Neural Networks I: The Perceptron

Vinay Bhaip*

December 2020

1 Introduction

Neural networks are at the foundation of machine learning and are the basis of complex models like Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Generative Adversarial Networks (GANs). While you can always create these models in a couple lines of code importing from some library, we will focus on the underlying math and theory behind neural networks. This is crucial to have an understanding when you actually go and implement these models.

2 The Perceptron

2.1 Basics

The most basic unit of the most complicated neural networks is the perceptron. A perceptron takes in multiple inputs, multiplies each input by some number known as a *weight* and adds the inputs together. Another number, known as a *bias*, is added to this sum, and an activation, in this case the step function, is applied.

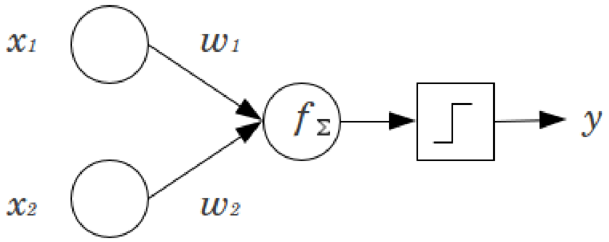


Figure 1: A basic perceptron with two inputs.

Figure 1 shows how a diagram for a perceptron. The mathematical representation of this is as follows:

$$f(x) = w_1x_1 + w_2x_2 + b$$

Given this value, however, we need to find a way to convert this into an output of 0 or 1. To do this, we can apply the step function. The step function is simply defined as outputting 0 when the input is less than or equal to 0 and outputting 1 when the input is greater than 0. Mathematically, this looks like as follows:

$$y = \begin{cases} 0 & \text{if } f(x) \le 0 \\ 1 & \text{if } f(x) > 0 \end{cases}$$

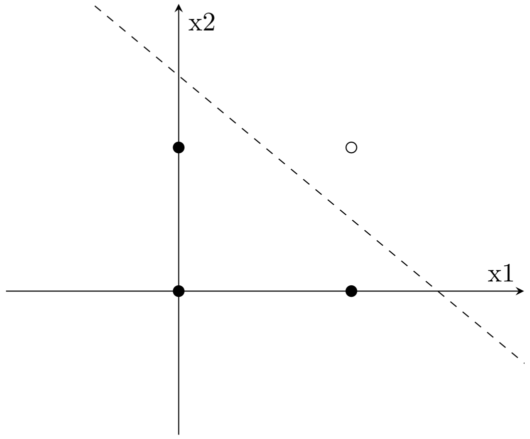
*Modified off Nikhil Sardana's lecture of the same name. Also edited by Saahith Janapati.

2.2 Visualization

The purpose of a perceptron is to classify data. A single perceptron can only be fully accurate when faced with a binary linear classification problem. Consider the following function.

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

Note how this function is the "AND" function in computer science, which means the output is 1 only when all the inputs are 1 as well, and 0 otherwise. Let's graph this data using an open-circle for $y = 1$ and an closed-circle for $y = 0$.



The line $x_2 = -x_1 + 1.5$ splits this data the best. Let's rearrange this to get $x_1 + x_2 - 1.5 = 0$. Going back to the perceptron formula

$$f(x) = w_1x_1 + w_2x_2 + b$$

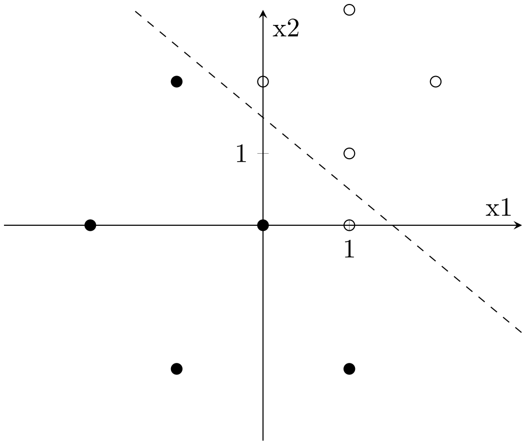
we can see that for the optimal perceptron, w_1 and w_2 are the coefficients of x and y, and $b = -1.5$. If $f(x) > 0$, then $x + y - 1.5 > 0$. We can see through this example that perceptrons are nothing more than linear functions. Above a line, perceptrons classify data points as 1, below the line, they classify data points as 0.

2.3 Learning

Now that we know how perceptrons classify data, the next question that follows is how we should choose the weights and bias of the perceptron. In other words, how do we determine which line to use that best classifies the data?

Neural networks are able to "learn" how to adjust their weights and biases based on the parameters it is using. To do this, they iteratively adjust these parameters until they find one that classifies the data sufficiently.

To help illustrate this, let's consider this data:



The perceptron that represents the dashed line $x_2+x_1-1.5=0$ has two inputs, x_1, x_2, with corresponding weights $w_1 = 1, w_2 = 1$, and bias $b = -1.5$. Let y represent the output of this perceptron. In the data above, the point $(1, 0)$ is the only misclassified point. The perceptron outputs 0 because it is below the line, but it should output a 1.

For some data point (input) i with coordinates (i_1, i_2), the perceptron adjusts its weights and bias according to this formula:

$$w_1 = w_1 + \alpha(d-y)(i_1)$$

$$w_2 = w_2 + \alpha(d-y)(i_2)$$

$$b = b + \alpha(d-y)$$

Where d is the desired output, and α is the learning rate, a constant usually between 0 and 1. Notice that the equation degenerates to $w = w$ and $b = b$ when the desired output equals the perceptron output. In other words, the perceptron only learns from misclassified points.

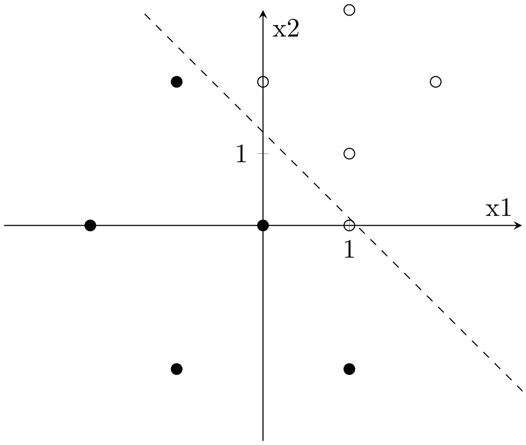
In the case of the above data, the perceptron only learns from the point $(1, 0)$. Let's set $\alpha = 0.2$ and compute the learning steps:

$$w_1 = 1 + 0.2(1-0)(1) = 1.2$$

$$w_2 = 1 + 0.2(1-0)(0) = 1$$

$$b = -1.5 + 0.2(1-0) = -1.3$$

After 1 iteration, the perceptron now represents the function $x_2 + 1.2x_1 - 1.3 = 0$, which is shown below:

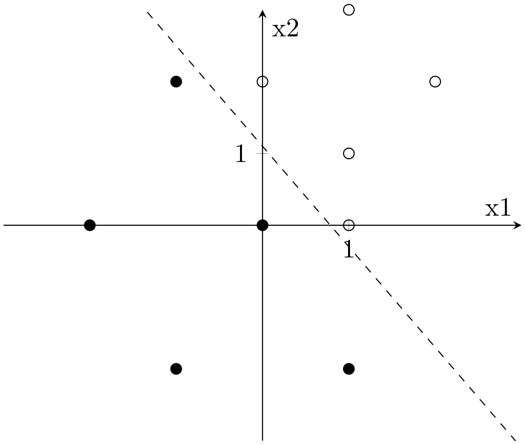


The next iteration follows:

$$w_1 = 1.2 + 0.2(1-0)(1) = 1.4$$

$$w_2 = 1 + 0.2(1-0)(0) = 1$$

$$b = -1.3 + 0.2(1-0) = -1.1$$



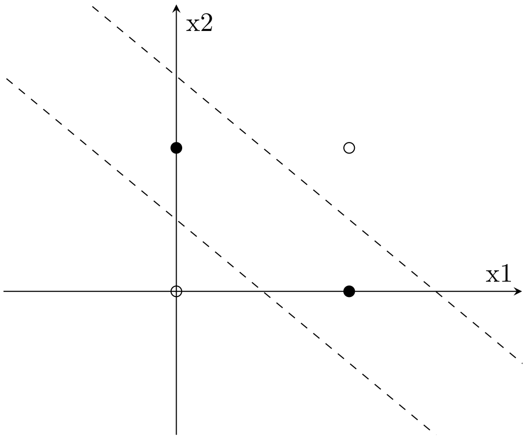
All the points are now correctly classified. The perceptron has learned! Notice how it has not learned the best possible line, only the first one that zeroes the difference between expected and actual output.

2.4 Non-Linearly Separable Data

Consider the function XOR, which outputs a 1 when the inputs are the same value and a 0 when the inputs are different values:

x1	x2	out
0	0	1
0	1	0
1	0	0
1	1	1

Let's graph this data.

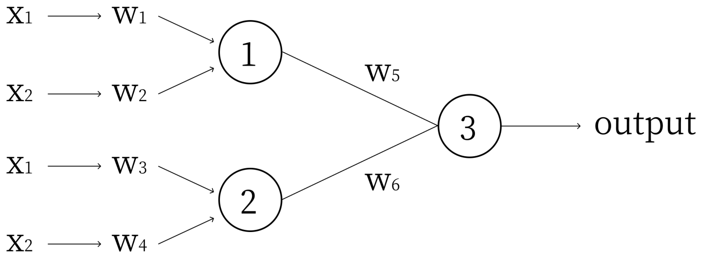


We need two lines to separate this data! A perceptron will never reach the optimal solution. However, multiple perceptrons can learn multiple lines, which can be used to classify non-linearly separable data.

3 Multi-Layer Perceptron

A neural network (NN) or Multi-Layer Perceptron (MLP) is an aggregate of these perceptrons glued together, and can be used to approximate multi-dimensional non-linearly separable data. Let us again consider the XOR function depicted above. How do we arrange perceptrons to represent the two functions?

Clearly, we need two perceptrons, one for each function. The output of these two perceptrons can be used as inputs to a third perceptron, which will give us our output. Refer to the diagram below.



Let Perceptron 1 model $x_2 + x_1 - 1.5 = 0$ (the upper line), and Perceptron 2 model $x_2 + x_1 - 0.5 = 0$ (the lower line). Because the weights are the coefficients of these functions, $w_1 = 1, w_2 = 1, w_3 = 1, w_4 = 1$ and the biases $b_1 = -1.5$ and $b_2 = -0.5$. These perceptrons refer to their respective numbers in the above diagram.

The output of Perceptron 1 will be a 1 for points above the upper line, and a 0 for the points below the upper line. The output of Perceptron 2 will be a 1 for points above the lower line, and a 0 for points below the lower line. Thus, above both lines, we get 2. In between the lines, we get 1. Below the lines, we get 0. However, in order to create a threshold to separate the points between the lines from the points outside, we would like the outputs for points between the lines to be additive.

In other words, we would like the inputs of Perceptron 3 to cancel outside the lines, and have a maximum for points inside the lines. Thus, we let $w_6 = -1$ and $w_5 = 1$ and $b_3 = 0.5$.

Now, for points between the lines we get an output of -0.5, which is negative and our network will classify as a 0, and for points outside of the two lines we get an output of 0.5, which is positive and our network will classify as a 1.

In practice, the perceptron is rarely used, since it is hard to detect non-linear information from the data. Next time, we'll be looking at more complex Neural Networks and also more complex activation functions, such as the sigmoid function, which allow for the detection of non-linearities in the data. It will involve much more math, specifically calculus and gradients, so we highly recommend checking out our Calculus for Machine Learning lecture if you are unfamiliar with these concepts.

4 Practice Problems

1. Write out the weights, biases, and structure of the Perceptron that classifies the "OR" function. The data are below:

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

2. Write out the weights, biases, and structure of the Multi-Layer perceptron that classifies the "XNOR" function.

x1	x2	y
0	0	1
0	1	0
1	0	0
1	1	1

3. Work through the example in Section 2.3 using an alpha of 0.05 and an alpha of 2. (Think about what the issues are of choosing an alpha that is too small and too large).

4. Write an implementation of the XOR Multi-Layer Perceptron (see Section 3) in Python.