3

PERCEPTRON ALGORITHM

Vasant Honavar

Artificial Intelligence Research Group
Department of Computer Science
Iowa State University
Ames, Iowa 50011

©Vasant Honavar, 1996.

1 LEARNING THRESHOLD FUNCTIONS FROM EXAMPLES

As we saw in the previous chapter, an n-input McCulloch-Pitts neuron can compute an interesting subset of the set of functions $\{\phi : \Re^n \rightarrow \{0,1\}\}$. 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 *learn* particular threshold functions from *examples* of the desired input–output mapping. This chapter explores a simple algorithm – the so-called *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 $(\mathbf{X}_k, c_k)$ where $\mathbf{X}_k = [x_{0k}, \cdots, x_{nk}]$ 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 $\mathbf{X}_k$. A training set $\mathcal{E}$ is simply a set of training examples $\mathcal{E} = \{\mathcal{E}_k\}$.

Let $\mathcal{S}^+ = \{\mathbf{X}_k | \mathcal{E}_k = (\mathbf{X}_k, c_k) \in \mathcal{E}, c_k = 1\}$ denote the set of 'positive' patterns (i.e., the training patterns for which the desired output is 1) and $\mathcal{S}^- =$

$\{\mathbf{X}_k | \mathcal{E}_k = (\mathbf{X}_k, c_k) \in \mathcal{E}, c_k = 0\}$ denote the set of 'negative' patterns (i.e., the training patterns for which the desired output is 0).

Learning Task: Given a training set $\mathcal{E}$, find a weight vector $\mathbf{W}_* = [w_{0*}, \cdots, w_{n*}]$ such that

$$\forall \mathbf{X}_k \in \mathcal{S}^+, \mathbf{W}_* \cdot \mathbf{X}_k > 0 \text{ and } \forall \mathbf{X}_k \in \mathcal{S}^-, \mathbf{W}_* \cdot \mathbf{X}_k \leq 0$$

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

2 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 ← [0 . . . 0]
until a complete pass through the training set results in  no
    weight updates do:  {
          1.  Select an example E_k = (X_k, c_k) and compute W · X_k,
              the output of the neuron o_k = 1 if W · X_k > 0
                                           = 0 otherwise
          2.
W ← W + η(c_k − o_k)X_k
}
W_* ← W
note:  η = learning rate > ∅
```

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.

$\mathbf{X}_k$	x_0	x_1	x_2	c_k
$\mathbf{X}_1$	1	0	0	0
$\mathbf{X}_2$	1	0	1	1
$\mathbf{X}_3$	1	1	0	1
$\mathbf{X}_4$	1	1	1	1

Assume that patterns are selected by cyclic order (i.e., $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4, \mathbf{X}_1, \cdots$) and $\eta = 1$ always.

initialize: $\mathbf{W} = [0\ 0\ 0]$

step 1: $\mathbf{W} \cdot \mathbf{X}_1 = [0\ 0\ 0] \cdot [1\ 0\ 0] = 0 \Rightarrow o_1 = 0 = c_1$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_2 = [0\ 0\ 0] \cdot [1\ 0\ 1] = 0 \Rightarrow o_2 = 0 \neq c_2$
step 2: $\mathbf{W} = [0\ 0\ 0] + (1-0)[1\ 0\ 1] = [1\ 0\ 1]$

step 1: $\mathbf{W} \cdot \mathbf{X}_3 = [1\ 0\ 1] \cdot [1\ 1\ 0] = 1 \Rightarrow o_3 = 1 = c_3$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_4 = [1\ 0\ 1] \cdot [1\ 1\ 1] = 2 \Rightarrow o_4 = 1 = c_4$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_1 = [1\ 0\ 1] \cdot [1\ 0\ 0] = 1 \Rightarrow o_1 = 1 \neq c_1$
step 2: $\mathbf{W} = [1\ 0\ 1] + (0-1)[1\ 0\ 0] = [0\ 0\ 1]$

step 1: $\mathbf{W} \cdot \mathbf{X}_2 = [0\ 0\ 1] \cdot [1\ 0\ 1] = 1 \Rightarrow o_2 = 1 = c_2$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_3 = [0\ 0\ 1] \cdot [1\ 1\ 0] = 0 \Rightarrow o_3 = 0 \neq c_3$
step 2: $\mathbf{W} = [0\ 0\ 1] + (1-0)[1\ 1\ 0] = [1\ 1\ 1]$

step 1: $\mathbf{W} \cdot \mathbf{X}_4 = [1\ 1\ 1] \cdot [1\ 1\ 1] = 3 \Rightarrow o_4 = 1 = c_4$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_1 = [1\ 1\ 1] \cdot [1\ 0\ 0] = 1 \Rightarrow o_1 = 1 \neq c_1$
step 2: $\mathbf{W} = [1\ 1\ 1] + (0-1)[1\ 0\ 0] = [0\ 1\ 1]$

step 1: $\mathbf{W} \cdot \mathbf{X}_2 = [0\ 1\ 1] \cdot [1\ 0\ 1] = 1 \Rightarrow o_2 = 1 = c_2$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_3 = [0\ 1\ 1] \cdot [1\ 1\ 0] = 1 \Rightarrow o_3 = 1 = c_3$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_4 = [0\ 1\ 1] \cdot [1\ 1\ 1] = 2 \Rightarrow o_4 = 1 = c_4$ (no change in $\mathbf{W}$)

step 1: $\mathbf{W} \cdot \mathbf{X}_1 = [0\ 1\ 1] \cdot [1\ 0\ 0] = 0 \Rightarrow o_1 = 0 = c_1$ (no change in $\mathbf{W}$)

Thus, a complete pass through the training set in the last four pattern presentations shown above results in no changes in $\mathbf{W}$. So the process terminates with the final weight vector $\mathbf{W}_* = [0\ 1\ 1]$ which implements the function specified by the training set.

3 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 *guranteed* 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 ($\mathcal{S}^+$ and $\mathcal{S}^-$ respectively) the perceptron learning algorithm is guaranteed to find a weight vector $\mathbf{W}_*$ such that $\forall \mathbf{X}_k \in \mathcal{S}^+(c_k = 1), \mathbf{W}_* \cdot X_k \geq \delta$ and $\forall \mathbf{X}_k \in \mathcal{S}^-(c_k = 0), \mathbf{W}_* \cdot X_k \leq -\delta$, for some $\delta > 0$, whenever such a $\mathbf{W}_*$ 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 $\mathbf{W}_*$ 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 $\mathbf{W}_* \cdot \mathbf{X}$ (the solution hyperplane) passes through the origin of the pattern space spanned by the co-ordinates $x_1 \cdots x_n$. In other words, $w_{0*} = 0$. 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:

$$\begin{aligned} \text{Let}\mathbf{Z}_k &= \mathbf{X}_k \; if \; c_k = 1 (\mathbf{W}_* \cdot \mathbf{X}_k > \delta \Leftrightarrow \mathbf{W}_* \cdot \mathbf{Z}_k \geq \delta) \\ &= -\mathbf{X}_k \; if \; c_k = 0 (\mathbf{W}_* \cdot \mathbf{X}_k \leq -\delta \Leftrightarrow \mathbf{W}_* \cdot \mathbf{Z}_k \geq \delta) \end{aligned}$$

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

Let $\mathbf{W}_t$ = 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 $\mathbf{W}_{t+1}$ = weight vector after $t+1$ weight updates. Let $\mathbf{W}_0 = [0 \cdots 0]$. (It is easy to extend the proof so that it works for the case where $\mathbf{W}_0$ is initialized to an arbitrary weight vector).

$$\begin{aligned} \mathbf{W}_{t+1} &= \mathbf{W}_t + \eta \underbrace{(c_k - o_k)}_{\text{always 1}} \mathbf{Z}_k \\ &= \mathbf{W}_t + \eta \mathbf{Z}_k \\ \mathbf{W}_* \cdot \mathbf{W}_{t+1} &= \mathbf{W}_* \cdot (\mathbf{W}_t + \eta \mathbf{Z}_k) \\ &= \mathbf{W}_* \cdot \mathbf{W}_t + \eta \underbrace{\mathbf{W}_* \cdot \mathbf{Z}_k}_{\geq \delta} \\ &\geq \mathbf{W}_* \cdot \mathbf{W}_t + \eta \delta \end{aligned}$$

It follows that:

$$\mathbf{W}_* \cdot \mathbf{W}_t \geq t \eta \delta \tag{3.1}$$

Now let us examine how the length of the weight vector changes as a result of weight updates:

$$\begin{aligned} ||\mathbf{W}_{t+1}||^2 &= \mathbf{W}_{t+1} \cdot \mathbf{W}_{t+1} \\ &= (\mathbf{W}_t + \eta \mathbf{Z}_k) \cdot (\mathbf{W}_t + \eta \mathbf{Z}_k) \\ &= \mathbf{W}_t \cdot \mathbf{W}_t + 2\eta \mathbf{W}_t \cdot \mathbf{Z}_k + \eta^2 \mathbf{Z}_k \cdot \mathbf{Z}_k \\ &= ||\mathbf{W}_t||^2 + 2\eta \underbrace{\mathbf{W}_t \cdot \mathbf{Z}_k}_{\leq 0} + \eta^2 ||\mathbf{Z}_k||^2 \end{aligned}$$

Now we note that given any finite training set, the length of the training patterns is bounded. That is, $\forall \mathbf{Z}_k$, $||\mathbf{Z}_k|| \leq L$. It follows that:

$$||\mathbf{W}_t||^2 \leq t\eta^2 L^2 \tag{3.2}$$

We use the above results to show that the number of updates, t, is bounded. Consider: $t\eta\delta \leq \mathbf{W}_* \cdot \mathbf{W_t}$

$t\eta\delta \leq ||\mathbf{W}_*||||\mathbf{W_t}|| \cos\theta \leq ||\mathbf{W}_*||||\mathbf{W_t}||$.

Substituting for $||\mathbf{W}_t||$ we have: $t\eta\delta \leq ||\mathbf{W}_*|| \sqrt{t}\eta L$

$$t \leq \left(\frac{||\mathbf{W}_*||\mathbf{L}}{\delta}\right)^2.$$

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 $||\mathbf{W}_*||$ 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 η is allowed to flucuate arbitrarily over time as long as $0 < \eta_t \leq \mathbf{B}$ where B is the upper bound on the learning rate.

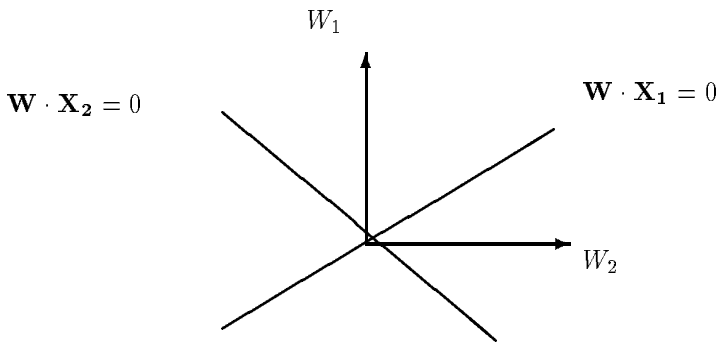
Corollary: Suppose the training patterns are binary:

$\mathbf{X_k} = [X_{0k} \cdots X_{nk}]$ $where X_{ik} = 0$ or 1. Thus we have: $L = \sqrt{n+1}$. If the patterns can be seperated (i.e., $\mathbf{W}_* \cdot \mathbf{Z_k} \geq \delta$ where $\delta > 0$) then there exists a $\mathbf{W}_*$ that seperates patterns with $\delta = 1$. It follows that: $t \leq \left(\frac{||\mathbf{W}_*||\mathbf{L}}{\delta}\right)$ or $\mathrm{t} \leq ||\mathbf{W}_*||^2(n+1)$.

4 VARIANTS OF THE PERCEPTRON ALGORITHM

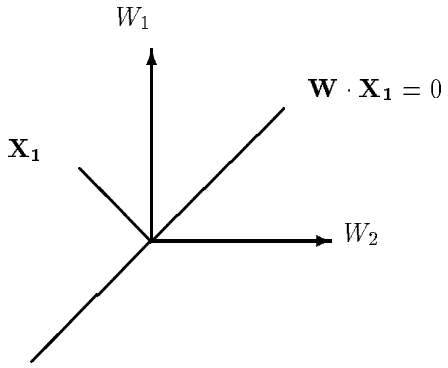
Perceptron algorithm offers a simple, provably convergent procedure for finding a weight vector that corresponds to a separating hyperplane whenever the training set is *linearly separable*. A number of variants of the perceptron algorithm that converge faster to a solution than the simple algorithm 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 $\mathbf{W_p} = [W_{0p} \cdots W_{np}]$ can be thought of as a point in an $(n+1)$-dimensional "weight space" with coordinate axes $(w_0 \cdots w_n)$. In weight space, a pattern vector $\mathbf{X_k} = [X_{0k} \cdots X_{nk}]$ defines a hyperplane given by $\mathbf{W} \cdot \mathbf{X_k} = 0$ where $\mathbf{W} = [w_0 \cdots w_n]$.

Thus, in the weight space represenatation 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 accross 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 η in the perceptron learning rule controls how far $\mathbf{W}$ is moved.

The perceptron algorithm discussed earlier changes $\mathbf{W}_t$ as follows:
$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t \pm \eta \mathbf{X}_k$ (where $\eta > 0$ is constant).

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

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

We then choose η_t such that

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

So

$\eta_t = \left\lceil \frac{|\mathbf{W}_t \cdot \mathbf{X}_k|}{\mathbf{X}_k \cdot \mathbf{X}_k} \right\rceil$

This gives the so-called absolute correction rule.

Alternatively, we could choose η 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 refered to [Nilsson, 1965].

5 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.

Some of these topics will be addressed in subsequent chapters.