Dr. George Karraz, Ph. D.

Neural Networks
Lecture V

Simple Neural Nets For Pattern Classification

Dr. George Karraz, Ph. D.

Contents

1. Introduction
2. Architecture
3. Biases and Thresholds
4. Hebb Nets
5. Perceptron.
6. Adaline.

1. Introduction

- One of the simplest tasks that neural nets can be trained to perform is pattern classification.
- In pattern classification problems, each input vector (pattern) belongs, or does not belong, to a particular class or category.
- We assume we have a set of training patterns for which the correct classification is known.
- The output represents membership in the class with a response of 1, a response of -1 or 0 indicates that pattern is not member of the class
- For a single Layer nets, extension to the more general case in which each pattern may or may not belong to any of several classes is possible. In such case there is an output unit for each class.

2. Architecture

- The basic architecture of the simplest possible neural networks that perform pattern classification consists of a layer of input units (as many units as the patterns to be classified have components) and a single output unit.
- This allows classification of vectors , which are n-tuples, but consider membership in only one category.

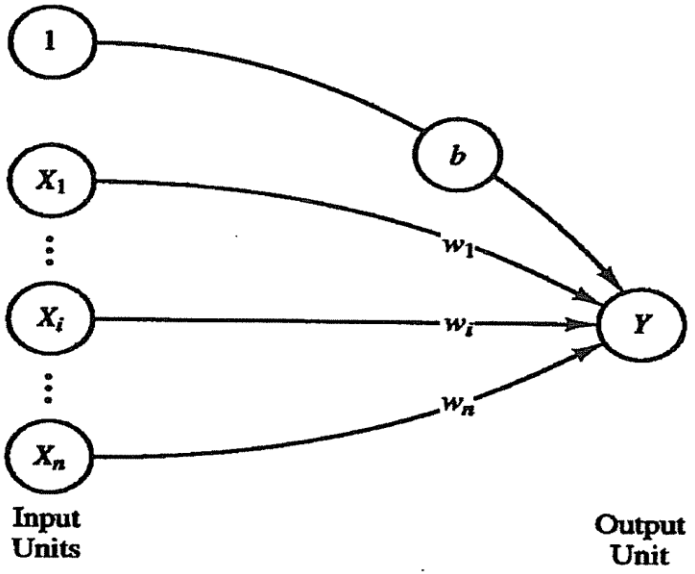


Figure 2.1 A single-layer net for pattern classification

3. Biases and Thresholds

- A bias acts exactly as a weight on connection from a unit whose activation is always 1. Increasing the bias increases the net input to the unit. If a bias is included, the activation function is typically taken to be:

Where

$$f(\text{net}) = \begin{cases} 1 & \text{if net} \geq 0; \\ -1 & \text{if net} < 0; \end{cases}$$

$$\text{net} = b + \sum_i x_i w_i.$$

Some authors do not use a bias weight, but instead use a fixed threshold for the activation function:

Where

$$f(\text{net}) = \begin{cases} 1 & \text{if net} \geq \theta; \\ -1 & \text{if net} < \theta; \end{cases}$$

$$\text{net} = \sum_i x_i w_i.$$

3. Biases and Thresholds

- Example: we consider the separation of input space into regions where the response of the net is positive and regions where response is negative. To facilitate a graphical display of the relationships, we illustrate the ideas for an input with two components while the output has only one component. As we illustrate in Figure 2.2

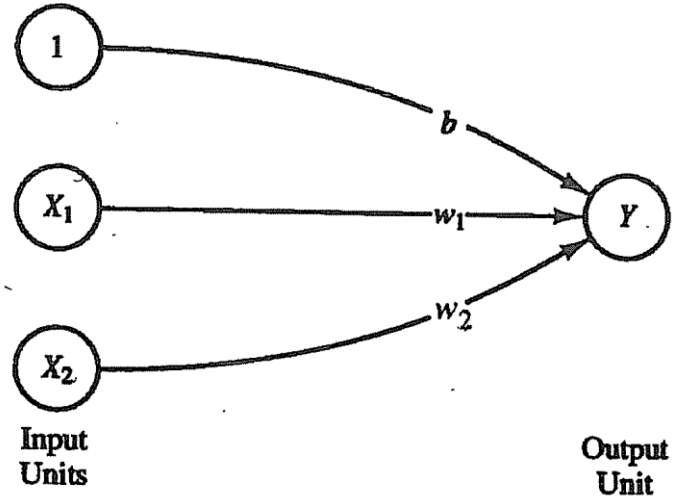


Figure 2.2 A single-layer net for pattern classification

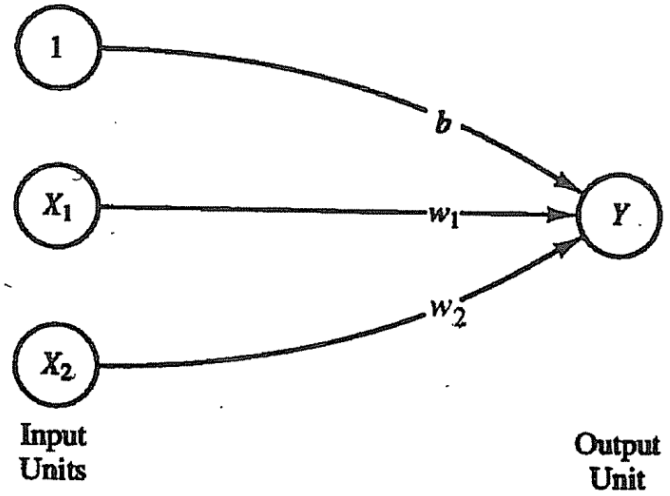
3. Biases and Thresholds

- The boundary between the values of x1and x2for which the net gives a positive response and the values for which it gives a negative response is the separating line

$$b + x_1 w_1 + x_2 w_2 = 0,$$

Assume that w2 ≠ 0

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$



- The requirement for a positive response from the output unit that $b + x_1 w_1 + x_2 w_2$ be greater than 0. During training values of w1,w2, and b are determined so that the net will have the correct response for training data

4. Hebb Nets

- The earliest and simplest learning rule.
- Hebb proposed that learning occurs by modification of synapse strengths (weights) in manner such that if two interconnected neurons are both "on" at the same time, then the weight between those neurons should be increased. The original statement only talks about neurons firing at the same time. A stronger form of learning occurs if we also increase the weights if both neurons are "off' at the same time.
- We shall refer to a single-layer (feed forward) neural net trained using the Hebb rule as a *Hebb net*
- If data are represented in bipolar form, it is easy to express the desired weight update as:

$$w_i(\text{new}) = w_i(\text{old}) + x_i y.$$

4. Hebb Nets

- Algorithm:

Step 0. Initialize all weights:

$$w_i = 0 \qquad (i = 1 \text{ to } n).$$

Step 1. For each input training vector and target output pair, s : t, do steps 2–4.

Step 2. Set activations for input units:

$$x_i = s_i \qquad (i = 1 \text{ to } n).$$

Step 3. Set activation for output unit:

$$y = t.$$

Step 4. Adjust the weights for

$$w_i(\text{new}) = w_i(\text{old}) + x_i y \qquad (i = 1 \text{ to } n).$$

Adjust the bias:

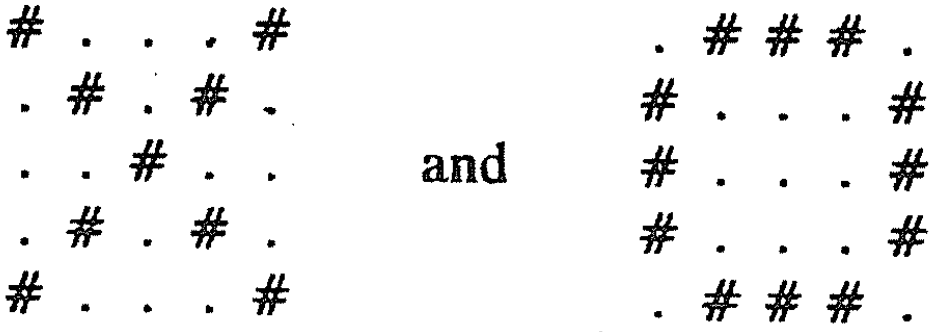
$$b(\text{new}) = b(\text{old}) + y.$$

Not that the bias is adjusted exactly like a weight from a 'unit' whose output signal is always 1. the weight update can also be expressed in vector form: $\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + \mathbf{x}y$

$$\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + \Delta\mathbf{w}$$

4. Hebb Nets

- **Example:** A simple example of using the Hebb rule for character recognition involves training the net to distinguish between the pattern 'X' and the pattern 'O' the patterns can be represented as:



Pattern 1 Pattern 2

4. Hebb Nets

- **Example Solution :**

1. we need firstly convert the patterns to input vectors (assigning each "#" the value 1, and each " ." the value 0.
2. To convert from the two dimensional pattern to an input vector, we simply concatenate the rows,
3. Pattern 1: 1 −1 −1 −1 1, −1 1 −1 1 −1, ...etc.
4. Pattern 2: −1 1 1 1 −1, 1 −1 −1 −1 1, ...etc.
5. The correct response for the first pattern "on" or "+1", so the weights after presenting the first pattern are simply the input pattern. The bias weight after presenting this is +1. The correct response for the second pattern "off" or "-1", so the weights change to the complement values of second pattern . In this case the bias weight is -1.

4. Hebb Nets

- **Example Solution**

6. The final weights represent the sum of the weights in the two previous cases:

   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

7. The net input (for any input pattern) is the dot product of the input pattern with the weight vector. For the first training vector the net input is 42, so the response is positive. For the second training pattern , the net input is - 42 so the response is clearly negative, as desired.

5. Perceptron

- The perceptron learning rule is a more powerful learning rule than the Hebb rule. Under suitable assumptions, its iterative learning procedure can be proved to converge to the correct weights, the weights that allow the net to produce the correct output value for each of the training input patterns.
- Some preceptron were self –organizing, most were trained.
- Typically, the original perceptrons had three types of units: sensory units, associator units, and a response unit- forming an approximate model of retina.

5. Perceptron

- Simple preceptron for pattern classification: the goal of the net is to classify each input pattern as belonging, or not belonging, to a particular class.
- Belonging is signified by output unit giving the response of +1, where not belonging is indicated by a response of -1, see Figure 2.3

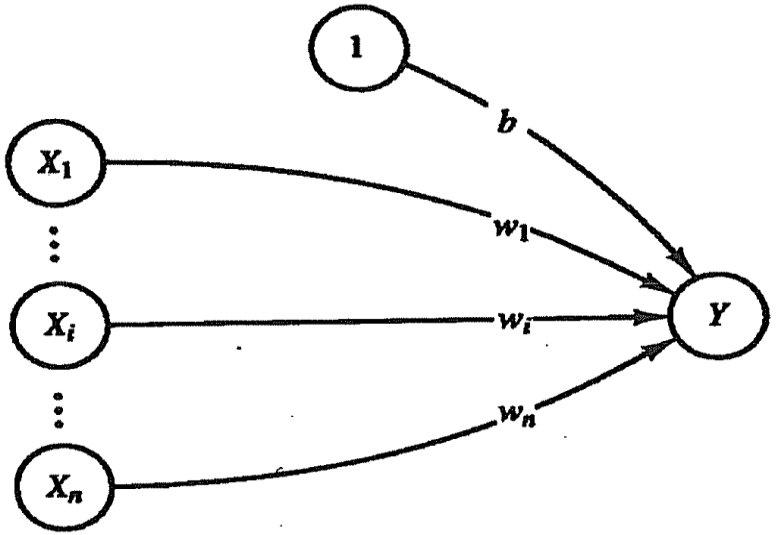


Figure 2.3 Perceptron to perform a single classification

- The net is trained to perform this classification by the iterative technique as the follow (θ is a threshold):

Step 0. Initialize weights and bias.
(For simplicity, set weights and bias to zero.)
Set learning rate α ($0 < \alpha \leq 1$).
(For simplicity, α can be set to 1.)

Step 1. While stopping condition is false, do Steps 2–6.

Step 2. For each training pair s:t, do Steps 3–5.

Step 3. Set activations of input units:

$$x_i = s_i.$$

Step 4. Compute response of output unit:

$$y\_in = b + \sum_i x_i w_i;$$

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \leq y\_in \leq \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

Step 5. Update weights and bias if an error occurred for this pattern.

If $y \neq t$,

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i,$$

$$b(\text{new}) = b(\text{old}) + \alpha t.$$

else

$$w_i(\text{new}) = w_i(\text{old}),$$

$$b(\text{new}) = b(\text{old}).$$

Step 6. Test stopping condition:
If no weights changed in Step 2, stop; else, continue.

5. Perceptron

- Perceptron Learning Rule Convergence **Theorem**

Given a finite set of P input training vectors

$$\mathbf{x}(p), \quad p = 1, \ldots, P,$$

each with an associated target value

$$t(p), \quad p = 1, \ldots, P,$$

which is either $+1$ or -1, and an activation function $y = f(y\_in)$, where

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta, \end{cases}$$

the weights are updated as follows:

If $y \neq t$, then

$$\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + t\mathbf{x};$$

else

no change in the weights.

If there is a weight vector W* such that $f(x(p).W^*) = t(p)$ for all p, then for any starting vector W, the preceptron learning rule will converge to a weight vector (not necessarily unique and not necessarily W*) that gives the correct response for all training patterns, and it will do so in a finite number of steps.

6. ADALINE

- Is a single unit (neuron) that receives input from several units. It also receives input from a "unit" whose signal is always +1, in order for the bias weight to be trained by the same process (the delta rule).

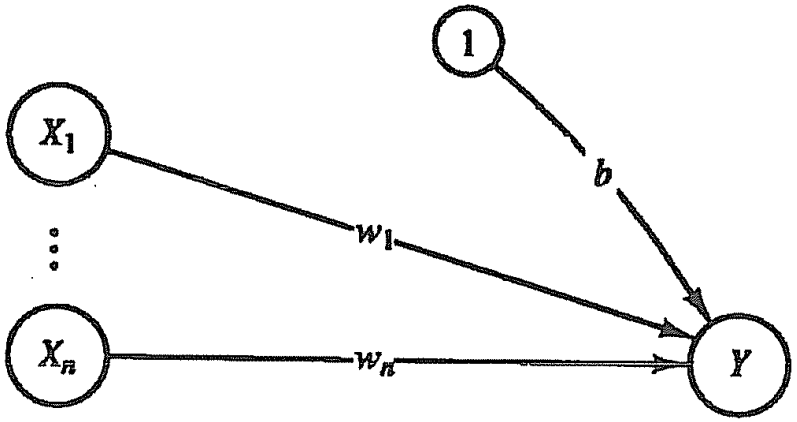


Figure 2.4 Adaline

6. ADALINE

- Algorithm (delta)

Step 0. Initialize weights.
(Small random values are usually used.)
Set learning rate α.

Step 1. While stopping condition is false, do Steps 2–6.

Step 2. For each bipolar training pair s:t, do Steps 3–5.

Step 3. Set activations of input units, $i = 1, \ldots, n$:

$$x_i = s_i.$$

Step 4. Compute net input to output unit:

$$y\_in = b + \sum_i x_i w_i.$$

Step 5. Update bias and weights, $i = 1, \ldots, n$:

$$b(\text{new}) = b(\text{old}) + \alpha(t - y\_in).$$

$$w_i(\text{new}) = w_i(\text{old}) + \alpha(t - y\_in)x_i.$$

Step 6. Test for stopping condition:
If the largest weight change that occurred in Step 2 is smaller than a specified tolerance, then stop; otherwise continue.

6. ADALINE

Setting the learning rate to a suitable value requires some care. According to Hecht-Nielsen (1990), an upper bound for its value can be found from the largest eigenvalue of the correlation matrix R of the input (row) vectors $\mathbf{x}(p)$:

$$R = \frac{1}{P} \sum_{p=1}^{P} \mathbf{x}(p)^T \mathbf{x}(p),$$

namely,

$$\alpha < \text{one-half the largest eigenvalue of } R.$$

However, since R does not need to be calculated to compute the weight updates, it is common simply to take a small value for α (such as $\alpha = .1$) initially. If too large a value is chosen, the learning process will not converge; if too small a value is chosen, learning will be extremely slow [Hecht-Nielsen, 1990]. The choice of learning rate and methods of modifying it are considered further in Chapter 6. For a single neuron, a practical range for the learning rate α is $0.1 \leq n\alpha \leq 1.0$, where n is the number of input units [Widrow, Winter & Baxter, 1988].

The proof of the convergence of the ADALINE training process is essentially contained in the derivation of the delta rule, which is given in Section 2.4.4.