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.

2.2 Perceptron

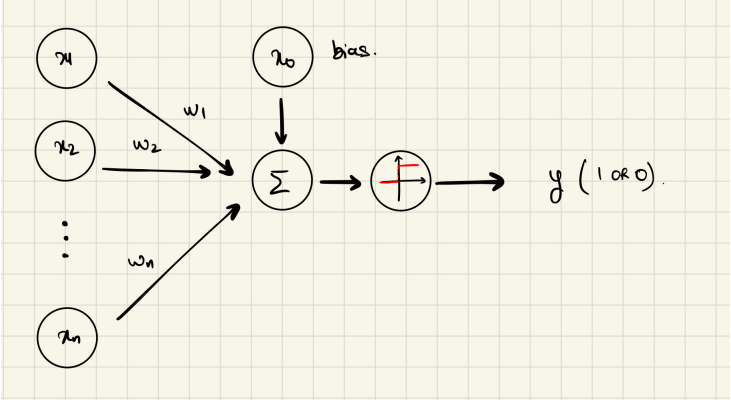

Figure 2.1: Rosenblatt perceptron

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.

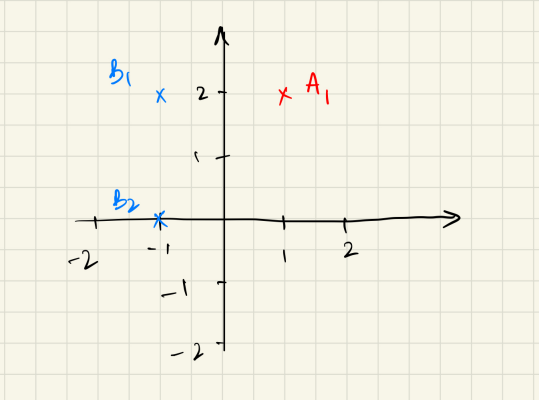

Figure 2.2: Caption

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).

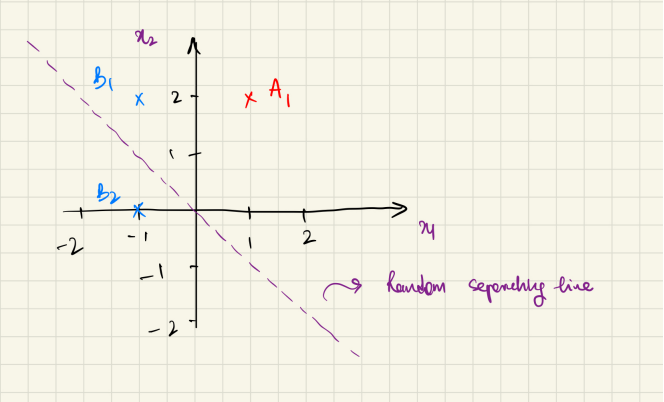

Figure 2.3: Caption

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 $\mathbf{B}_1$.

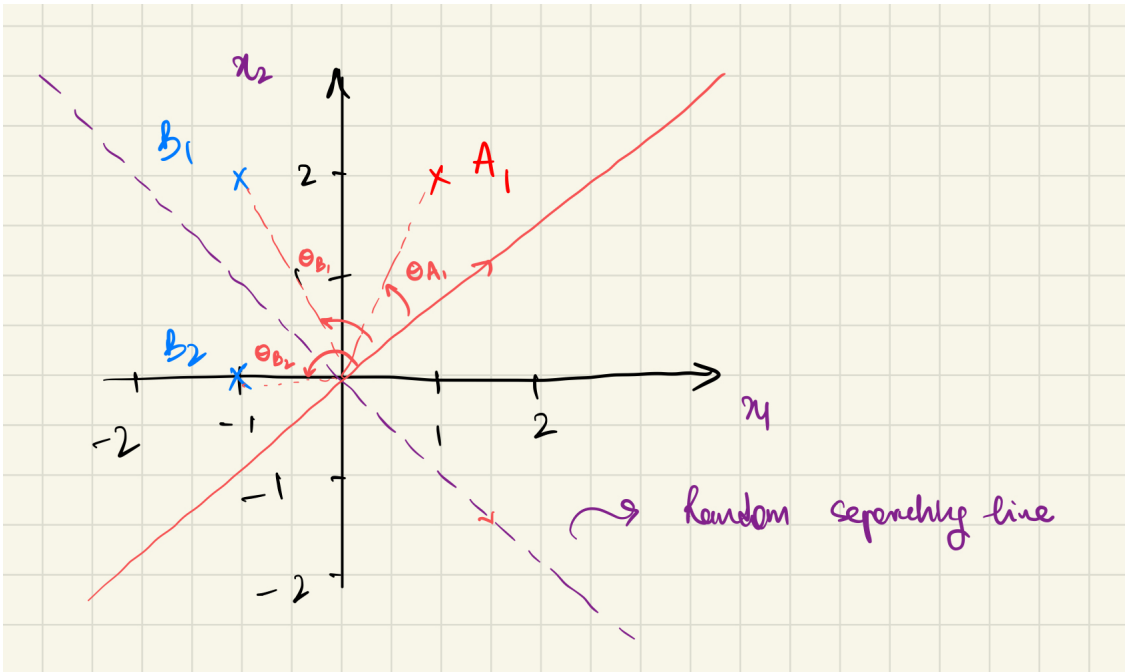

Figure 2.4: !!!TO BE CHANGED!!!

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.

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

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

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

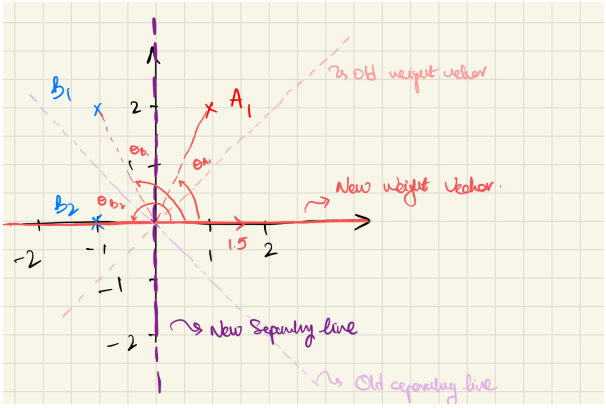

Figure 2.5: !!!!TO BE CHANGED!!!!

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**.

2.3 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.

2.4 Solving Gate problem

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:

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

2.4.1 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

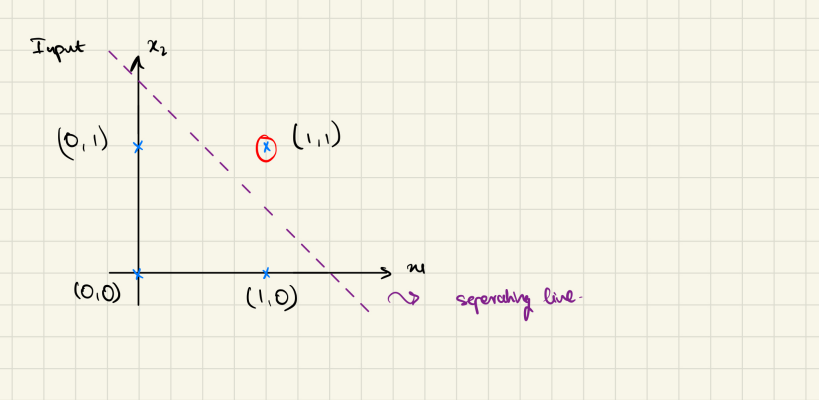

Figure 2.6: AND Function with separating line

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.

2.4.2 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

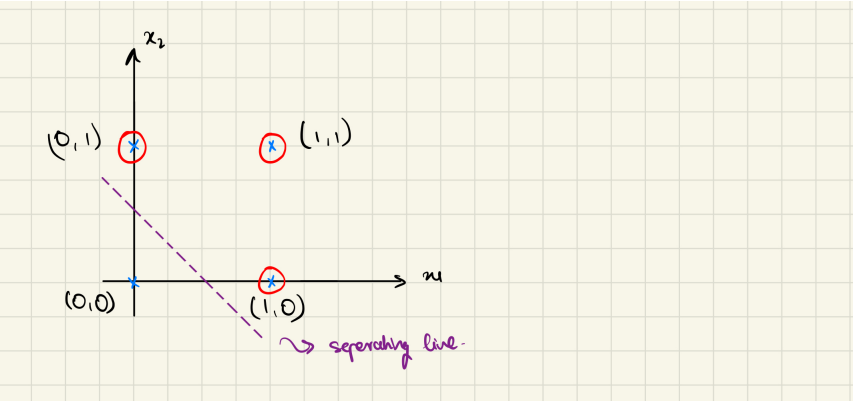

Figure 2.7: AND Function with separating line

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.

2.4.3 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

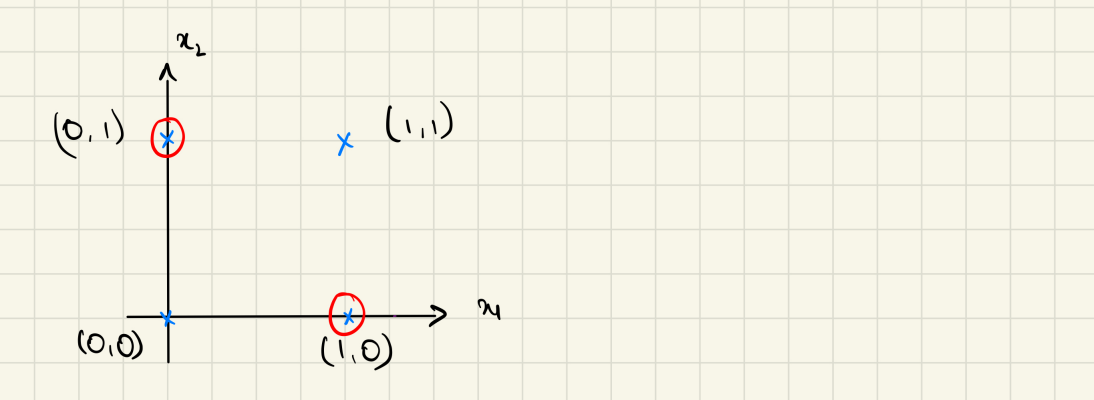

Figure 2.8: OR function

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

$$\begin{aligned}2 &= 2w_1 + 2w_2 + 4b \\ w_1 + w_2 + 2b &= 1 \qquad (2.49)\end{aligned}$$

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

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

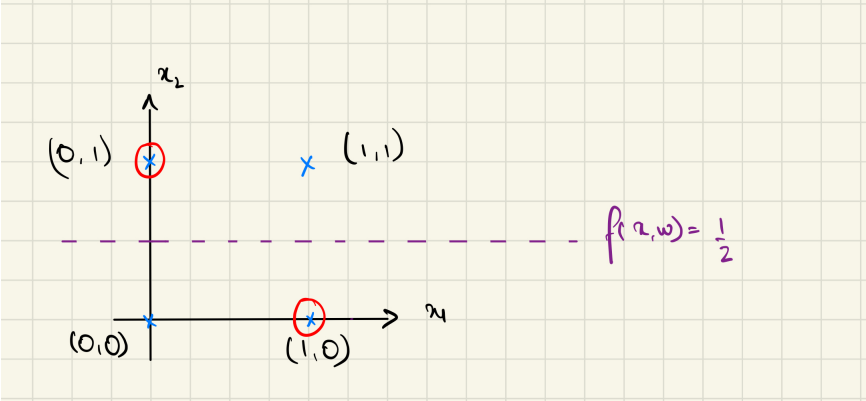

Figure 2.9: XOR approximate solution

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.

2.5 Feedforward Network

We have seen that the single perceptron cannot solve the XOR problem. In order to solve the problem, we use a set of perceptrons that learns two different functions within a connected network.

First, let's consider two NOT boolean function $f_1^* = x_1$ AND (NOT x_2) and $f_2^* =$ (NOT x_1) AND x_2.

Consider f_1^* which returns 1 if x_1 and (not x_2) are equal to 1.

x_1	x_2	NOT (x_2)	x_1 AND (NOT x_2)
0	0	1	0
1	0	1	1
1	1	0	0
0	1	0	0

Table 2.1: Truth Table for f_1^*

The function f_1^* can be separated using the perceptron.

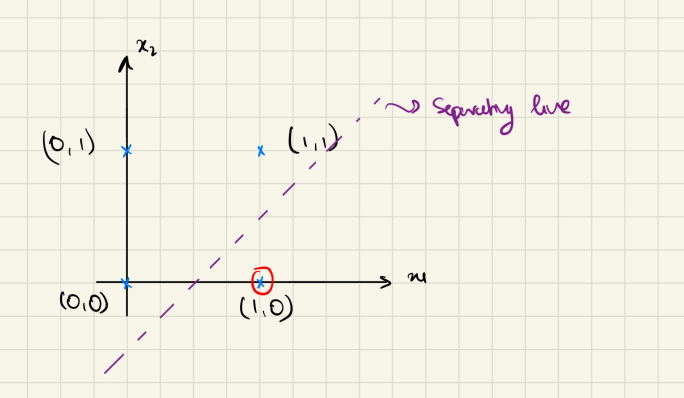

Figure 2.10: f_1

The equation of the separating line in figure 2.10 is given by

$$x_1 - x_2 - \frac{1}{2} = 0 \tag{2.56}$$

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

$$\mathbf{w} = \begin{bmatrix} -0.5 \\ 1 \\ -1 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \tag{2.57}$$

Similarly, consider f_2^* which returns 1 if (not x_1) and x_2 are equal to 1.

x_1	x_2	NOT (x_1)	(NOT x_1) AND x_2
0	0	1	0
1	0	0	0
1	1	0	0
0	1	1	1

Table 2.2: Truth Table for f_2^*

The function f_2^* can be separated using the perceptron.

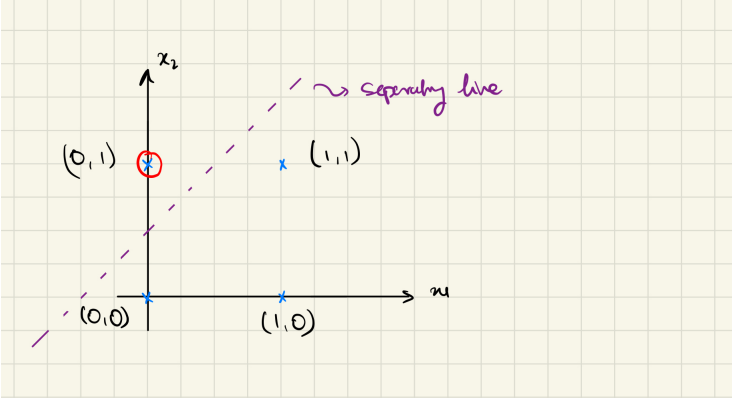

Figure 2.11: f_1

The equation of the separating line in figure 2.11 is given by

$$-x_1 + x_2 - \frac{1}{2} = 0 \tag{2.58}$$

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

$$\mathbf{w} = \begin{bmatrix} -0.5 \\ -1 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \tag{2.59}$$

Consequently, the XOR function defined in (2.42) can be expressed as f^* such that

$$f^*(x_1, x_2) = f_1^*(x_1, x_2) \text{ OR } f_2^*(x_1, x_2)$$

x_1	x_2	$f_1^*(x_1, x_2)$	$f_2^*(x_1, x_2)$	$f_2^*(x_1, x_2)$
0	0	0	0	0
1	0	1	0	1
1	1	0	0	0
0	1	0	1	1

Table 2.3: Truth Table for f^*

Consider a feedforward network with a hidden layer with two units h_1 and h_2 such that h_1 learns $f_1^*(x_1, x_2)$ and h_2 learns $f_2^*(x_1, x_2)$.

$$\begin{aligned} h_1 &= f_1^*(x_1, x_2) \\ &= \phi(z_1) \end{aligned}$$

where $z_1 = w_{11}x_1 + w_{12}x_2 + b_1$ and $w_{11} = 1$, $w_{12} = -1$ and $b_1 = -0.5$ from separating line in (2.57).

$$\begin{aligned} h_2 &= f_2^*(x_1, x_2) \\ &= \phi(z_2) \end{aligned}$$

where $z_2 = w_{21}x_1 + w_{22}x_2 + b_1$ and $w_{21} = -1$, $w_{22} = 1$ and $b_2 = -0.5$ from separating line in (2.59).

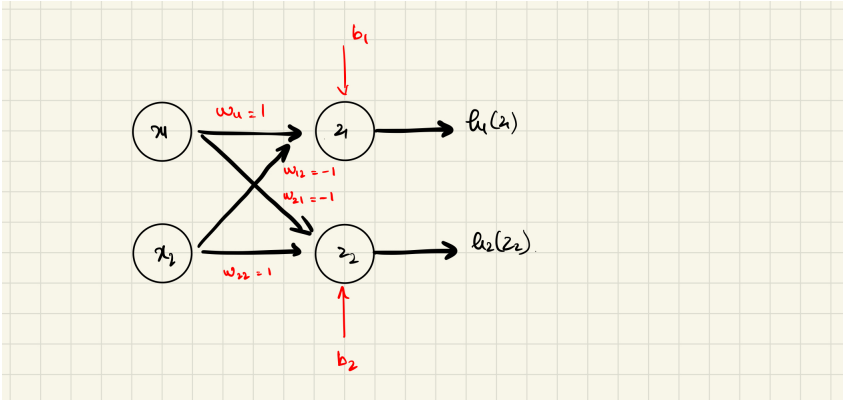

Figure 2.12: FFN hidden layer

Thus we obtain the system below for the hidden layer.

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} b_1 & w_{11} & w_{12} \\ b_1 & w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \tag{2.60}$$

We then feed forward the information from the hidden layer to another layer.

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \phi \left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \right) \tag{2.61}$$

At the hidden layer, we are left with the following truth table.

x_1	x_2	XOR	
0	0	0	
1	0	1	
0	0	0	redundant
0	1	1	

Table 2.4: Truth Table from hidden layer

The XOR problem can not be separated from using another perceptron after the hidden layer.

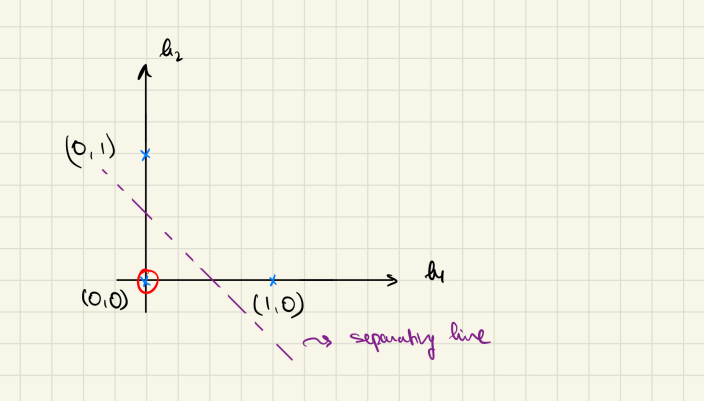

Figure 2.13: XOR layer

The equation of the separation line is in figure (2.13) is given binary

$$h_1 + h_2 - \frac{1}{2} = 0 \tag{2.62}$$

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

$$\mathbf{w} = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{h} = \begin{bmatrix} 1 \\ h_1 \\ h_2 \end{bmatrix} \tag{2.63}$$

Finally, the XOR function can be represented using the set of transform defined below.

$$f(x_1, x_2) = \phi' \left(\mathbf{w}_2^T \left(\phi \left(\mathbf{w}_1^T \mathbf{x} \right) \right) \right) \tag{2.64}$$

where

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \quad \mathbf{w}_1 = \begin{bmatrix} -0.5 & -0.5 \\ 1 & -1 \\ -1 & 1 \end{bmatrix} \quad \mathbf{w}_2 = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix} \tag{2.65}$$

ϕ' taken an augmented input from the calculation of $\phi(\mathbf{w}_1^T\mathbf{x})$ to accommodate for the bias.

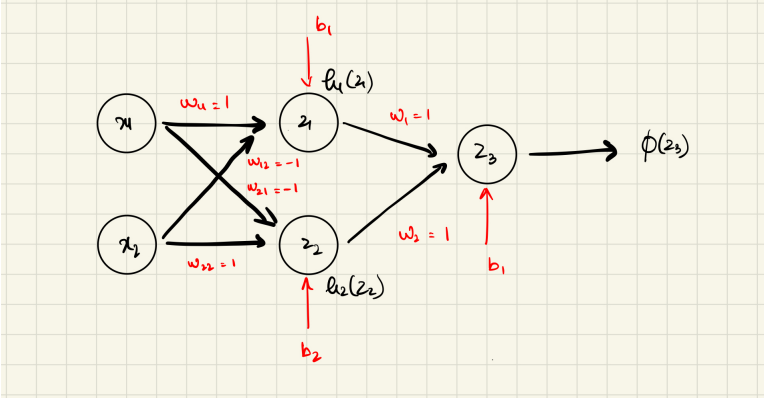

Figure 2.14: Feedforward network for XOR problem

Checking for $\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$

$$\mathbf{w}_1^T\mathbf{x} = \begin{bmatrix} -0.5 & 1 & -1 \\ -0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}$$

$$\phi(\mathbf{w}_1^T\mathbf{x}) = \phi\left(\begin{bmatrix} -0.5 \\ -0.5 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\phi^{'}\left(\mathbf{w}_2^T\left(\phi\left(\mathbf{w}_1^T\mathbf{x}\right)\right)\right) = \phi^{'}\left(\begin{bmatrix} -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right) = \phi^{'}(-0.5)$$

Thus, for $\mathbf{p}_1 = \begin{bmatrix} 0, 0 \end{bmatrix}^T$

$$f(x_1, x_2) = \phi^{'}(-0.5) = 0$$

Checking for $\mathbf{x} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$

$$\mathbf{w}_1^T\mathbf{x} = \begin{bmatrix} -0.5 & 1 & -1 \\ -0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 0.5 \end{bmatrix}$$

$$\phi(\mathbf{w}_1^T\mathbf{x}) = \phi\left(\begin{bmatrix} -1.5 \\ 0.5 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\phi^{'}\left(\mathbf{w}_2^T\left(\phi\left(\mathbf{w}_1^T\mathbf{x}\right)\right)\right) = \phi^{'}\left(\begin{bmatrix} -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right) = \phi^{'}(0.5)$$

Thus, for $\mathbf{p}_4 = \begin{bmatrix} 0, 0 \end{bmatrix}^T$

$$f(x_1, x_2) = \phi^{'}(0.5) = 1$$

2.6 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.

2.6.1 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

$$f^{'}(a) = \frac{df}{dx}(a) \tag{2.66}$$

$$= \lim_{\Delta \to 0} \frac{f(a+\Delta) - f(a-\Delta)}{2 \times \Delta} \tag{2.67}$$

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

2.6.2 Directional Derivatives

2.6.3 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

$$h^{'}(x) = \frac{dh}{dx} \tag{2.68}$$

$$= f^{'}(g(x))g^{'}(x) \tag{2.69}$$

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,

$$h^{'}(x) = \frac{dh}{dx} \tag{2.70}$$

$$= \frac{dh}{du}\frac{du}{dx} \tag{2.71}$$

2.6.4 Hessian

2.7 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

$$\begin{aligned} \underset{x}{\text{minimize/maximize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned} \tag{2.72}$$

where

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

$g_i(x) \leq 0$ 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.

2.7.1 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

$$x_{n+1} = x_n - \frac{g(xn)}{g^{'}(x_n)} \tag{2.73}$$

$$= x_n - \frac{f^{'}(xn)}{f^{''}(x_n)} \tag{2.74}$$

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

$$\underset{\mathbf{x}\in\mathbb{R}^n}{min} f(\mathbf{x}), \quad \mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n \tag{2.75}$$

The Newton's iteration for multivariate function is given by

$$\mathbf{x_{n+1}} = \mathbf{x_n} - H(\mathbf{x_n})^{-1} \nabla f(\mathbf{x_n}) \tag{2.76}$$

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

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \alpha \mathbf{I} \nabla f(\mathbf{x_n}) \tag{2.77}$$

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

2.8 Deep Learning

2.8.1 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

$$\phi(z) = z \tag{2.78}$$

Unit Step (Heaviside Function)

$$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases} \tag{2.79}$$

Signum

$$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases} \tag{2.80}$$

Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.81}$$

Hyperbolic Tangent(tanh)

$$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.82}$$

ReLU

$$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases} \tag{2.83}$$

2.8.1.1 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.

$$\text{Network Error } e = \frac{1}{2} \sum_{i}^{\text{Total Input}} (\widehat{y_i} - y_i)^2 \tag{2.84}$$

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 (2.72) an optimization problem. The network error in (2.84) is a quadratic equation.

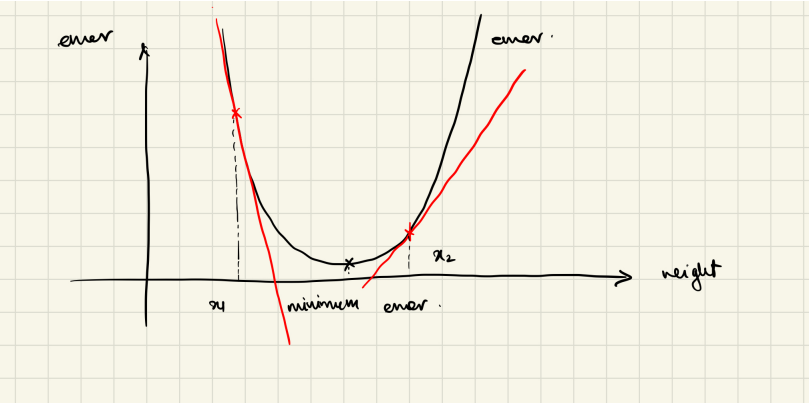

Figure 2.15: Network Error

Figure (2.15) 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.

2.8.1.2 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 (2.84) for a single input is given by

$$\frac{de}{dw_{ij}} = (\widehat{y}_i - y_i)\frac{d\widehat{y}_i}{dw_{ij}} \tag{2.85}$$

From figure (2.15); 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 (2.85) 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

$$w_{ij}^{t+1} = w_{ij}^{t} - \frac{de}{dw_{ij}} \tag{2.86}$$

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.

2.8.1.3 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

$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{de}{dw_{ij}} \tag{2.87}$$

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

2.8.1.4 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 a neural network. 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.

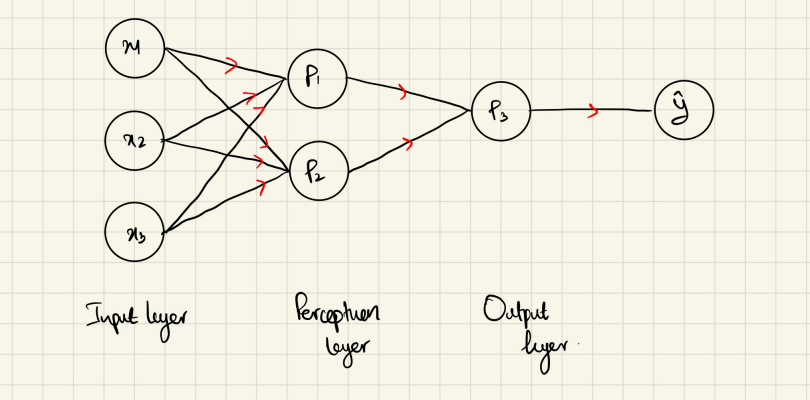

Figure 2.16: Back propagation

2.8.2 Deep Neural Network

2.8.3 Overfitting

2.8.3.1 Dropout

2.8.4 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