ML Exercise 1 : Due October 28.

3 The Perceptron

Perceptron Training

A key neural network problem has traditionally been the problem of separating patterns into two categories. The first neurode type that was applied to this was proposed by Warren S. McCulloch and Walter Pitts in 1943. They suggested the simple device shown in figure 3.1.

The McCulloch-Pitts neurode is still the basis for most neural networks today. The idea is extremely simple. The neurode computes the weighted sum of the input signals and compares that net weighted input to a threshold value *T*. If the net input is greater than or equal to the threshold, the neurode outputs +1; if not, it outputs –1. This McCulloch-Pitts neurode is virtually indistinguishable from the neurodes used in the perceptron and the adaline and is the clear ancestor of nearly all neurodes in current neural networks. As it is encountered today, this neurode uses the transfer function listed below:

$$I = \sum_{i=1}^{n} w_i x_i$$

$$y = \begin{pmatrix} +1, & \text{if } I \geq T \\ -1, & \text{if } I < T \end{pmatrix}$$

Here, I is the net weighted input to the neurode, w_i and x_i are the components of the weight vector and the input vector, respectively, and y is the output of the perceptron. The threshold value T is the minimum activity required for the neurode to generate a positive output. For discussion purposes here, $T = 0$.

If McCulloch and Pitts defined a simple model of the neuron that has become a standard today, it took Frank Rosenblatt to turn their neurode into the first trainable neural network. His perceptron training algorithm, introduced in 1958, provided the first procedure that could be used to allow a network to learn a task. And the task the perceptron learns is the classic one of separating patterns into two categories.

Rosenblatt's training law is quite simple:

The pattern elements x_1 and x_2 are the components of vector **x**.

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \beta y \mathbf{x}$$

$$\beta = \begin{cases} +1, & \text{if the perceptron's answer is correct} \\ \text{and} & \\ -1, & \text{if the perceptron's answer is wrong} \end{cases}$$

$$y = \text{the perceptron's answer}$$

The weight vector in this expression is represented by **w** and the input vector pattern by **x**.

The perceptron training algorithm is easiest to understand in a graphical context, so it is described as a vector training procedure in the following exercise. Listing 3.1 also provides a pseudocode description of the algorithm.

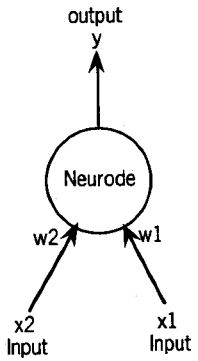


Figure 3.1 **The McCulloch-Pitts neurode, shown here with two input signals.**

Listing 3.1 *The Perceptron Training Algorithm*

```
for each pattern in the training set
{
    apply the next pattern to the perceptron
    record the perceptron's response (either +1 or -1)
    if the perceptron's answer is correct
    {
        and the answer was +1, then
            the new weight vector = the old weight vector + the input pattern vector
        and the answer was -1, then
            the new weight vector = the old weight vector - the input pattern vector
    }
    if the perceptron's answer is incorrect
    {
        and the perceptron's answer was +1, then
            the new weight vector = the old weight vector - the input pattern vector
        and the perceptron's answer was -1, then
            the new weight vector = the old weight vector + the input pattern vector
    }
} /* end for each pattern in the training set */
```

Exercise 3.1

In the perceptron, the initial weight vector can either be a random vector as used here, or the zero vector with all components equal to zero.

1. Figure 3.2 displays a collection of data points plotted on a graph. There are two points listed in the A category with a +1 output, and two listed in the B category with a –1 output. You will build a perceptron (on paper) that can correctly distinguish between the A and B data points. The initial weight vector $\mathbf{w}_0$ is also graphed and its components given as (–0.6, 0.8). The first few steps in training the perceptron are shown in detail here to use as an example. Assume the threshold is 0 for this example.

   a. Apply A_1 to the perceptron and compute the net input I. This means taking the weighted sum of the input pattern elements. Each pattern element is multiplied by the corresponding perceptron weight and the resulting products added to yield I. The input pattern A_1 has $x_1 = 0.3$ and $x_2 = 0.7$. Thus the computation for I is:

   $I = w_1 x_1 + w_2 x_2$

   $I = (-0.6)\ (0.3) + (0.8)\ (0.7)$

   $I = -0.18 + 0.56 = 0.38$

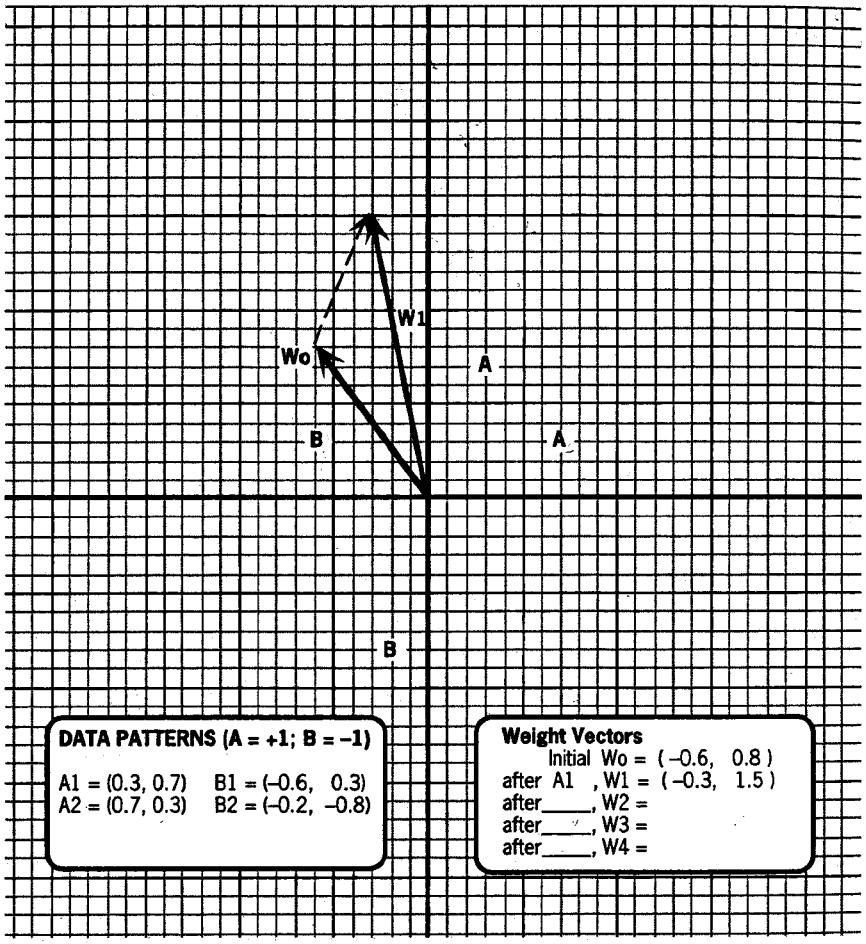


Figure 3.2 **Use the area above to train the perceptron graphically. The first input pattern's weight change (for pattern A_1) has been done for you.**

The change in the weight vector, called the "delta vector," is shown by a dashed arrow in figure 3.2. It is constructed by placing the tail of the $\mathbf{A}_1$ vector at the head of the $\mathbf{w}_0$ vector and drawing an arrow from the origin to the tip of $\mathbf{A}_1$.

Since $I > 0$, the output = +1

The desired output for this pattern is +1. Thus, the new weight vector is the old weight vector plus the input vector.

$$\mathbf{w}_1 = \mathbf{w}_0 + \mathbf{A}_1$$

Vectors are added by summing their corresponding components. The new x and y components for the weight vector $\mathbf{w}_1$ are therefore

$$w_x = -0.6 + 0.3 = -0.3$$
$$w_y = 0.8 + 0.7 = 1.5$$

Plot the change in the weight vector from $\mathbf{w}_0$ to $\mathbf{w}_1$ by graphically adding the A_1 pattern vector to the $\mathbf{w}_0$ vector. Confirm that the graphical result is approximately the $\mathbf{w}_1$ vector computed above.

b. Apply B_1 to the perceptron and compute the net input I:
Net input I =

Perceptron's output =

Is this correct? (The correct output for any B pattern is –1.)

Is the B_1 vector added to or subtracted from the current weight vector to make $\mathbf{w}_2$?

Draw your result on the graph in figure 3.2.

c. Continue training the perceptron with patterns A_2 and B_2. For each input pattern, add the change in the weight vector to the current weight vector, and draw the result on the graph of figure 3.2.
Applying A_2:
Net input I:

Perceptron's output:

Is this correct?

Should vector A_2 be added to or subtracted from the current weight vector?

What is the new weight vector?

Applying B_2:
Net input I:

Perceptron's output:

Do not change the weight vector when checking the perceptron's categorization.

Is this correct?

Should vector B_2 be added or subtracted from the current weight vector?

What is the final weight vector after all patterns have been trained?

d. Using this final weight vector, confirm that the perceptron now correctly classifies each of the four training patterns by computing the output of the perceptron for each pattern. Does it generate the correct +1 or –1 response for each case?

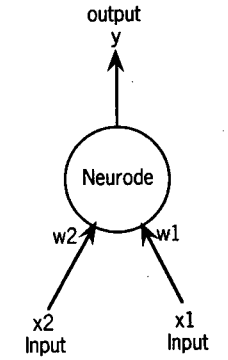
e. Choose several more A and B data patterns and demonstrate to yourself that the perceptron will correctly classify them. Indicate on the graph in figure 3.2 where you think the dividing line is between A and B patterns. Can the perceptron correctly classify all possible patterns in the areas you have drawn?

How did you decide where to put the separator between the two categories?

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \beta\, y\, \mathbf{x}$$

β = { +1, if the perceptron's answer is correct; and; –1, if the perceptron's answer is wrong }

y = the perceptron's answer



Summary of the Perceptron