Insert Title Here

by

PARBHU Varun

Project Supervisor: Professor (Dr) BHURUTH MUDDUN OSK

Project submitted to the Department of Mathematics,
Faculty of Science, University of Mauritius,
as a partial fulfillment of the requirement
for the degree of
M.Sc. Mathematical and Scientific Computing (Part time)

December 2022

Contents

List of Figures

List of Tables

Acknowledgement

Placeholder

Abstract

Placeholder

Terms and Definitions

α	Learning Rate
$\mathbf{I}$	Identity Matrix

Placeholder

Placeholder

Chapter 1

Introduction

- Discussion around Cryptocurrency and frequency of changes
- Discussion around Twitter and the API (amount of data)
- Discussion around the advancement Compute processing speed and Deep Learning algorithms

Chapter 2

Deep Learning Networks

2.1 Introduction

Deep learning is a multidisciplinary field ranging from linear algebra, calculus and probability theory. With major advancement made in compute processing power; deep learning is commonly used in the industry now.

McCullough and Walter Pitts (MCP) Neuron

In 1943, McCulloch and Walter Pitts demonstrated that simple binary threshold units wired up as logical gates could be used to build a digital computer (add-ref). The McCulloch-Pitts (MCP) neuron is a simple mathematical model of a biological neuron. It was the earliest mathematical model of a neural network and had only three types of weights; excitatory (1), inhibitory (-1) and inactive (0). The model had an activation function which had a value of 1 if the weighted sum was greater or equal to a given threshold, else 0. Using the MCP neuron, one of the first digital computers that contained stored programs was built. However, the MCP neuron was very restrictive.

Rosenblatt's Perceptron Algorithm

In 1958, Frank Rosenblatt proposed the perceptron and an algorithm to adjust weights of the MCP neuron, extending the work done by McCulloch and Pitts. The MCP neuron had some limitations that the Perceptron managed to solve, for exam-

ple the input was not restricted to boolean values but expanded to real numbers. Rosenblatt proved that if the data used to train the perceptron are linearly separable classes, then the perceptron algorithm convergences and separates the two classes by a hyperplane.

Perceptron

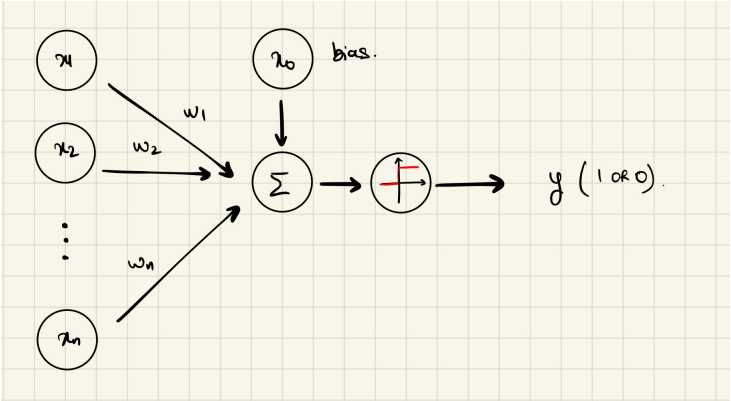

Figure 2.1: Rosenblatt perceptron

Let $\mathbf{x}$ be a vector of inputs where each $x_i \in \mathbb{Z}$ and $\mathbf{w}$ be a vector of weights corresponding to the input signals where each $w_i \in \mathbb{Z}$.

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

Then, the mathematical definition of the perceptron is given by

$$\phi(z) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $\phi(z)$ is known as the hard delimiter.

Consider the two sets of point **A** and **B**

$$\mathbf{A} = \left\{ \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\} \quad \mathbf{B} = \left\{ \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right\} \tag{2.2}$$

and fitting the perceptron function (2.1) to learn how to classify points in each respective set. Let set **A** and **B** be class 1 and 0 respectively.

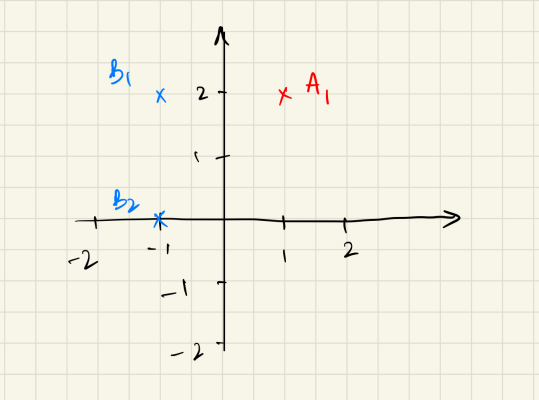

Figure 2.2: Caption

From 2.2, we can separate the two classes by a single line. Consider the random separating line passing through the origin (this will set the value of x_0 in (2.1) to 0).

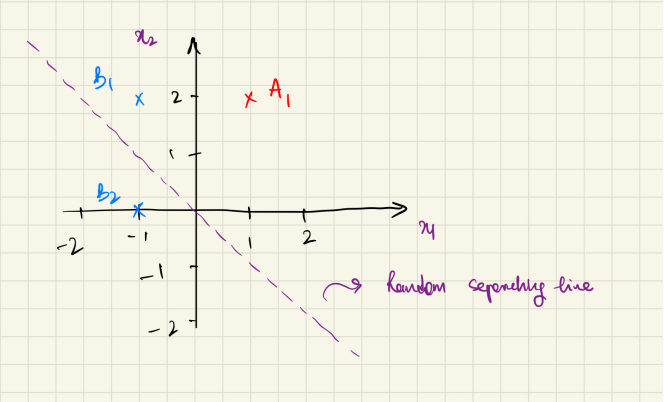

Figure 2.3: Caption

The equation of the random line is given by

$$x_1 = -x_2 \tag{2.3}$$

Comparing to the equation (2.1); we can deduce that

$$w_1 = 1 \quad w_2 = 1 \tag{2.4}$$

Visually, we can already see that the separating line does not split the points properly. The computation of the binary classification using the random line is given by

$$\phi(A_1) = \phi\left(\begin{bmatrix}1\\1\end{bmatrix} . \begin{bmatrix}1\\2\end{bmatrix}\right) = \phi(3) = 1 \tag{2.5}$$

$$\phi(B_1) = f\left(\begin{bmatrix}1\\1\end{bmatrix} . \begin{bmatrix}-1\\2\end{bmatrix}\right) = \phi(1) = 1 \tag{2.6}$$

$$\phi(B_2) = f\left(\begin{bmatrix}1\\1\end{bmatrix} . \begin{bmatrix}-1\\0\end{bmatrix}\right) = \phi(-1) = 0 \tag{2.7}$$

The computation confirms the visual representation and groups $\mathbf{A}_1$ and $\mathbf{B}_1$ together and $\mathbf{B}_2$ separated which is incorrect. We can either alter the separating line (by moving the weight vector) with respect to $\mathbf{A}_1$ and/or $\mathbf{B}_1$.

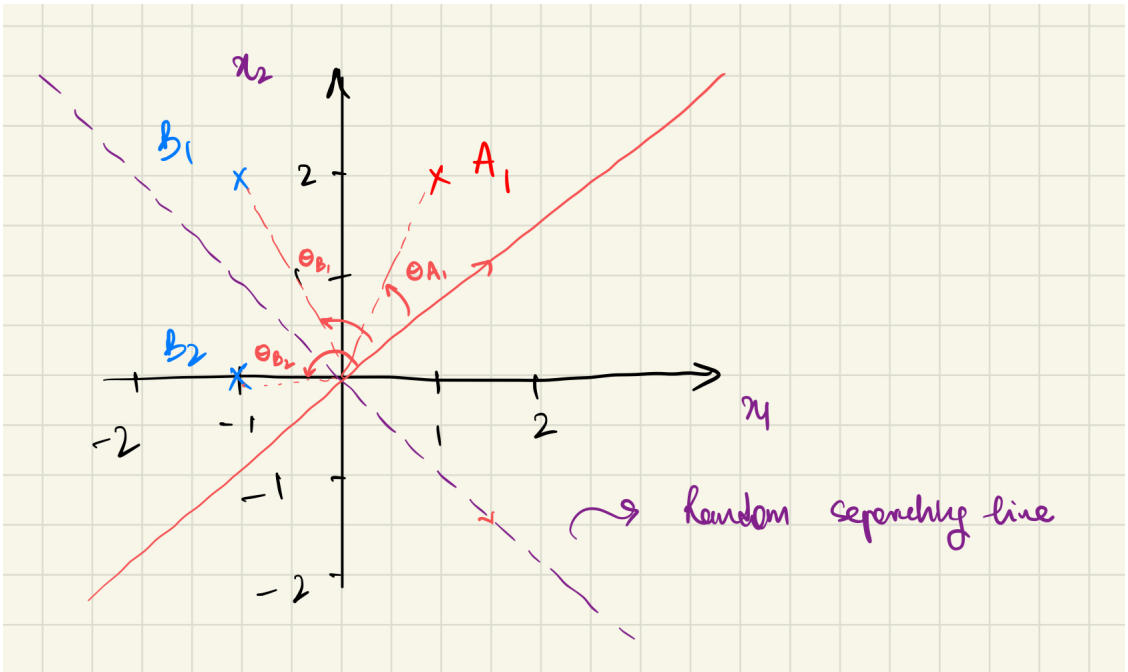

Figure 2.4: !!!TO BE CHANGED!!!

Since $\mathbf{B}_1$ is incorrectly classified, using the property of subtraction of vectors; we move the weight vector ($\mathbf{w}$) away from $\mathbf{B}_1$ and check the classification again.

$$\mathbf{w}^{(1)} = \mathbf{w} - \mathbf{B}_1 \tag{2.8}$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix} \tag{2.9}$$

$$= \begin{bmatrix} 2 \\ -1 \end{bmatrix} \tag{2.10}$$

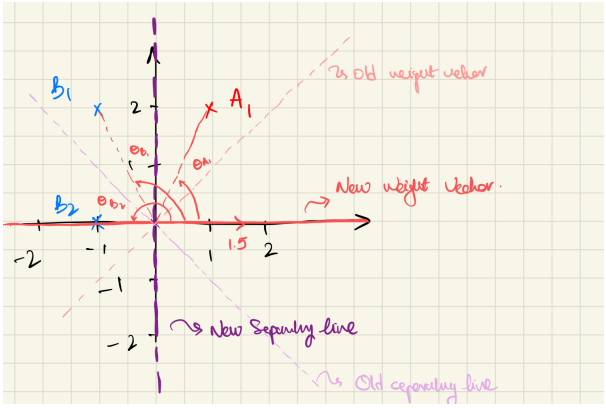

Figure 2.5: !!!!TO BE CHANGED!!!!

Based on the new separating line, we can observe visually that the points are correctly classified. The computation of the binary classification using the new separating line $\mathbf{w}^{(1)}$ is given by

$$\phi(A_1) = \phi\left(\begin{bmatrix} 2 \\ -1 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) = \phi(0) = 1 \tag{2.11}$$

$$\phi(B_1) = \phi\left(\begin{bmatrix} 2 \\ -1 \end{bmatrix} . \begin{bmatrix} -1 \\ 2 \end{bmatrix}\right) = \phi(-4) = 0 \tag{2.12}$$

$$\phi(B_2) = \phi\left(\begin{bmatrix} 2 \\ -1 \end{bmatrix} . \begin{bmatrix} -1 \\ 0 \end{bmatrix}\right) = \phi(-2) = 0 \tag{2.13}$$

Thus, the perceptron learned how to classify the two sets of points. The function $f(\mathbf{x})$ could be re-used to classify new points added to the sets.

Perceptron Algorithm

The perceptron algorithm adjusts the weights and as a result, the hard delimiter as well in order to linearly separate a set of binary labelled input.

Algorithm 1 Perceptron Algorithm

Require:

Let $t = 0$ and $\mathbf{w} = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}^T$
Consider the training set D $s.t$ $D = C_1 \cup C_2$
$C_1 \leftarrow$ input with label 1
$C_2 \leftarrow$ input with label 0
Start
while !convergence **do**
    Select a random input $\mathbf{x}$
    if $\mathbf{x} \in A$ and $\mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$
    end if
    if $\mathbf{x} \in B$ and $\mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$
    end if
end while

If the input data are binary and linearly separable; then algorithm (1) converges. The proof of convergence of the algorithm is known as the **perceptron convergence theorem**.

Perceptron Convergence Theorem

Theorem

Consider algorithm (1) and let D be a set of training vectors which are linearly separable. Let $\mathbf{w}^*$ be the weight vectors which defines the separating line with $||\mathbf{w}^*|| = 1$. Then the number of mistakes m made by the perceptron algorithm satisfies

$$m \leq \frac{1}{\gamma^2} \quad \text{where} \quad \gamma = \min_{\mathbf{x} \in D} \frac{|\mathbf{w}^{*T}\mathbf{x}|}{||\mathbf{x}||_2} \tag{2.14}$$

Note

1. Since, $||\mathbf{w}^*|| = 1 \implies \cos(\theta) = \dfrac{\mathbf{w}^{*T}\mathbf{x}}{||\mathbf{x}||_2}$

2. If $||\mathbf{x}||_2 = 1$, that is, we scale all the training examples to have unit norm (which has no effect their orientation), then $\gamma = \min_{\mathbf{x}\in D}|\mathbf{w}^{*T}\mathbf{x}|$ is the minimum distance from any example $\mathbf{x} \in D$ to the separating line.

Proof

We first prove the inequality given by

$$(\mathbf{w}^{(t+1)})^T\mathbf{w}^* \geq (\mathbf{w}^{(t)})^T\mathbf{w}^* + \gamma \tag{2.15}$$

State 1

In state 1, let us consider $\mathbf{x}$ being positive ($\mathbf{x} \in C_1$) and incorrectly classified then

$$(\mathbf{w}^{(t+1)})^T\mathbf{w}^* = (\mathbf{w}^{(t)} + \mathbf{x})^T\mathbf{w}^* = (\mathbf{w}^{(t)})^T\mathbf{w}^* + \mathbf{x}^T\mathbf{w}^*$$

Assuming that all $||\mathbf{x}||_2 = 1$ then $\mathbf{x}^T\mathbf{w}^* \geq \gamma$ since γ is the minimum.

$$(\mathbf{w}^{(t+1)})^T\mathbf{w}^* \geq (\mathbf{w}^{(t)})^T\mathbf{w}^* + \gamma \tag{2.16}$$

Thus, we have proved (2.15) under **State 1**.

State 2

In state 2, let us consider $\mathbf{x}$ being negative ($\mathbf{x} \in C_2$) and incorrectly classified then

$$(\mathbf{w}^{(t+1)})^T\mathbf{w}^* = (\mathbf{w}^{(t)} - \mathbf{x})^T\mathbf{w}^* = (\mathbf{w}^{(t)})^T\mathbf{w}^* - \mathbf{x}^T\mathbf{w}^*$$

Since $\mathbf{x} \in C_2 \implies |\mathbf{w}^{*T}\mathbf{x}| = -\mathbf{x}^T\mathbf{w}^* \geq 0$ and $|\mathbf{x}^T\mathbf{w}^*| \geq \gamma$

$$\begin{aligned}(\mathbf{w}^{(t+1)})^T\mathbf{w}^* &= (\mathbf{w}^{(t)})^T\mathbf{w}^* + |\mathbf{w}^{*T}\mathbf{x}| \\ (\mathbf{w}^{(t+1)})^T\mathbf{w}^* &\geq (\mathbf{w}^{(t)})^T\mathbf{w}^* + \gamma \end{aligned} \tag{2.17}$$

Thus, we have proved (2.15) under **State 2**.

From (2.16) and (2.17), we have shown that $\forall \mathbf{x} \in D$ that inequality (2.15) holds.

Assume that the inequality (2.15) holds for an arbitrary integer value m and $m-1$

$$(\mathbf{w}^{(m)})^T\mathbf{w}^* \geq (\mathbf{w}^{(m-1)})^T\mathbf{w}^* + \gamma \tag{2.18}$$

$$(\mathbf{w}^{(m-1)})^T\mathbf{w}^* \geq (\mathbf{w}^{(m-2)})^T\mathbf{w}^* + \gamma \tag{2.19}$$

Merging inequality (2.18) and (2.19) gives us

$$(\mathbf{w}^{(m)})^T\mathbf{w}^* \geq (\mathbf{w}^{(m-2)})^T\mathbf{w}^* + 2\gamma \tag{2.20}$$

Hence, by induction, after M mistakes, inequality (2.20) becomes

$$(\mathbf{w}^{(m)})^T\mathbf{w}^* \geq (\mathbf{w}^{(0)})^T\mathbf{w}^* + m\gamma \tag{2.21}$$

Since $\mathbf{w}^{(0)} = \mathbf{0}$ (the zero vector), we have

$$(\mathbf{w}^{(m)})^T\mathbf{w}^* \geq m\gamma \tag{2.22}$$

Next, we show that

$$||\mathbf{w}^{(t+1)}||_2^2 \leq ||\mathbf{w}^{(t)}||_2^2 + 1 \tag{2.23}$$

State 1

Consider the same state 1 as before, thus we have

$$(\mathbf{w}^{(t)})^T\mathbf{x} \leq 0 \quad \text{and} \quad \mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{x} \tag{2.24}$$

then

$$\begin{aligned} ||\mathbf{w}^{(t+1)}||_2^2 &= (\mathbf{w}^{(t)} + \mathbf{x})^T(\mathbf{w}^{(t)} + \mathbf{x}) \\ &= (\mathbf{w}^{(t)} + \mathbf{x})^T\mathbf{w}^{(t)} + (\mathbf{w}^{(t)} + \mathbf{x})^T\mathbf{x} \\ &= (\mathbf{w}^{(t)})^T\mathbf{w}^{(t)} + \mathbf{x}^T\mathbf{w}^{(t)} + (\mathbf{w}^{(t)})^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \\ &= (\mathbf{w}^{(t)})^T\mathbf{w}^{(t)} + 2(\mathbf{w}^{(t)})^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \end{aligned} \tag{2.25}$$

Since (2.24) and $\mathbf{x}^T\mathbf{x} = 1$, equation (2.25) becomes

$$||\mathbf{w}^{(t+1)}||_2^2 \leq ||\mathbf{w}^{(t)}||_2^2 + 1 \tag{2.26}$$

State 2

Consider the same state 2 as before, thus we have

$$(\mathbf{w}^{(t)})^T\mathbf{x} > 0 \quad \text{and} \quad \mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \mathbf{x} \tag{2.27}$$

then

$$\begin{aligned} ||\mathbf{w}^{(t+1)}||_2^2 &= (\mathbf{w}^{(t)} - \mathbf{x})^T(\mathbf{w}^{(t)} - \mathbf{x}) \\ &= (\mathbf{w}^{(t)} - \mathbf{x})^T\mathbf{w}^{(t)} - (\mathbf{w}^{(t)} - \mathbf{x})^T\mathbf{x} \\ &= (\mathbf{w}^{(t)})^T\mathbf{w}^{(t)} - \mathbf{x}^T\mathbf{w}^{(t)} - (\mathbf{w}^{(t)})^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \\ &= (\mathbf{w}^{(t)})^T\mathbf{w}^{(t)} - 2(\mathbf{w}^{(t)})^T\mathbf{x} + \mathbf{x}^T\mathbf{x} \end{aligned} \tag{2.28}$$

Since (2.27) and $\mathbf{x}^T\mathbf{x} = 1$, equation (2.28) becomes

$$||\mathbf{w}^{(t+1)}||_2^2 \leq ||\mathbf{w}^{(t)}||_2^2 + 1 \tag{2.29}$$

Thus, we have proved (2.23) under **State 2**.

By the same induction defined in (2.21), we have

$$||\mathbf{w}^{(m)}||_2^2 \leq m \tag{2.30}$$

Finally, we merge result (2.22) and (2.30) using the Cauchy-Schwarz (CS) inequality

Recall that the CS inequality is given by

$$|\mathbf{w}^{*T}\mathbf{x}| \leq ||\mathbf{w}^*||_2||\mathbf{x}||_2 \tag{2.31}$$

Hence, using (2.31) we get

$$|(\mathbf{w}^m)^T\mathbf{w}^*| \leq ||(\mathbf{w}^m)^T||_2||\mathbf{w}^*||_2 \tag{2.32}$$

From the result (2.22), (2.32) becomes

$$m\gamma \leq |(\mathbf{w}^m)^T\mathbf{w}^*| \leq ||(\mathbf{w}^m)^T||_2||\mathbf{w}^*||_2 \tag{2.33}$$

Using the result (2.30) and the fact that $||\mathbf{w}^*||_2 = 1$, we get

$$m\gamma \leq ||(\mathbf{w}^m)^T||_2$$

$$m^2\gamma^2 \leq ||(\mathbf{w}^m)^T||_2^2 \tag{2.34}$$

$$m^2\gamma^2 \leq m \tag{2.35}$$

$$m \leq \frac{1}{\gamma^2} \tag{2.36}$$

This proves the Perceptron Convergence Theorem that the number of mistakes is at most $\frac{1}{\gamma^2}$, where γ is the margin.

Solving Gate problem using the Perceptron

The perceptron alone is not always a good model to solve binary classification problem. It has certain limitation by nature of its definition. Consider the logic functions AND, OR and XOR across these 4 points:

$$\mathbf{p}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.37}$$

AND Function

The AND function returns a value of 1 if both of the inputs is 1. Let f be the AND function such that

$$f(\mathbf{p}_1) = 0 \quad f(\mathbf{p}_2) = 0 \quad f(\mathbf{p}_3) = 1 \quad f(\mathbf{p}_4) = 0$$

Consider the AND function with the separating line below

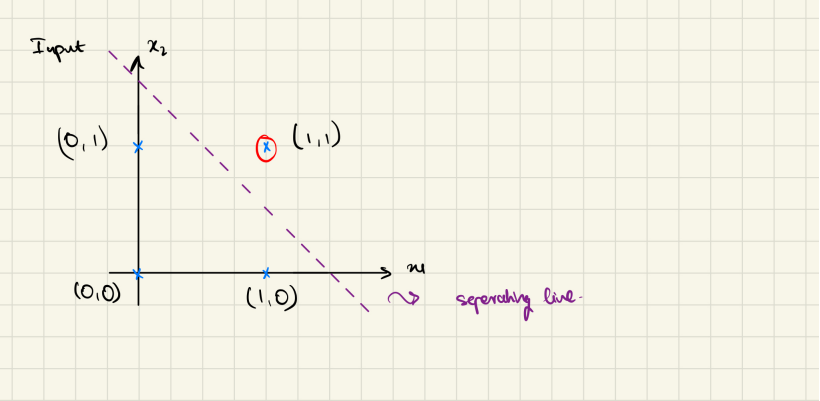

Figure 2.6: AND Function with separating line

The equation of the separating line in figure 2.6 is given by

$$x_1 + x_2 - 1.5 = 0 \tag{2.38}$$

We can transform the separating line in the form of $\mathbf{w}^T\mathbf{x} = 0$ to fit the perceptron. Let

$$\mathbf{w} = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \tag{2.39}$$

Applying function (2.1) on each point to check whether they are correctly classified.

$$\phi(p_1) = \phi\left(\begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right) = \phi(-1.5) = 0$$

$$\phi(p_2) = \phi\left(\begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right) = \phi(-0.5) = 0$$

$$\phi(p_3) = \phi\left(\begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right) = \phi(0.5) = 1$$

$$\phi(p_4) = \phi\left(\begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right) = \phi(-0.5) = 0$$

Thus, the perceptron is able to solve the AND function.

OR Function

The OR function returns a value of 1 if either of the inputs is 1. Let f be the OR function such that

$$f(\mathbf{p}_1) = 0 \quad f(\mathbf{p}_2) = 1 \quad f(\mathbf{p}_3) = 1 \quad f(\mathbf{p}_4) = 1$$

Consider the OR function with the separating line below

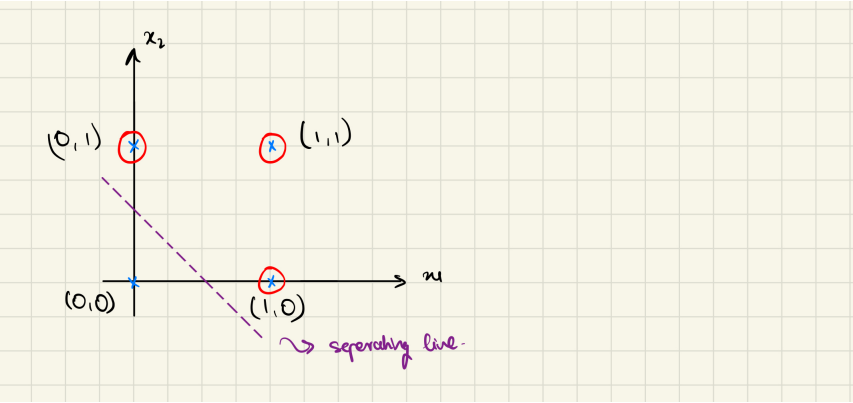

Figure 2.7: AND Function with separating line

The equation of the separating line in figure 2.7 is given by

$$x_1 + x_2 - 0.5 = 0 \tag{2.40}$$

We can transform the separating line in the form of $\mathbf{w}^T\mathbf{x} = 0$ to fit the perceptron. Let

$$\mathbf{w} = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \tag{2.41}$$

Applying function (2.1) on each point to check whether they are correctly classified.

$$\phi(p_1) = \phi\left(\begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right) = \phi(-0.5) = 0$$

$$\phi(p_2) = \phi\left(\begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right) = \phi(0.5) = 1$$

$$\phi(p_3) = \phi\left(\begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right) = \phi(1.5) = 1$$

$$\phi(p_4) = \phi\left(\begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right) = \phi(0.5) = 1$$

Thus, the perceptron is able to solve the OR function.

XOR Function

The XOR function returns a value of 1 if either of the inputs is 1 but not both. Let f be the XOR function such that

$$f(\mathbf{p}_1) = 0, \quad f(\mathbf{p}_2) = 1, \quad f(\mathbf{p}_3) = 0, \quad f(\mathbf{p}_4) = 1 \tag{2.42}$$

Consider a plot of the XOR function below

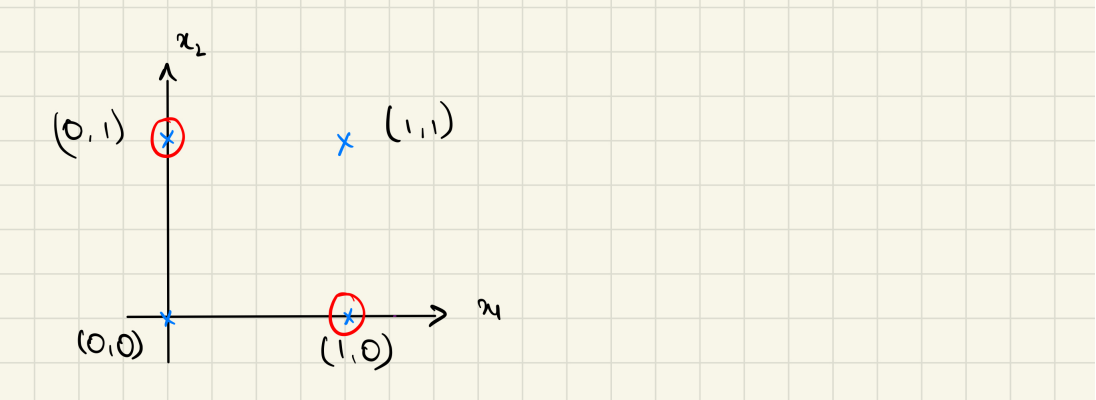

Figure 2.8: OR function

From figure 2.8; we can observe that no separating line alone could group the two different classes together. We attempt to find a combination of weight for the perceptron which reduces the error (margin) as much as possible.

Let f^* be the exact function and $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} + b$, where $\mathbf{w} = [w_1 \ w_2]^T$, be the approximate function. Consider the error function (the difference between the exact and approximate value)

$$\mathbb{E}(\mathbf{w}, b) = \frac{1}{4}\sum_{x \in D}(f^*(\mathbf{x}) - \mathbf{w}^T\mathbf{x} - b)^2 \tag{2.43}$$

we minimise the error function $\mathbb{E}$ by setting the partial derivatives to 0. The partial

derivative with respect to the bias is given by

$$\frac{\partial \mathbb{E}}{\partial b} = -\frac{1}{2}\sum_{x \in D}(f^*(\mathbf{x}) - \mathbf{w}^T\mathbf{x} - b)$$

$$\frac{\partial \mathbb{E}}{\partial b} = 0 \rightarrow \sum_{x \in D} f^*(\mathbf{x}) = \sum_{x \in D} \mathbf{w}^T\mathbf{x} - \sum_{x \in D} b \tag{2.44}$$

The partial derivative with respect to the weight vector is given by

$$\begin{aligned}\frac{\partial \mathbb{E}}{\partial \mathbf{x}} &= \frac{1}{2}\sum_{x \in D}(f^*(\mathbf{x}) - \mathbf{w}^T\mathbf{x} - b)\frac{\partial(-\mathbf{w}^T\mathbf{x})}{\partial \mathbf{w}} \\ &= -\frac{1}{2}\sum_{x \in D}(f^*(\mathbf{x}) - \mathbf{w}^T\mathbf{x} - b)\mathbf{x}\end{aligned}$$

$$\frac{\partial \mathbb{E}}{\partial \mathbf{w}} = 0 \rightarrow \sum_{x \in D} \mathbf{x} f^*(\mathbf{x}) = \sum_{x \in D}(\mathbf{w}^T\mathbf{x})\mathbf{x} + b\sum_{x \in D}\mathbf{x} \tag{2.45}$$

Substituting for values in equation (2.44)

$$\begin{aligned}\sum_{x \in D} f^*(\mathbf{x}) &= f^*(\mathbf{p}_1) + f^*(\mathbf{p}_2) + f^*(\mathbf{p}_3) + f^*(\mathbf{p}_4) \\ &= 0 + 1 + 1 + 0 \\ &= 2 \qquad (2.46)\end{aligned}$$

$$\begin{aligned}\sum_{x \in D} \mathbf{w}^T\mathbf{x} &= \mathbf{w}^T \begin{bmatrix}0\\0\end{bmatrix} + \mathbf{w}^T \begin{bmatrix}1\\0\end{bmatrix} + \mathbf{w}^T \begin{bmatrix}1\\1\end{bmatrix} + \mathbf{w}^T \begin{bmatrix}0\\1\end{bmatrix} \\ &= 2w_1 + 2w_2 \qquad (2.47)\end{aligned}$$

$$\begin{aligned}\sum_{x \in D} b &= b\sum_{x \in D} 1 \\ &= 4b \qquad (2.48)\end{aligned}$$

Thus, we get

$$2 = 2w_1 + 2w_2 + 4b$$

$$w_1 + w_2 + 2b = 1 \tag{2.49}$$

Substituting for values in equation (2.45)

$$\sum_{x\in D}\mathbf{x}f^*(\mathbf{x}) = \begin{bmatrix}0\\0\end{bmatrix}(0) + \begin{bmatrix}1\\0\end{bmatrix}(1) + \begin{bmatrix}1\\1\end{bmatrix}(0) + \begin{bmatrix}0\\1\end{bmatrix}(1)$$
$$= \begin{bmatrix}1\\1\end{bmatrix} \tag{2.50}$$

$$\sum_{x\in D}(\mathbf{w}^T\mathbf{x})\mathbf{x} = \begin{bmatrix}w_1 & w_2\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}w_1 & w_2\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}$$
$$+ \begin{bmatrix}w_1 & w_2\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} + \begin{bmatrix}w_1 & w_2\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}$$
$$= \begin{bmatrix}2w_1 + w_2\\w_1 + 2w_2\end{bmatrix} \tag{2.51}$$

$$b\sum_{x\in D}\mathbf{x} = b\begin{bmatrix}0\\0\end{bmatrix} + b\begin{bmatrix}1\\0\end{bmatrix} + b\begin{bmatrix}1\\1\end{bmatrix} + b\begin{bmatrix}0\\1\end{bmatrix}$$
$$= b\begin{bmatrix}2\\2\end{bmatrix} \tag{2.52}$$

Thus, we get

$$\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}2w_1 + w_2\\w_1 + 2w_2\end{bmatrix} + b\begin{bmatrix}2\\2\end{bmatrix} \tag{2.53}$$

From equations (2.49) and (2.53), we get the set of equations that forms the approximate function with the minimum error

$$w_1 + w_2 + 2b = 1$$
$$2w_1 + w_2 + 2b = 1$$
$$w_1 + 2w_2 + 2b = 1$$

Solving the system of linear equations we get

$$w_1 = 0 \quad w_2 = 0 \quad b = \frac{1}{2} \tag{2.54}$$

Hence, the approximate function $f(\mathbf{x}, \mathbf{w})$ is given by

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{2} \tag{2.55}$$

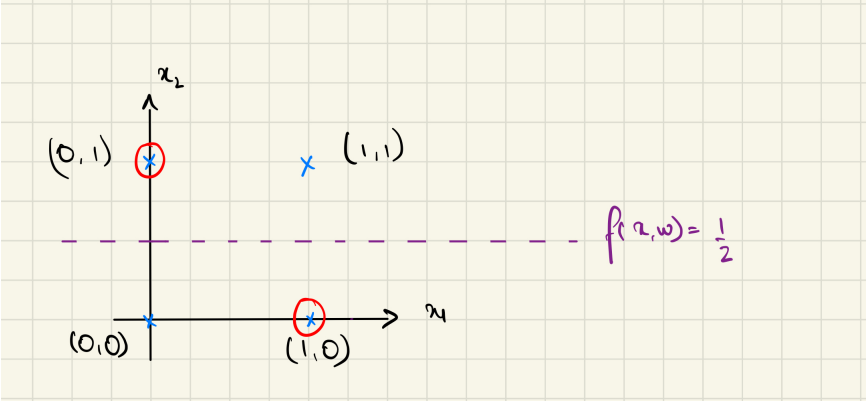

Figure 2.9: XOR approximate solution

Clearly from figure 2.9, the perceptron model cannot solve the XOR logic function. The XOR problem is actually not linearly separable. Thus, we introduce **feedforward networks** to overcome this problem.

2.1.1 Feedforward Networks

2.2 Differential calculus

Differential calculus is the study of the definition, properties, and applications the derivative of a function. The process of finding the derivative is called differentiation. It is the also referred to as the study of rate of change and slopes of curves. In optimization and deep learning, major concepts of differential calculus are used.

2.2.1 Derivatives

Derivatives are important concept for understanding deep learning. The derivative of a function at a particular point is the rate of change of the output of the function with respect to the input at that point. It is also an indication of the slope of the function at a particular point.

The mathematical definition of the derivative of a function $f = f(x)$ at a point a is given by

$$f^{'}(a) = \frac{df}{dx}(a) \tag{2.56}$$

$$= \lim_{\Delta \to 0} \frac{f(a+\Delta) - f(a-\Delta)}{2 \times \Delta} \tag{2.57}$$

Image Placeholder derivative of a function f at a in $\mathbb{R}^2$ and $\mathbb{R}^3$

Since the derivative of a function is an indication of the slope of a function; we can deduce that a minimum or maximum point exists when the derivative is equal to 0 (may also be indication of an inflection point).

Derivative of Multivariate Functions

2.2.2 Directional Derivatives

2.2.3 Chain Rule

The chain rule is a concept in calculus to help us understand and calculate the derivative for composite functions which can be made up of two or more functions chained together.

Suppose that we have two function $f(x)$ and $g(x)$ which are both differentiable and define the composite function $h(x) = f(g(x))$ then the derivative of $h^{'}(x)$ is given by

$$h^{'}(x) = \frac{dh}{dx} \tag{2.58}$$

$$= f^{'}(g(x))g^{'}(x) \tag{2.59}$$

Consider that we have $y = f(u)$ and $u = g(x)$, a different definition of the composite function $y = f(g(x))$, then the derivative of y is,

$$h^{'}(x) = \frac{dh}{dx} \tag{2.60}$$

$$= \frac{dh}{du}\frac{du}{dx} \tag{2.61}$$

2.2.4 Hessian

2.3 Optimization

The optimization problem is a computational problem in which the objective is to find the best of all possible solutions. Deep neural networks is a form of an optimization problem to find the best possible set of weights in order to reduce the error in a network.

A generic form of an optimization problem is given by

$$\begin{aligned} \underset{x}{\text{minimize/maximize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned} \tag{2.62}$$

where

$f : \mathbb{R}^n \to \mathbb{R}$ is the objective/loss function to be minimized,

$g_i(x) \leq 0$ are called inequality constraints,

$h_j(x) = 0$ are called equality constraints, and

$m \geq 0$ and $p \geq 0$

If $m = p = 0$, the problem is an unconstrained optimization problem.

2.3.1 Optimization algorithms

The solutions to the optimization problem are vital in modern machine learning and artificial intelligence algorithms, which includes weight optimization in deep learning. There are a number of popular optimization algorithm currently developed

to solve the problem. Hence, choosing the right algorithm can be challenging as well. We explore few of the gradient-based solution to the optimization below.

Gradient-based methods are iterative methods that use the gradient information of the objective function during iterations.

Newton's Method

For minimizing $f(x)$, $x \in \mathbb{R}$, we need to solve $g(x) = f^{'}(x) = 0$. Newton's iteration is given by

$$x_{n+1} = x_n - \frac{g(xn)}{g^{'}(x_n)} \tag{2.63}$$

$$= x_n - \frac{f^{'}(xn)}{f^{''}(x_n)} \tag{2.64}$$

For multivariate functions we need to minimize $f(\mathbf{x})$ over $\mathbf{x} \in \mathbb{R}^n$, that it

$$\underset{\mathbf{x}\in\mathbb{R}^n}{min} f(\mathbf{x}), \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n \tag{2.65}$$

The Newton's iteration for multivariate function is given by

$$\mathbf{x_{n+1}} = \mathbf{x_n} - H(\mathbf{x_n})^{-1} \nabla f(\mathbf{x_n}) \tag{2.66}$$

where

$H(\mathbf{x_n})$ is the Hessian matrix of $f(\mathbf{x})$

We can observe from (equation ref) that calculating the inverse of the Hessian matrix can be computationally very expensive for higher dimensions. Replacing $H(\mathbf{x_n})^{-1}$ by $\alpha\mathcal{I}$ where $\mathcal{I}$ is the identity matrix; we get the **method of steepest descent** given by

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \alpha \mathbf{I} \nabla f(\mathbf{x_n}) \tag{2.67}$$

where $\alpha \in (0, 1)$

2.4 Deep Learning

Deep learning is part of Machine Learning which deals mainly with computers that can learn either on their own or supervised. The latter can solve problem in the industry ranging from computer vision (image), natural language processing (text), automatic speech recognition (audio), time-series prediction amongst others. Deep learning primarily uses the concept of artificial neural networks, which derivatives from how the human brain works, to solve complex linear and non-linear problem.

We start by introducing how a simple perceptron works, and gradually increase the complexity of the network until we reach a deep neural network that can solve non-linear problems.

2.4.1 Perceptron

A perceptron is an analogy to the human neuron. It is a computational model that takes an input (scalar or vector) and learns to classify the latter between two classes (binary classifier). It consists of an input, a weighed sum and an activation function. The mathematical definition a perceptron that maps its input $\mathbf{x}$ to an output value $f(\mathbf{x})$ using a step function as activation function is given by

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}.\mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.68}$$

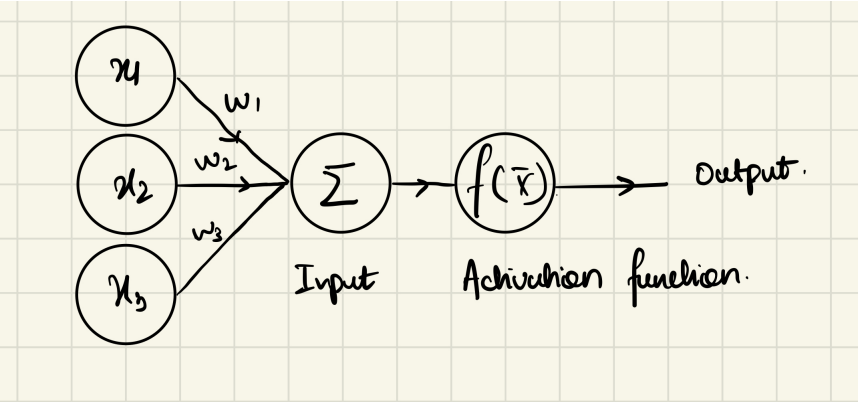

Figure 2.10: Perceptron graphical representation

From the perceptron mathematical definition 2.68, we can observe that the dot product is dependent on the angle between the weight vector ($\mathbf{w}$) and the input vector ($\mathbf{x}$). Using dot product (6.11), addition (6.7) and subtraction (6.8) of vector property; we can define a learning rule for modifying the perceptron to learn how to classify a set of input.

2.4.2 Perceptron Learning Ruelee

Consider the two sets of point **A** and **B**

$$\mathbf{A} = \left\{ \begin{bmatrix} 1 \\ 2 \end{bmatrix} \right\} \quad \mathbf{B} = \left\{ \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right\} \tag{2.69}$$

and fitting the perceptron function (2.68) to learn how to classify points in each respective set. Let set **A** and **B** be class 1 and 0 respectively.

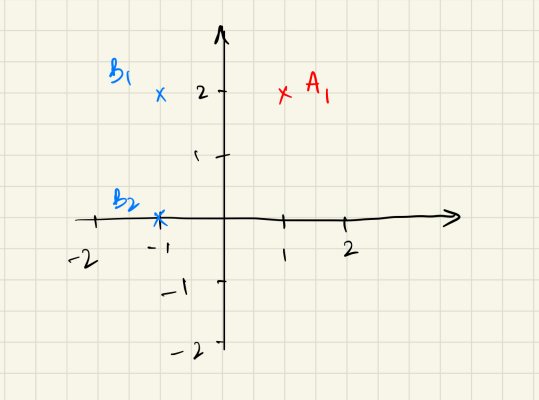

Figure 2.11: Caption

From 2.2, we can separate the two classes by a single line. Consider the random separating line passing through the origin (this will set the value of b in (2.68) to 0).

The equation of the random line is given by

$$x_1 = -x_2 \tag{2.70}$$

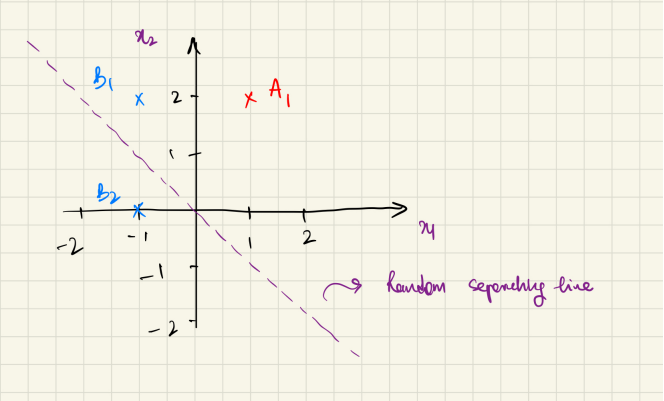

Figure 2.12: Caption

Comparing to the equation (2.68); we can deduce that

$$w_1 = 1 \quad w_2 = 1 \tag{2.71}$$

Visually, we can already see that the separating line does not split the points properly. The computation of the binary classification using the random line is given by

$$f(A_1) = f\left(\begin{bmatrix}1\\1\end{bmatrix} . \begin{bmatrix}1\\2\end{bmatrix}\right) = f(3) = 1 \tag{2.72}$$

$$f(B_1) = f\left(\begin{bmatrix}1\\1\end{bmatrix} . \begin{bmatrix}-1\\2\end{bmatrix}\right) = f(1) = 1 \tag{2.73}$$

$$f(B_2) = f\left(\begin{bmatrix}1\\1\end{bmatrix} . \begin{bmatrix}-1\\0\end{bmatrix}\right) = f(-1) = 0 \tag{2.74}$$

The computation confirms the visual representation and groups $\mathbf{A}_1$ and $\mathbf{B}_1$ together. We can either alter the separating line (by moving the weight vector) with respect to $\mathbf{A}_1$ and/or $\mathbf{B}_1$.

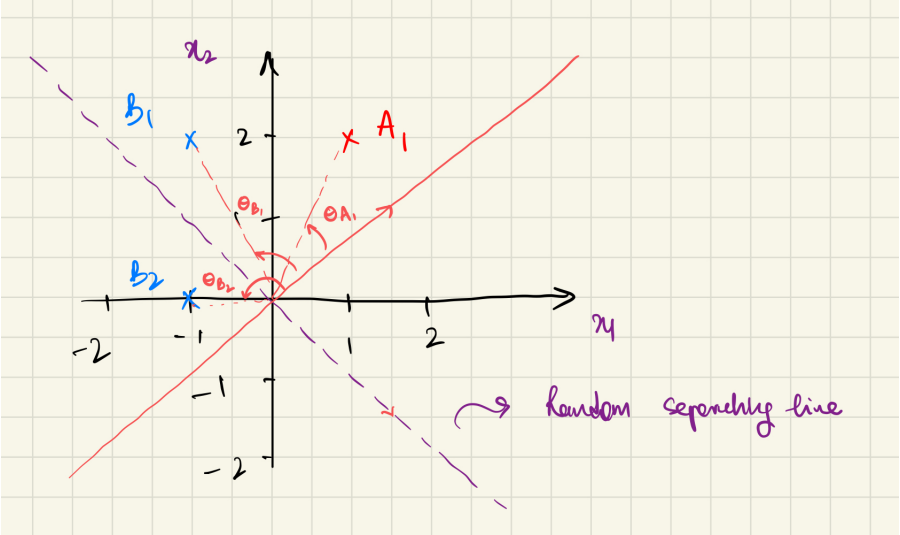

Figure 2.13: Caption

Using the property of subtraction of vectors; we move the weight vector ($\mathbf{w}$) away from $\mathbf{B}_1$ and check the classification again. We also introduce a learning rate α to control how much rotation we want. Let $\alpha = 0.5$

$$\mathbf{w}^{(1)} = \mathbf{w} - \alpha\mathbf{B}_1 \tag{2.75}$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.5 \begin{bmatrix} -1 \\ 2 \end{bmatrix} \tag{2.76}$$

$$= \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} \tag{2.77}$$

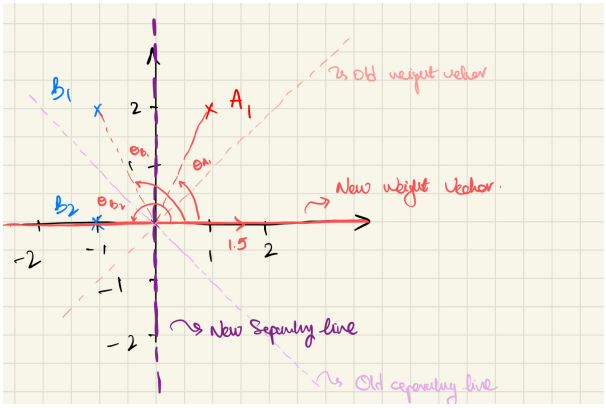

Figure 2.14: Caption

Based on the new separating line, we can observe visually that the points are correctly classified. The computation of the binary classification using the new separating line $\mathbf{w}^{(1)}$ is given by

$$f(A_1) = f\left(\begin{bmatrix}1.5\\0\end{bmatrix}.\begin{bmatrix}1\\2\end{bmatrix}\right) = f(1.5) = 1 \tag{2.78}$$

$$f(B_1) = f\left(\begin{bmatrix}1.5\\0\end{bmatrix}.\begin{bmatrix}-1\\2\end{bmatrix}\right) = f(-1) = 0 \tag{2.79}$$

$$f(B_2) = f\left(\begin{bmatrix}1.5\\0\end{bmatrix}.\begin{bmatrix}-1\\0\end{bmatrix}\right) = f(-1.5) = 0 \tag{2.80}$$

Thus, the perceptron learned how to classify the two sets of points. The function $f(\mathbf{x})$ could be re-used to classify new points added to the sets.

2.4.2.1 Perceptron Update Rule

We can define a set number of rules for the perceptron to find the optimal weight vector to classify the points. Based on the properties of the dot product; the classification of the points are dependent on the inner angle between the normal of the separating line and the vector to be classified. If the inner angle is less than 90° then the point belongs to class 1 else to class 0.

Target Classes and Errors

Let the target(actual) classes be $y = 1$ and $y = 0$. Then the network error ϵ is given by the difference between the true value and the predicted value.

$$\begin{aligned}
\text{Network Error} \quad & \epsilon(\mathbf{x}) = y - f(\mathbf{x}) \\
\text{Correct Classification} \quad & y = 1, f(\mathbf{x}) = 1 \rightarrow \epsilon(\mathbf{x}) = 0 \\
& y = 0, f(\mathbf{x}) = 0 \rightarrow \epsilon(\mathbf{x}) = 0 \\
\text{Incorrect Classification} \quad & y = 1, f(\mathbf{x}) = 0 \rightarrow \epsilon(\mathbf{x}) = 1 \\
& y = 0, f(\mathbf{x}) = 1 \rightarrow \epsilon(\mathbf{x}) = -1
\end{aligned}$$

We find that updating the weight vector can be written as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha\epsilon(\mathbf{x})\mathbf{x} \tag{2.81}$$

Perceptron Learning Algorithm

The steps for the perceptron to learn how to perform binary classification is given by the steps below. Let t be the t^{th} iteration when learning and b be in an input bias.

Assignment $\rightarrow$ Assign $t = 0$, $\mathbf{w}^{(0)} = (0, 0, \ldots, 0)$, $b^{(0)} = 0$

Start $\rightarrow$ For $t = 0, 1, 2, \ldots$ until convergence

Step 1 $\rightarrow$ Randomly choose a vector $\mathbf{x}^{(t)}$ with the corresponding $y^{(t)}$ (known)

Step 2 $\rightarrow$ Compute $\mathbf{z} = \mathbf{w}^{(t)}.\mathbf{x}^{(t)} + b^{(t)}$

Step 3 $\rightarrow$ Compute $\epsilon^{(t)}(\mathbf{z}) = y^{(t)} - f(\mathbf{z})$

Step 4 $\rightarrow$ Update weight $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha\epsilon^{(t)}(\mathbf{z})\mathbf{x}^{(t)}$

Step 5 $\rightarrow$ Update bias $b^{(t+1)} = b^{(t)} + \alpha\epsilon^{(t)}(\mathbf{z})$

The current perceptron we have defined is for a binary classification; we can also define multiclass perceptron and/or use different types of activation functions.

2.4.2.2 Limitation of the Perceptron

The perceptron alone is not always a good model to solve binary classification problem. It has certain limitation by nature of its definition. Consider the logic functions AND, OR and XOR across these 4 points:

$$\mathbf{x}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{x}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.82}$$

AND Function

Placeholder

OR Function

The OR function returns a value of 1 if either of the inputs is 1.

$$f^*(\mathbf{x}_1) = 0, \quad f^*(\mathbf{x}_2) = 1, \quad f^*(\mathbf{x}_3) = 1, \quad f^*(\mathbf{x}_4) = 1 \tag{2.83}$$

From figure 2.7; we can easily define a separating line by the perceptron to separate

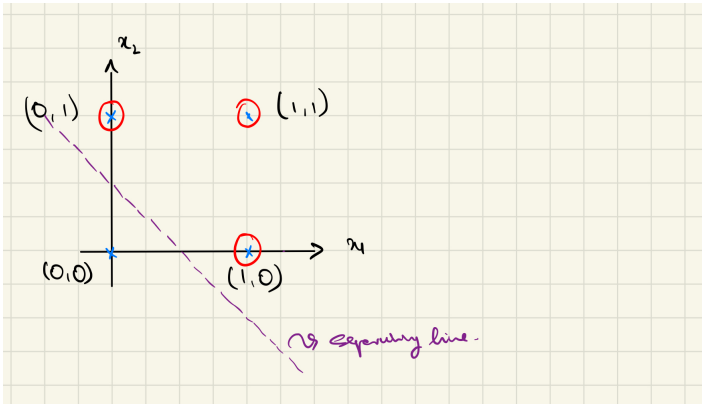

Figure 2.15: OR function

the two classes.

XOR Function

The XOR function returns a value of 1 if either of the inputs is 1 but not both.

$$f^*(\mathbf{x}_1) = 0, \quad f^*(\mathbf{x}_2) = 1, \quad f^*(\mathbf{x}_3) = 0, \quad f^*(\mathbf{x}_4) = 1 \tag{2.84}$$

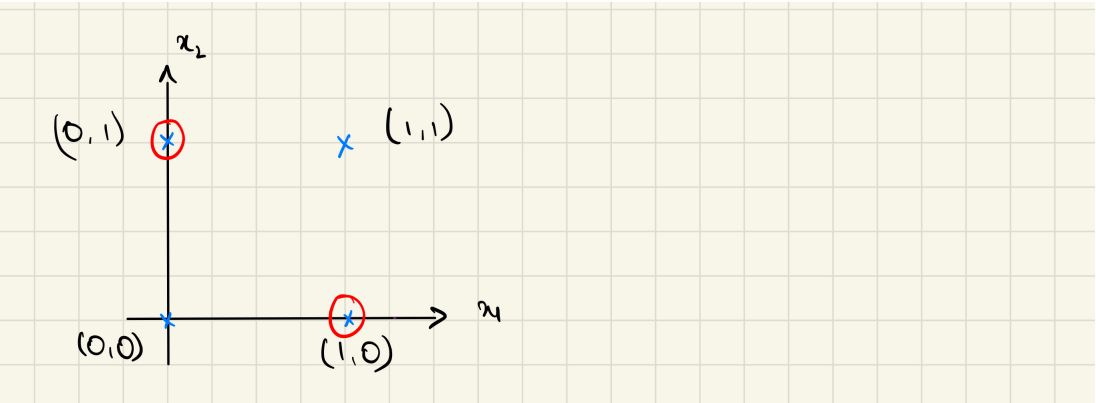

Figure 2.16: OR function

From figure 2.8; we can observe that no separating line alone could group the two different classes together. The perceptron model cannot solve the XOR logic function. Thus, we introduce **neural networks** to overcome this problem.

2.4.3 Neural Network

A neural network is a set of perceptron connected together which takes an input, manipulate the information to learn from it and outputs a prediction. A neural network attempts to learn a mapping from an input to an output. The goal is to reduce the error between the prediction by the network compared to the true value as much as possible.

The neural network model can be represented by a series of matrix multiplication which takes the input and feeds the network. This is also known as a **feedforward network**.

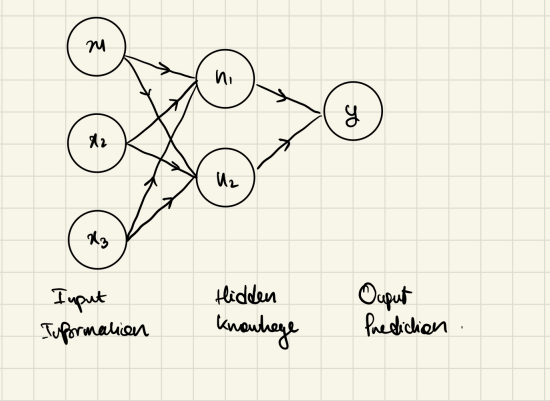

Figure 2.17: Caption

2.4.3.1 Feedforward Network

Consider the simple network below with 2 input neurons, 2 perceptrons and 1 output.

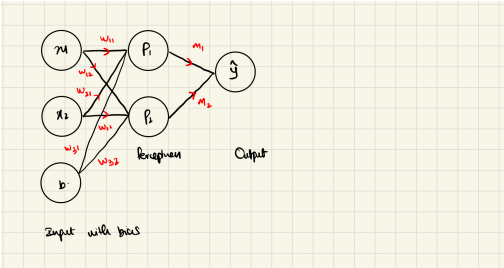

Figure 2.18: Caption

The feedforward network takes the information at the input and forwards it to the next layer after applying the weight vector for each respective hidden layer. This operation can be expressed as a series of matrix multiplication defined next. The input vector is given by $\mathbf{x}$, random weight matrices by $\mathbf{W}_i$ and the actual output be y.

$$\mathbf{x} = \begin{bmatrix} x_{11} \\ x_{21} \\ b \end{bmatrix} \quad \mathbf{W}_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \quad \mathbf{W}_2 = \begin{bmatrix} m_{11} \\ m_{21} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y \end{bmatrix} \tag{2.85}$$

The feedforward neural network for a single input vector can then be expressed as a series matrix multiplication to estimate the actual value of $\mathbf{y}$. Let $\widehat{y}$ be the estimated value of y.

$$\begin{aligned}\widehat{y} &= f\left(f(\mathbf{x}^T\mathbf{W}_1)\mathbf{W}_2\right) \\ &= f\left(f\left(\begin{bmatrix} x_{11} & x_{21} & b \end{bmatrix}\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}\right)\begin{bmatrix} m_{11} \\ m_{21} \end{bmatrix}\right)\end{aligned} \tag{2.86}$$

where f is the function (2.68).

Solution to the XOR function

One solution to the XOR function can be obtained using the neural network (2.86) with the set go weights given by

$$\mathbf{x} = \begin{bmatrix} x_{11} \\ x_{21} \\ -\frac{1}{2} \end{bmatrix} \quad \mathbf{W}_1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{W}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y \end{bmatrix} \tag{2.87}$$

Verifying the Solution

$$\begin{aligned} &\begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \ \widehat{y} = 0 \quad \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \ \widehat{y} = 1 \\ &\begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \ \widehat{y} = 1 \quad \begin{bmatrix} x_{11} \\ x_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow \ \widehat{y} = 0 \end{aligned} \tag{2.88}$$

The neural network correctly classifies all input. While feed-forward network of this form helped us to solve the XOR function it is very limited. The architecture defined can only be used for linearly separable problem. In order to allow for non-linear solution; we need to introduce some non-linearity in the model. This is done by using different kinds of **activation functions** in the perceptrons.

2.4.3.2 Activation Functions

Activation functions can be used with a perceptron to introduce non-linearity. The function to be used is subjective to the problem being solved and the form of the desired result we want. Some activation functions are defined below.

Linear

$$\phi(z) = z \tag{2.89}$$

Unit Step (Heaviside Function)

$$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases} \tag{2.90}$$

Signum

$$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases} \tag{2.91}$$

Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.92}$$

Hyperbolic Tangent(tanh)

$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.93}$$

ReLU

$$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases} \tag{2.94}$$

2.4.3.3 Network Error

The network error is given by the squared difference between the predicted value and the true value. The difference is squared to avoid the sum of errors of multiple input vector to be zero which can mislead the network to have perfect predictive power.

$$\text{Network Error } e = \frac{1}{2} \sum_{i}^{\text{Total Input}} (\widehat{y_i} - y_i)^2 \tag{2.95}$$

The goal of the network is to adjust the weight to reduce the network error as much as possible. The idea of reducing a function to a value is synonymous to (2.62) an optimization problem. The network error in (2.95) is a quadratic equation.

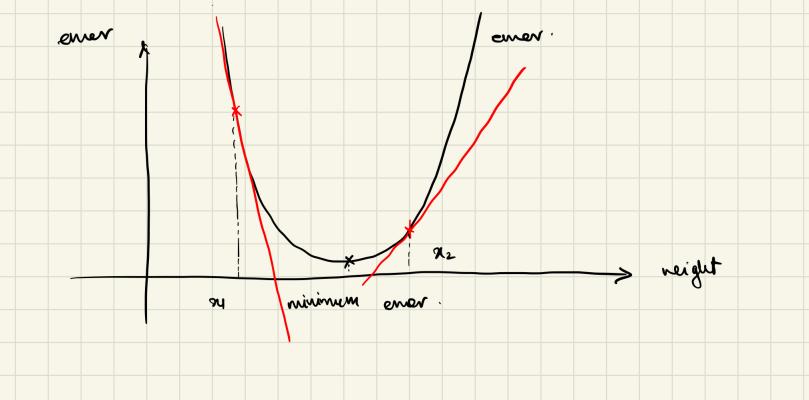

Figure 2.19: Network Error

Figure (2.19) is a graphical representation of the network error function. Our goal

is to reach the minimum of the function by adjusting the weight value. We use the **gradient descent method** to reach the bottom of the curve.

2.4.3.4 Gradient Descent Method

In order to reach the bottom of the function; we need to adjust the weight such that the derivative of the error function is 0. The approach of updating the weight based on the gradient of the error function is known as **gradient descent**. The derivative with respect to the weights of the network error (2.95) for a single input is given by

$$\frac{de}{dw_{ij}} = (\widehat{y_i} - y_i)\frac{d\widehat{y_i}}{dw_{ij}} \tag{2.96}$$

From figure (2.19); we can observe that if the gradient is negative then we have underestimated the predicted value and need to increase the weight to reach the optimal value. On the other hand, if the gradient is positive then we have overestimated the predicted value and need to decrease the weight to reach the optimal value. The equation (2.96) provides us with the opposite direction and amount to adjust the weight. Hence, the update rule of the weights for gradient descent method is given by

$$w_{ij}^{t+1} = w_{ij}^{t} - \frac{de}{dw_{ij}} \tag{2.97}$$

In the gradient descent method, the network learns from the gradient of the error function and adjust the weights accordingly to reduce the error. However, the gradient alone can be quite large, causing oscillations as we go down the error function. We fix this problem by introducing a **learning rate** α prior to adjusting the weight.

2.4.3.5 Learning Rate

The learning rate is typically denoted by the Greek letter alpha α. It helps the network to control the rate at which the weights are changing. Having a system with a high learning rate may lead to an oscillating network when trying to find the

optimal weight and having a slow learning rate increases the number of iterations required when optimizing the network. The adjusted update rule of the weight is given by

$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{de}{dw_{ij}} \tag{2.98}$$

where $\alpha \in (0, 1)$

2.4.3.6 Back Propagation

The input data has been fed forward in the network. Then the error function and the gradient descent method we have defined are both executed at the end of neural network we have defined in equation (2.86). We now need a way to back propagate the weight changes that need to happened from the output neuron to the input neuron in between each layer. This whole process is known as **back propagation**. Consider the neural network below.

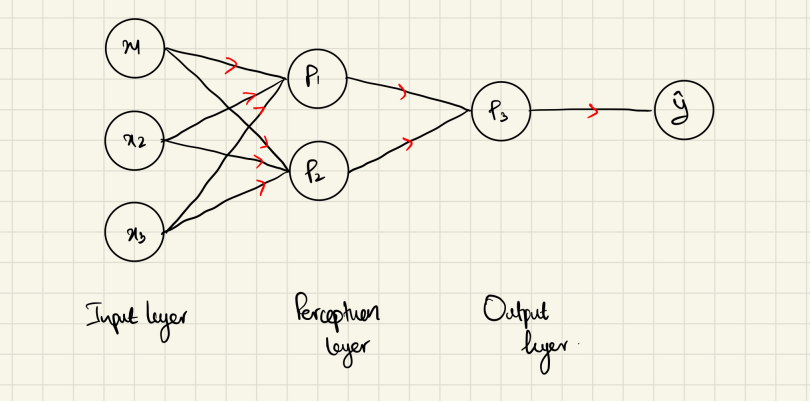

Figure 2.20: Back propagation

Let the perceptron at output layer have a linear activation function while the per-

ceptron layer have ReLU activation functions.

- Explain what happens at the end of the neural network - Explain what happens in between layers - Explain how to back propagate to the start of the network.

2.4.4 Deep Neural Network

2.4.5 Overfitting

2.4.5.1 Dropout

2.4.6 Convolutional Layer

Chapter 3

Sentiment Analysis

Chapter 4

PyTorch

Chapter 5

Tweet Data

5.1 Tweets

5.1.1 Tweepy

5.1.1.1 Scrapping using Tweepy

5.1.1.2 Cleaning using RegEx

5.2 Tweet Count

5.3 Tweet Search Volume Index (SVI)

5.4 Feature Engineering

Bibliography

AuthorName (Year*a*), Conferencetitle, *in* 'BookTitle'.

AuthorName (Year*b*), 'Title of article', *Journal* .

BookAuthors (Year), *BookTitle*, Publisher.

- McCullough - First Neural Network Paper 1943 - F. Rosenblatt - 1958 - 1962

Chapter 6

Placeholder - to be deleted

6.1 Linear Algebra

We introduce briefly the realm of linear algebra to get a better understanding of more complex concepts in neural networks. The focus is set on mainly the properties useful in deep learning.
Linear algebra is the study of linear equations of the form:

$$x_1 a_1 + \cdots + x_n a_n = b \tag{6.1}$$

or as a linear map defined by the following:

$$(x_1, \ldots, x_n) \mapsto (a_1 x_1 + \cdots + a_n x_n) \tag{6.2}$$

and their representation, manipulation and operation using vectors and matrices within a set of predefined axioms.

6.1.1 Vectors

A vector is quantity that has both a magnitude and direction. It can be represented by an ordered set of numbers or graphically using an arrow. Consider the vector **u**

in $\mathbb{R}^2$ given by:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{6.3}$$

The vector can be expressed graphically as shown below.

Image Placeholder vector in R^2

The length of the arrow is an indication of the magnitude of a vector and its orientation given by the direction in which the arrow is pointing. The magnitude of the above vector can be computed as such:

$$||\mathbf{u}|| = \sqrt{u_1^2 + u_2^2} \tag{6.4}$$

The direction of the vector with respect to the x-axis is given by:

$$\theta = \tan^{-1}\left(\frac{u_2}{u_1}\right) \tag{6.5}$$

A vector space is a set of vectors which satisfy the following axioms:

Placeholder - Table of Axioms for Vector Space

Geometry representation of addition and subtraction of vectors in $\mathbb{R}^2$
Vector addition and subtraction is performed componentwise along each element of two vectors. Consider the two vectors **a** and **b** where:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{6.6}$$

Addition

The addition of vector **u** and **v** is given by:

$$\mathbf{u} + \mathbf{v} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \end{bmatrix} \tag{6.7}$$

The geometrical effect of adding **u** and **v** together is shown below.

Image Placeholder vector addition in R^2

We can observe from (image ref) above that adding **u** to **v** has the effect of rotating **u** towards **v**.

Subtraction

The subtraction of vector **v** from **u** is given by:

$$\mathbf{u} - \mathbf{v} = \begin{bmatrix} u_1 - v_1 \\ u_2 - v_2 \end{bmatrix} \tag{6.8}$$

Image Placeholder vector subtraction in R^2

We can observe from (image ref) above that subtracting **v** from **u** has the effect of rotating **u** in the opposite direction of **v**.

The effect of rotating two vectors though addition and subtraction are useful properties utilized in deep learning.

6.1.1.1 Inner Product

The inner product or dot product of:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{6.9}$$

is given by **u**.**v**

$$\mathbf{u.v} = u_1 v_1 + u_2 v_2 \tag{6.10}$$

$$= ||\mathbf{u}|| \, ||\mathbf{v}|| \cos\theta \tag{6.11}$$

where θ is the angle between vectors **u** and **v**.

Image Placeholder inner product in R^2

The dot product of **u.v** equals **v.u**. The order does not make a difference.

6.1.2 Matrices

A matrix is an $m \times n$ array of numbers, where m is the number of rows and n is the number of columns; the matrix is said to be of dimension $(m \times n)$.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m_2} & \dots & a_{mn} \end{bmatrix} \tag{6.12}$$

Similar to vectors, there are a couple of matrix operations that can be done under certain condition.

Matrix Operations

Matrix Addition

The sum of two matrices $\mathbf{A}$ and $\mathbf{B}$ is given by the element wise sum of the elements provided that they both have the same dimension.

Consider two matrices $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \tag{6.13}$$

the sum $\mathbf{A} + \mathbf{B}$ is given by

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix} \tag{6.14}$$

Scalar Matrix Multiplication

The product $\alpha\mathbf{A}$, where $\alpha \in \mathbb{R}$ and $\mathbf{A}$ is a matrix, is calculated by multiplying every entry of $\mathbf{A}$ by α.

Consider a matrix $\mathbf{A}$ and constant α:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{6.15}$$

the scalar multiplication $\alpha\mathbf{A}$ is given by

$$\alpha\mathbf{A} = \alpha \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \alpha a_{11} & \alpha a_{12} \\ \alpha a_{21} & \alpha a_{22} \end{bmatrix} \tag{6.16}$$

Matrix Transpose

The transpose of a $(m \times n)$ matrix $\mathbf{A}$ is calculated by swapping the rows into columns or the columns into rows. This operation result in a $(n \times m)$ denoted as $\mathbf{A}^T$.
Consider a matrix $\mathbf{A}$, the transpose is given by $\mathbf{A}^T$:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix} \tag{6.17}$$

Matrix Multiplication

The multiplication operation between two matrices is only defined is the number of columns of the left matrix is the same as the number of rows in the right matrix. If $\mathbf{A}$ is a $(m \times n)$ matrix and $\mathbf{B}$ is a $(n \times p)$ matrix, then their product $\mathbf{AB}$ is the $(m \times n)$ matrix whose entries are given by inner product of the corresponding row of $\mathbf{A}$ and the corresponding column of $\mathbf{B}$.
Consider two matrices $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \tag{6.18}$$

the matrix multiplication is given by $\mathbf{AB}$

$$\mathbf{AB} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \tag{6.19}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix} \tag{6.20}$$

6.1.2.1 Outer Product

The outer product of two vectors $\mathbf{u}$ and $\mathbf{v}$:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{6.21}$$

is given by $\mathbf{u} \otimes \mathbf{v}$

$$\mathbf{u} \otimes \mathbf{v} = \begin{bmatrix} u_1 v_1 & u_1 v_2 \\ u_2 v_1 & u_2 v_2 \end{bmatrix} \tag{6.22}$$

The outer product of $\mathbf{u} \otimes \mathbf{v}$ is not equal to $\mathbf{v} \otimes \mathbf{u}$; the order does matter. However, the transpose of latter outer product are the same:

$$\mathbf{u} \otimes \mathbf{v} = (\mathbf{v} \otimes \mathbf{u})^T \tag{6.23}$$

Chapter 7

test