wrongly off neuron and the wrongly on neuron and leave the rest of the weight vectors unchanged. (This is necessary to guarantee that a solution will be found if one exists).
Suppose for the input pattern \( X_k \) belonging to class \( C_j \), the output \( O_{ik} = 1 \). We want \( O_{jk} \) to be a 1, so neuron \( i \) is wrongly on and neuron \( j \) is wrongly off. We update the corresponding weight vectors as follows:

\[
W_i \leftarrow W_i - \eta X_k \\
W_j \leftarrow W_j + \eta X_k
\]
Note that all other weight vectors \( W_l (l \neq i; l \neq j) \) are left unchanged.
In the event of a tie, we add a fraction of the pattern vector to the weight vector of the neuron that was wrongly off.
Note that there are clearly other ways to solve an M-category pattern classification task. For instance, we can transform the problem into one of solving M 2-category classification problems by attempting to separate each class from the rest using single neuron training algorithm. However, this solution has the following drawbacks relative to WTA training:

- There may be regions in the pattern space for which classification is not unique (where more than one neuron outputs 1).
• There are situations where such a strategy fails to find appropriate weight vectors for a WTA group even if they exist.

Thus it is advantageous to use WTA groups for Multi-category pattern classification.

10 WTA Convergence Theorem

If a given multi-category pattern set \( S \) is linearly separable then the WTA algorithm will find the weight vectors \( W_1 \ldots W_M \) that correctly classify \( S \) using a WTA network with the correspondly weight vectors.

Proof Sketch

Suppose \( X \in S \) and \( X \in C_i \). So we want:

\[
(W_i - W_j) \cdot X > 0 \forall j \neq i
\]

1 can be written as:

\[
W \cdot P_{ij} > 0 \forall j \neq i
\]

where \( W = [W_1 \ldots W_j \ldots W_m] \) is obtained by concatenating the individual weight vectors and \( \{P_{ij}\} \) \( (j \neq i, 1 \leq i, j \leq M) \) denotes a modified training set constructed as explained below.

Consider a pattern \( X \) that belongs to \( C_1 \). Then we construct \( (M - 1) \) modified training patterns as follows:

\[
P_{12} = [X -X \phi \phi \ldots \phi].
\]

\[
P_{13} = [X \phi -X \phi \ldots \phi].
\]

\[
P_{1M} = [X \phi \ldots \phi -X].
\]

where \( \phi \) denotes a vector of zeros (with the same dimension as \( X \). Thus, there are \( M - 1 \) patterns in the modified training set for each pattern in the original training set. It is easy to see that the original \( M \)-neuron WTA group training problem has a solution if and only if the single neuron training problem has a solution. That is, \( 2 \) is a single neuron (perception) training problem with weight vector \( W \) and a modified training set made of \( \{P_{ij}\} \). \( 2 \) has a solution iff \( 1 \) has a solution. So the convergence of the WTA training algorithm follows from the perceptron convergence proof.

Note that the transformation of the training set outlined above is not a practical approach to finding the weight vectors for the WTA group (since it requires processing \( M - 1 \) modified patterns for each pattern in the given training set).