Chapter 2

Single Layer Feedforward Networks

Perceptrons: These are the oldest and simplest artificial neural networks known, defined by Rosenblatt (1958) to be "a machine that learns, using examples, to assign input vectors (samples) to different classes, using a linear function of the inputs." Minsky and Papert (1969) describe the perceptron as: "A stochastic gradient-descent algorithm that attempts to linearly separate a set of n-dimensional training data."

We use the word *perceptron* in the former sense as a machine, and refer explicitly to the "perceptron learning algorithm".

In its simplest form, a perceptron has a single output

whose values determine to which of two classes each input pattern belongs. The node applies a step function to the net weighted sum of its inputs. The input pattern is considered to belong to one class or the other depending on whether the node output is 0 or 1.

Example Consider 2-dimensional samples $(0,0)$, $(0,1)$, $(1,0)$, $(-1,-1)$ that belong to one class, and samples $(2.1,0)$, $(0,-2.5)$, $(1.6,-1.6)$ that belong to another class. These classes are linearly separable as shown. A

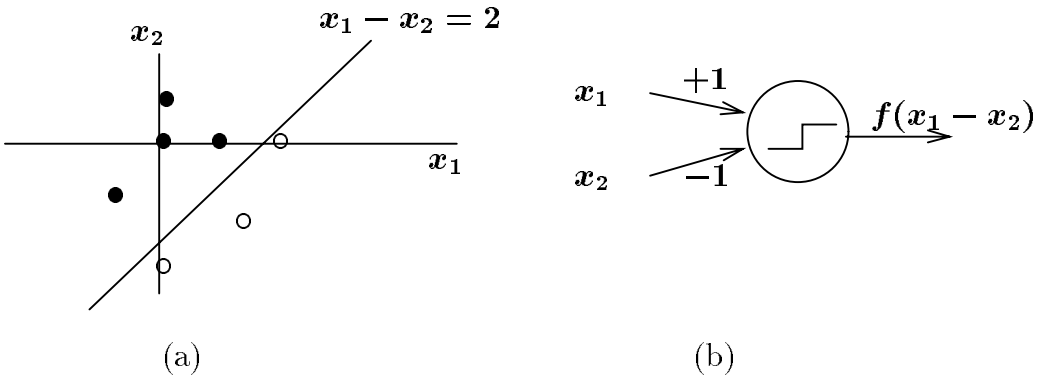

Figure 2.1: **Samples from two linearly separable classes, and the associated perceptron whose output $= 1$ if $(x_1 - x_2) > 2$ and 0 otherwise.**

perceptron that separates samples of opposite classes is also shown.

An artificial input whose value is always equal to 1 allows us to replace the threshold by a weight from an input to the node, and makes it easier to write learning algorithms.

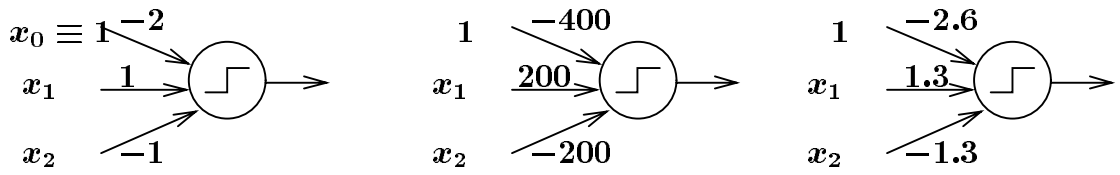

Figure 2.2: **Equivalent perceptrons that use a constant input $= 1$ instead of a threshold.**

* .....*

Linear Separability **For 2-dimensional inputs, if there exists a line $w_0 + w_1 x_1 + w_2 x_2 = 0$ that separates all samples of one class from the other class, then an appropriate perceptron with weights w_0, w_1, w_2 for the connections from inputs $1, x_1, x_2$, respectively can be derived from the equation of the separating line. Such classification problems are said to be "linearly separable," i.e., separable by a linear combination of inputs.**

Conversely, if there is a simple perceptron with weights w_0, w_1, w_2 for the connections from inputs $1, x_1, x_2$, respectively that can separate samples of two classes, then we can obtain the equation $w_0 + w_1 x_1 + w_2 x_2 = 0$ of a line that separates samples of the two classes.

In general, linear separability is equivalent to the existence of a hyperplane dividing $n-$dimensional space into two regions $w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \geq 0$ and $w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < 0$. Several researchers

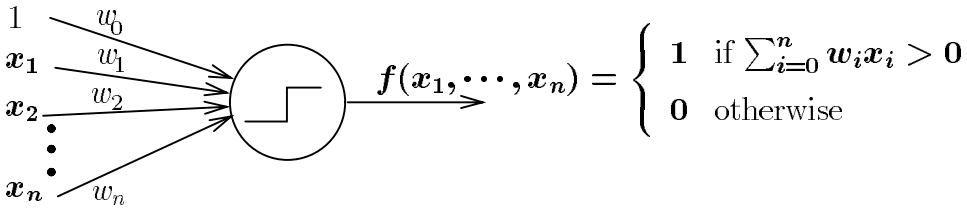

Figure 2.3: **Generic perceptron for n-dimensional input space, implementing the hyperplane $\sum_{i=0}^{n} w_i x_i = 0$.**

have reported better performance by using perceptron output values $\in \{-1, 1\}$ instead of $\{0, 1\}$. We use this criterion in the following development.

*** .....***

Perceptron Training Algorithm

Algorithm Perceptron:

Start with a randomly chosen weight vector w_0;

Let $k = 1$;

while some input vectors remain misclassified, do

Let i_j be a misclassified input vector;

Let $x_k = class(i_j).i_j$, implying that $w_{k-1} \cdot x_k < 0$;

Update the weight vector to $w_k = w_{k-1} + \eta x_k$;

Increment k;

end-while;

Figure 2.4: Perceptron Training Algorithm

* .....*

Justification: Let w be the weight vector (including the threshold) and i be the input vector (including the dummy input constant $i_0 = 1$) for a given iteration of the while loop. If i belongs to class for which the desired node output is -1 but $w \cdot i > 0$, then the weight vector needs to be modified to $w + \Delta w$ so that $(w+\Delta w) \cdot i < w \cdot i$; so that i would have a better chance of

correct classification in the following iteration. Choose

$$\Delta w = -\eta i$$

where $\eta > 0$ is a constant. This will work since

$$(w + \Delta w) \cdot i = (w - \eta i) \cdot i = w \cdot i - \eta i \cdot i < w \cdot i,$$

because $i \cdot i > 0$.

Similarly, if $w \cdot i < 0$ when the desired output value is 1, the weight vector needs to be modified so that $(w + \Delta w) \cdot i > w \cdot i$, which can be done by choosing Δw to be ηi for $\eta > 0$, since $(w + \eta i) \cdot i = w \cdot i + \eta i \cdot i > w \cdot i$.

This is the essential rationale for the perceptron training algorithm.

Some important questions;

1. How long should we execute this procedure, i.e., what is the termination criterion if the given samples are not linearly separable?

2. What is the appropriate choice for the learning rate η?

3. How can the perceptron training algorithm be applied to problems in which the inputs are non-numeric values (e.g., values of the attribute "color" instead of numbers)?

4. Is there a guarantee that the training algorithm will always succeed whenever the samples are linearly separable?

5. Are there useful modifications of the perceptron training algorithm that work reasonably well when samples are not linearly separable?

* .....*

Termination Criterion: "Halt when the goal is achieved." The goal is the correct classification of all samples, assuming these are linearly separable. So the algorithm continues to execute until all samples are correctly classified. Termination is assured if η is sufficiently small, and samples are linearly separable. (Perceptron Convergence Theorem)

The procedure will run indefinitely if the given samples are not linearly separable or if the choice of η is inappropriate. So, if the number of misclassifications has not changed in a large number of steps, the samples may not be linearly separable.
Else, if the problem is with the choice of η, then experimenting with a different choice of η may yield improvement.

*** .....***

Choice of learning rate: If η is too large then the components of $\Delta w = \pm\eta\, x$ can have very large magnitudes, (assuming components of x are not infinitesimally small). Consequently, each weight update swings perceptron outputs completely in one direction, so that the perceptron now considers all samples to be in the same class as the most recent one. This effect is reversed when a new sample of the other class is presented.

If $\eta \approx 0$, the change in weights in each step is going to be infinitesimally small, assuming components of

x are not very large in magnitude. So an extremely large number of such (infinitesimally small) changes to weights are needed.

A common choice is $\eta = 1$. To ensure that the sample x is correctly classified following the weight change, we need $\Delta w = \pm \eta x$ such that $(w + \Delta w) \cdot x$ and $x \cdot x$ have opposite sign, i.e.,

$$|\Delta w \cdot x| > |w \cdot x| \longleftrightarrow \eta |x \cdot x| > |w \cdot x| \longleftrightarrow \eta > \frac{|w \cdot x|}{|x \cdot x|}.$$

* .....*

Non-numeric inputs: In some problems, the input dimensions are non-numeric; their values do not have any inherent order. For instance, the input may be "color," with values in{red, blue, green, yellow}. Depicted as points on an axis, e.g., red–blue–green–yellow, the resulting figure it gives erroneous implicit inter-relationships, e.g., that "red" is closer to "blue" than to "green" etc..

The simplest alternative is to generate four new dimensions ("red," "blue," "green" and "yellow") instead of the single "color" dimension, and replace each origi-

nal attribute-value pair by a binary vector with one component corresponding to each color. For instance, color = "green" is represented by the input vector (0,0,1,0) standing for red=0, blue=0, green=1, yellow=0. In the general case, if an attribute can take one of n different values, then n new dimensions are obtained.

*** .....***

Guarantee of Success:

Theorem 2.1 Given training samples from two linearly separable classes, the perceptron training algorithm terminates after a finite number of steps, and correctly classifies all elements of the training set.

Proof: Let $\eta = 1$. After k steps,

$$w_k = w_0 + x_1 + x_2 + \ldots + x_k. \tag{2.1}$$

Linear separability implies that there is some 'perfect' separator w^*, with $sgn(w^* \cdot i_k) = class(i_k)$. Hence

$$w^* \cdot w_k = w^* \cdot w_0 + w^* \cdot x_1 + \ldots + w^* \cdot x_k.$$

For each input vector i_j, $w^* \cdot i_j$ has the same sign as

$class(i_j)$. But $x = class(i_j)i_j$, hence

$$w^* \cdot x = w^* \cdot (class(i_j)i_j) > 0.$$

Therefore, there exists an $\varepsilon > 0$ such that $w^* \cdot x_i > \varepsilon$ for every member x_i of the training sequence. Hence

$$w^* \cdot w_k > w^* \cdot w_0 + k\varepsilon. \tag{2.2}$$

By the Cauchy–Schwartz inequality,

$$\|w^* \cdot w_k\|^2 \leq \|w^*\|^2 \|w_k\|^2. \tag{2.3}$$

Assume $\|w^*\| = 1$, since $w^*/\|w^*\|$ also correctly separates the same samples. Hence

$$\|w_k\|^2 > (w_0 \cdot w^* + k\varepsilon)^2. \tag{2.4}$$

But since $w_j = w_{j-1} + x_j$, the following upper bound can be obtained for this vector's squared length:

$$\begin{aligned} \|w_j\|^2 &= w_j \cdot w_j \\ &= w_{j-1} \cdot w_{j-1} + 2w_{j-1} \cdot x_j + x_j \cdot x_j \\ &= \|w_{j-1}\|^2 + 2w_{j-1} \cdot x_j + \|x_j\|^2. \end{aligned}$$

Since $w_{j-1} \cdot x_j < 0$ whenever a weight change is required by the algorithm, we have

$$\|w_j\|^2 - \|w_{j-1}\|^2 < \|x_j\|^2. \tag{2.5}$$

Summation over $j = 1, \ldots, k$ gives

$$\|w_k\|^2 - \|w_0\|^2 < k \max \|x_j\|^2. \tag{2.6}$$

Combining this with the inequality 2.4, we have

$$(w_0 \cdot w^* + k\varepsilon)^2 < \|w_k\|^2 < \|w_0\|^2 + k \max \|x_j\|^2. \tag{2.7}$$

These inequalities establish two conflicting requirements on the size of $\|w_k\|^2$: the lower bound increases at the rate of k^2 whereas the upper bound increases at linear rate k. But any quadratic function with positive coefficient for k^2 will eventually exceed a linear function. In other words, there will be some finite k for which

$$(w_0 \cdot w^* + k\varepsilon)^2 > \|w_0\|^2 + k \max \|x_j\|^2. \tag{2.8}$$

Hence k cannot increase without bound, hence the algorithm must terminate.

* .....*

Modifications Nothing useful can be assumed about the behavior of the Perceptron training algorithm when applied to classification problems in which the classes are not linearly separable. The "Pocket" and "Least Mean Squares" (LMS) algorithms attempt to achieve robust classification when the two classes are not linearly separable.

The pocket algorithm, a useful modification of the perceptron training algorithm, whose weight change mechanism is exactly the same as that of the perceptron. In addition, the pocket algorithm identifies the weight vector with the longest unchanged run as the best solution among those examined so far and separately stores (in a "pocket") the best solution explored so far, as well as the length of the run associated with it. The contents of the pocket are replaced whenever a new weight vector with a longer successful run is found.

The goal of this algorithm is to find the "optimal" set

of weights, i.e., the linear classifier for which the number of misclassifications is the least. If the number of training samples is finite, and samples are presented randomly, then running the Pocket algorithm long enough will ensure that the optimal set of weights is discovered with high probability.

A lucky run of several successes may allow a poor solution to replace a better solution in the pocket. To avoid this, the Pocket algorithm with ratchet ensures that the pocket weights always "ratchet up": a set of weights w_1 in the pocket is replaced by w_2 with longer successful run only after testing (on all training samples) whether w_2 does correctly classify a greater number of samples than w_1.

The Pocket algorithm gives good results, although there is no guarantee of reaching the optimal weight vector in a reasonable number of iterations.

* .....*

Algorithm Pocket;

Start with randomly chosen weight vector w_0; $k = 0$;

$best\_run\_length = 0$; $cur\_run\_length = 0$;

while $k <$ max. allowed no. of presentations, do

Increment k;

Let i be a randomly selected input vector;

Let $x = class(i).i$;

If $w_{k-1} \cdot x_k > 0$,

then increment cur_run_length;

else

if ($best\_run\_length < cur\_run\_length$) and

if w_{k-1} misclassifies fewer samples

than current pocket contents

then replace pocket contents by w_{k-1} and

$best\_run\_length$ by cur_run_length;

end-if

Update the weight vector to $w_k = w_{k-1} + \eta x_k$;

end-If

end-while.

Figure 2.5: Pocket Algorithm with ratchet check.

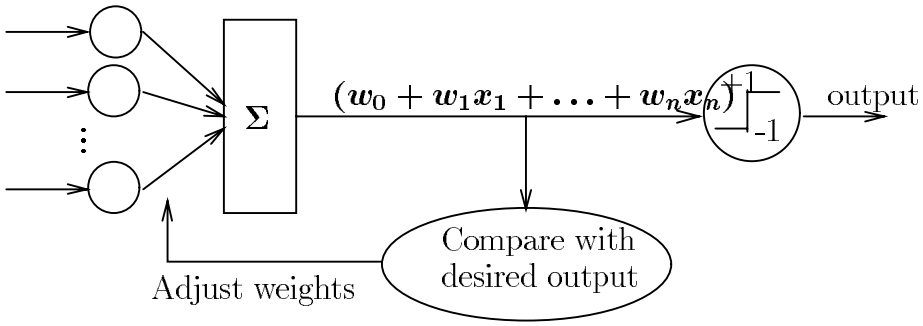

Figure 2.6: **Adaptive linear element (Adaline).**

ADALINES Instead of aiming to reduce the number of misclassifications, one may choose to minimize the mean squared error (MSE). An "adaptive linear element" or Adaline, proposed by Widrow (1959, 1960), modifies weights in such a way as to diminish the MSE at every iteration.

Adaline's weight change rule: Let $i_j = (i_{0,j}, i_{1,j} \ldots, i_{n,j})$ be an input vector with desired output value d_j, (with $i_{0,j} \equiv 1$). Let $w = (w_0, \ldots, w_n)$ be the current weight vector and $\text{net}_j = \sum_l w_l i_{l,j}$ be the net input to the Adaline. Then squared error $E = (d_j - \text{net}_j)^2$, &

$$\begin{aligned} \frac{\partial E}{\partial w_k} &= 2(d_j - \text{net}_j)\frac{\partial}{\partial w_k}\,(\text{-net}_j) \\ &= -2(d_j - \text{net}_j)\; i_{k,j}. \end{aligned}$$

Gradient descent suggests Δw_k to be a negative multiple of $\partial E/\partial w_k$, leading to the weight update rule

$$\Delta w_k = \eta \left(d_j - \sum_l w_l i_{l,j} \right) i_{k,j} i = \eta (d_j - net_j) i_{k,j}.$$

Algorithm LMS-Adaline;

Start with a randomly chosen weight vector w_0;

Let $k = 1$;

while MSE is unsatisfactory and

computational bounds are not exceeded, do

Let i be an input vector

(chosen randomly or in some sequence)

for which d is the desired output value;

Update the weight vector to

$$w_k = w_{k-1} + \eta (d - w_{k-1} \cdot i) i$$

Increment k;

end-while.

Figure 2.7: LMS (Adaline) Training Algorithm.

α–LMS rule: weight change magnitude is independent of the magnitude of the input vector:

$$\Delta w = \eta \ (d_j - \text{net}_j) \ \frac{i_j}{||i_j||},$$

where $\text{net}_j = \sum_{l=0}^{n} w_i i_{l,j}$.

The threshold or step function is invoked whenever the Adaline needs to generate an output for some input pattern. As in the perceptron, the output of the Adaline for classification problems is:

$$o_j = \begin{cases} 1 & \text{if } \sum \ w_l \ i_{l,j} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Adalines can also be used for function approximation tasks. The training part of the Adaline (before applying the step function) resembles that of statistical "linear regression," where a set of $(n+1)$ linear equations

$$\sum_{j=1}^{P} (d_j - (w_0 + w_1 i_{1,j} + \cdots + w_n, i_{n,j}))i_{i,j} = 0$$

must be solved for the unknown values $w_0, \ldots, w_n$; here P denotes the number of patterns.

* .....*

Multiclass discrimination The number of classes > 2 in

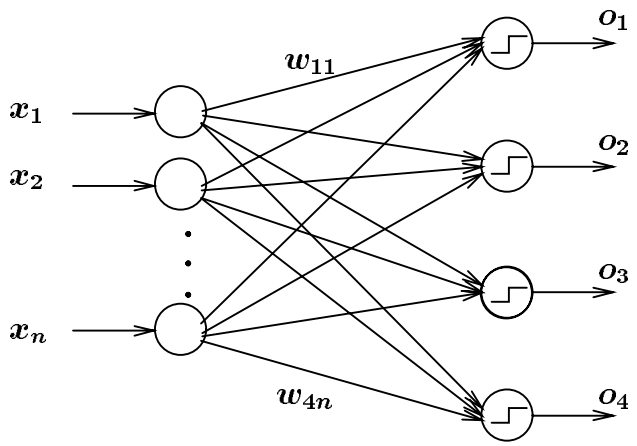

Figure 2.8: **4-node perceptron layer to solve 4-class problem**

many problems (e.g., character recognition), requiring a layer of > 2 perceptrons or Adalines. Each perceptron or Adaline can be trained separately to distinguish one class from all the rest. An input vector is assigned to the ith class iff output $o_i = 1$, and $o_k = 0$, for $k \neq i$.

If node output values can lie between 0 and 1, a "maximum-selector" can be used to select the highest-valued output that exceeds a prespecified threshold.

$m - 1$ nodes are sufficient for m-class problems in which one of the classes is an "otherwise" case.