

## 1.1. Objectives

In this laboratory session you will learn how to create a Perceptron model of the neuron, generate some test vectors, and evaluate the performance of the Perceptron as a classification tool. Your objectives are:

1. Learn about the use of functions and libraries in Python
2. Create a function that is based on the Perceptron model of the neuron
3. Evaluate this function as a binary (Boolean) classification tool
4. Learn how to perform basic plotting in Python to graphically visualise data

At the end of this laboratory session you should have the basic framework for a Perceptron model, you are free to modify this in any way you want for your future coursework.

## 1.2. Introduction

We will be the same tools as the previous laboratory (i.e Visual Studio Code & the Anaconda Python distribution). Follow the instructions from Laboratory 1 to create a new project, be sure to select Anaconda as the Python environment to run.

### The Perceptron

The Perceptron was first described in 1957 by Frank Rosenblatt at Cornell. It is a simple type of linear classifier - the output depends on a linear combination of the inputs. When first introduced, many Perceptrons were connected to form the first ever artificial neural network. In this laboratory we will start by considering a single Perceptron, whose output can be described by Equation 3.1:

$$f(x) = \begin{cases} 1, & \text{if } w \cdot x + b > 0 \\ 0, & \text{otherwise} \end{cases} \quad (3.1)$$

Where  $w$  is a vector of real-valued weights,  $w \cdot x$  is the dot product and  $b$  is a bias term that can be used to scale the effective threshold. We can begin a simple analysis by constraining the inputs to be Boolean (0 or 1), and we can set the weights and the bias by hand. A graphical description of the model is given in Figure 3.1 for  $N$  inputs, and a mathematical expansion of Equation 3.1 is given in Equation 3.2, where  $\theta$  represents the threshold function.

$$f(x) = \theta \left( \sum_{i=1}^N (w_i x_i + b) \right) \quad (3.2)$$

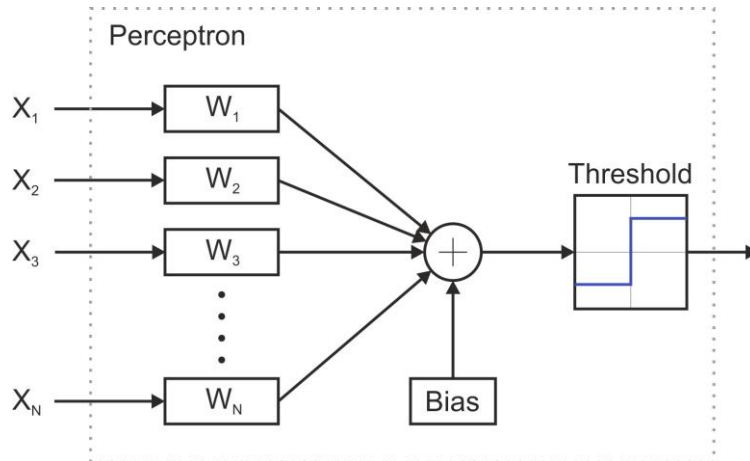


Figure 3.1: A single Perceptron with  $N$  inputs  $X_N$ , the inputs are multiplied by weights  $W_N$  and summed with the bias before a simple hardthreshold.

## Python Libraries

In order to extend the capabilities of Python we are going to use some of the libraries that are provided by the Anaconda distribution (these libraries are actually written by the open source community, Anaconda is just a package that installs the most popular libraries. There are two libraries that we will make use of in this laboratory, NumPy and matplotlib.

### Important Message

On the laboratory computers I have noticed that the first time you run a program that imports a library there can be quite a long delay before the program starts (around 6 seconds). This problem seems to be less noticeable on successive runs.

NumPy is a library that adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these matrices. In many ways, NumPy adds a matrix feature set that resembles the capabilities of MATLAB, for example the dot product in Equation 3.1 can be performed in a single line of code, rather than writing a long for loop. This makes working with arrays and matrices much easier.

matplotlib is a plotting library that provides a good interface for drawing graphs and charts. As the name suggests it has been influenced by the plotting engine in MATLAB, as so the syntax is broadly similar and should be familiar to most of you.


You do not need to understand the intricacies of these libraries, fully worked examples will be given, however for your coursework you may choose to explore some of the capabilities in more detail.

## 1.3. Getting Started

If you have not already done so, create a new Python project using the instructions given in the lab from week 1.

### Creating the Perceptron

Code listing 3.1 shows a simple program for simulating the Perceptron. Read through this code and type it into your own program. Library imports occur at the start of the program, followed by function definitions and then the main code. Note that in Python, functions work just like they do in MATLAB, here our Perceptron function takes three inputs and returns a single output. In the main code section, near the bottom, we create two lists that represent our inputs  $X_N$  and our weights  $W_N$ . Within the Perceptron function, we convert those lists into NumPy arrays. The reason for this is shown in line 14, using the NumPy array type we can compute the dot product in a single line, rather than using a for loop.

```
 Perceptron
1  # Import the NumPy library for matrix math
2  import numpy
3
4
5  # A single perceptron function
6  def perceptron(inputs_list, weights_list, bias):
7      # Convert the inputs list into a numpy array
8      inputs = numpy.array(inputs_list)
9
10     # Convert the weights list into a numpy array
11     weights = numpy.array(weights_list)
12
13     # Calculate the dot product
14     summed = numpy.dot(inputs, weights)
15
16     # Add in the bias
17     summed = summed + bias
18
19     # Calculate output
20     # N.B this is a ternary operator, neat huh?
21     output = 1 if summed > 0 else 0
22
23     return output
24
25  # Our main code starts here
26
27  # Test the perceptron
28  inputs = [1.0, 0.0]
29  weights = [1.0, 1.0]
30  bias = -1
31
32  print("Inputs: ", inputs)
33  print("Weights: ", weights)
34  print("Bias: ", bias)
35  print("Result: ", perceptron(inputs, weights, bias))
```

Code 3.1: perceptron.py

## 1.4. Exercises

Spend some time familiarising yourself with the basic Perceptron model. Once you have the model running, you should try the following exercises:

1. For a two-input perceptron with  $W = [1.0, 1.0]$  and  $b = -1$ , simulate the output for all possible Boolean input combinations where  $X \in \{0,1\}$ .
2. Interpreting this information as a truth table, what logical function is being performed?
3. What are the weight vectors and the bias for the logical functions: AND, OR, NAND, NOR and XOR?
4. Why was 3. a trick question?

## Further Exercises

Once you are happy with the basic Perceptron model, you can extend it using the matplotlib library to give you a graphical interpretation of the data. Code listing 3.2 shows a basic program that plots a scatter diagram of the input vector state space (every possible value of every input) for our two input Boolean Perceptron. Here are a few features you could add:

1. Modify your Perceptron code so that it automatically tests all possible input vectors (i.e  $[0, 0]$ ,  $[0, 1]$ ,...) *Hint: store the input vectors in a list, and use a for loop to test each input vector.*
2. With the aid of the example plotting code in listing 3.2, modify your code so that it automatically generates a plot of the input vector state space - use colour coding to signify if the Perceptron fired (output = 1) for that input.
3. Calculate the linear separator for the Perceptron and add it to the plot, an example of the final result is shown in Figure 3.2.

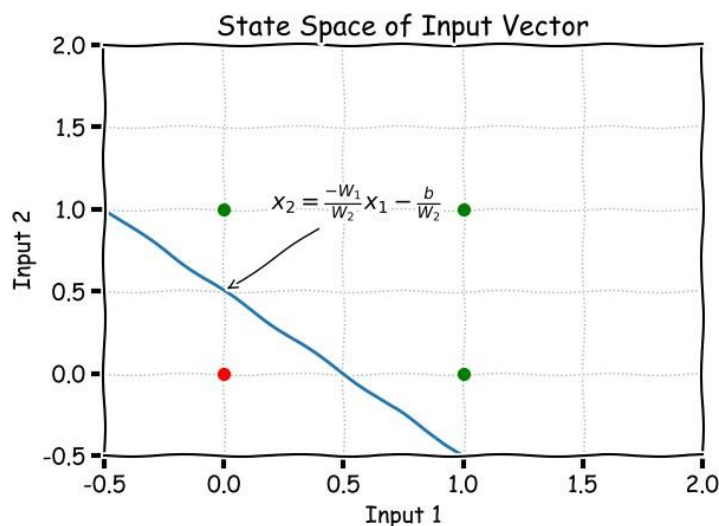


Figure 3.2: Example state space diagram with overlaid linear separator.

## Plotting

```
1 # Import the matplotlib pyplot library
2 # It has a very long name, so import it as the name plt
3 import matplotlib.pyplot as plt
4
5 # Make a new plot (XKCD style)
6 fig = plt.xkcd()
7
8 # Add points as scatters - scatter(x, y, size, color)
9 # zorder determines the drawing order, set to 3 to make the
10 # grid lines appear behind the scatter points
11 plt.scatter(0, 0, s=50, color="red", zorder=3)
12 plt.scatter(0, 1, s=50, color="red", zorder=3)
13 plt.scatter(1, 0, s=50, color="red", zorder=3)
14 plt.scatter(1, 1, s=50, color="green", zorder=3)
15
16 # Set the axis limits
17 plt.xlim(-2, 2)
18 plt.ylim(-2, 2)
19
20 # Label the plot
21 plt.xlabel("Input 1")
22 plt.ylabel("Input 2")
23 plt.title("State Space of Input Vector")
24
25 # Turn on grid lines
26 plt.grid(True, linewidth=1, linestyle=':')
27
28 # Autosize (stops the labels getting cut off)
29 plt.tight_layout()
30
31 # Show the plot
32 plt.show()
```

Code 3.2: plotting.py