

# M01S01 – Structure of Neural Networks

## Introduction to Neural Networks

1. Introduction
2. Neural Networks - Inspiration from the Human Brain
3. Introduction to Perceptron
4. Binary Classification using Perceptron
5. Perceptrons - Training
6. Multiclass Classification using Perceptrons
7. Working of a Neuron
8. Inputs and Outputs of a Neural Network I
9. Inputs and Outputs of a Neural Network II
10. Assumptions made to Simplify Neural Networks
11. Parameters and Hyperparameters of Neural Networks
12. Activation Functions
13. Summary
14. Graded Questions

## Introduction

Most sophisticated and cutting-edge models in machine learning - **Artificial Neural Networks** or **ANNs**. Inspired by the structure of the human brain, neural networks have established a reputation for successfully learning complex tasks such as object recognition in images, automatic speech recognition (ASR), machine translation, image captioning, video classification etc.

Working principles of artificial neural networks - information flow in **feedforward networks**, training using **backpropagation**, **optimization techniques**, **hyperparameter tuning** etc. Also learn the **practical aspects** of training large neural networks.

Last session covers implementing neural networks using Keras (a high-level, user-friendly interface) with Tensorflow (the low-level workhorse library) as the backend.

Finally, **build your own neural network** from scratch using Numpy as assignment.

### In this session

Get an intuitive idea about neural networks

1. The inspiration for ANNs from the brain
2. The perceptron - a simple idea as the basis for larger neural networks
3. How artificial neurons work
4. The structure and topology of neural networks
5. Hyperparameters and simplifying assumptions

## Neural Networks - Inspiration from the Human Brain

Design of **Artificial Neural Networks (ANNs)** is inspired by the animal brain. Although not as powerful as the brain (yet), artificial neural networks are the most powerful learning models in the field of machine learning.

In the past few years, **deep artificial neural networks** have proven to perform surprisingly well on complex tasks such as speech recognition (converting speech to text), machine translation, image and video classification, etc. Such models are also commonly called **deep learning** models.

<b>Bottlenecks with Neural Networks</b> <ol style="list-style-type: none"><li>1. Availability of Data</li><li>2. Training Methodology</li><li>3. Computational Power</li></ol>	<b>Deep Learning Applications</b> <ol style="list-style-type: none"><li>1. Image Recognition</li><li>2. Image tagging and video analysis</li><li>3. Auto text generation</li><li>4. Annotations for text and video</li><li>5. Speech recognition</li><li>6. Grammar change recommendations</li><li>7. Translating text</li><li>8. Automating games</li><li>9. Search text and draw inferences from it</li></ol>
--	---

The main bottleneck in using neural networks is the availability of abundant training data. Neural networks have applications across various domains such as images and videos (computer vision), text, speech, etc. Deep learning and neural networks are often used interchangeably.

Artificial neural networks are said to be inspired by the structure of the brain. Let's first understand the basic structure of the brain, the structure of a neuron and how information travels through neurons.

Simply speaking, the biological neuron works as follows - it receives signals through its **dendrites** which are either **amplified or inhibited** as they pass through the **axons** to the dendrites of other neurons.

Let's now look at how an Artificial Neural Network is similar to the human brain.

<b>Simplified Human Brain Analogy for Neural Networks</b> <ol style="list-style-type: none"><li>1. Collection of simple devices working together - Neurons</li><li>2. Can have billions of neurons</li><li>3. Requires a lot of training - A lot of data is required</li></ol>	Artificial Neural Networks are a collection of a large number of simple devices called <b>artificial neurons</b> . The network 'learns' to conduct certain tasks, such as recognizing a cat, by <b>training the neurons</b> to 'fire' in a certain way when given a particular input, such as a cat. In other words, the network learns to <b>inhibit or amplify</b> the input signals in order to perform a certain task, such as recognizing a cat, speaking a word, identifying a tree etc.
--	--

Next segment covers the **perceptron**, one of the earliest proposed models for learning simple classification tasks which later become the fundamental building block of artificial neural networks.

## Introduction to Perceptron

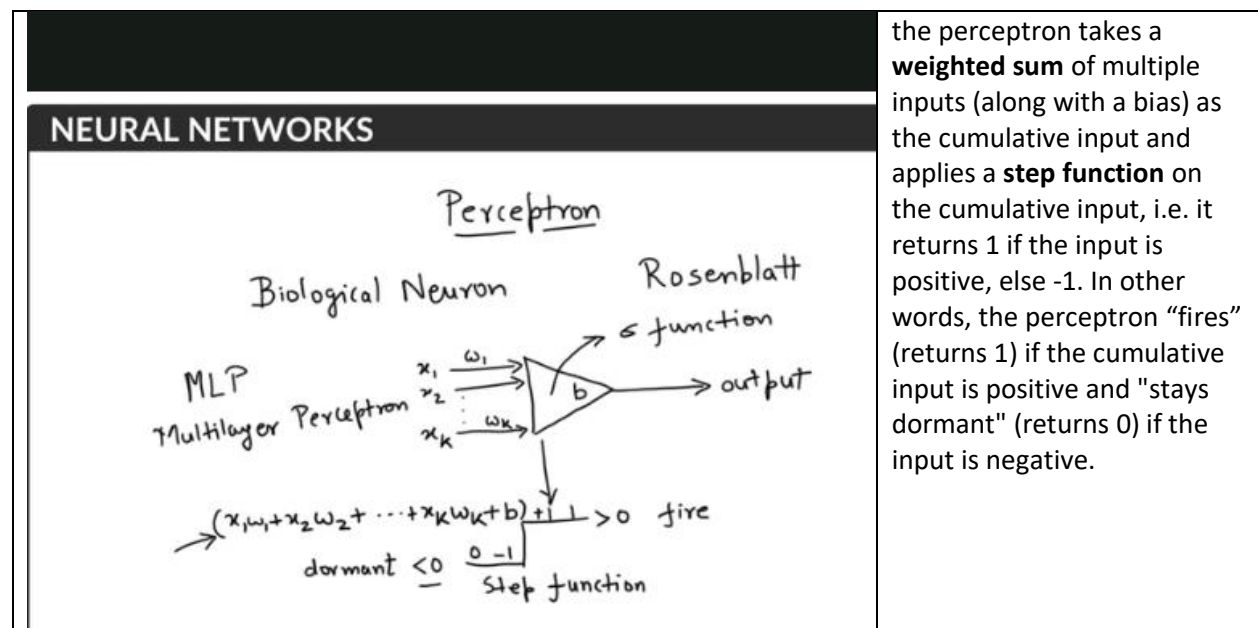
Study a simple device called the **perceptron** - first step towards creating the large neural networks we have today.

Consider a sushi place you plan to visit on the coming Saturday. There are various factors that would affect this decision, such as:

1. How far is it?
2. How costly is it?
3. How many people are accompanying you?

We take a decision based on multiple such factors. Also, each decision factor has a different 'weight' - for example, the distance of the place might be more important than the number of people.

Perceptrons work in a similar way. They take some signals as inputs and perform a set of simple calculations to arrive at a decision.



There are different ways to define the step function. Above is a step function defined in the following way, though one can rather use 1 and -1 as well instead of 1 and 0:

$$y=1 \text{ if } x>0$$

$$y=0 \text{ if } x\leq 0$$

Input to a perceptron is the sum of weights multiplied with their respective inputs and the bias:

$$\text{Cumulative Input} = w_1x_1 + w_2x_2 + \dots + w_kx_k + b$$

Let's start using the vector algebra lingo. Say  $w$  and  $x$  are vectors representing the weights and inputs as follows (by default, a vector is assumed to be a **column vector**):

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}$$

A neat and concise way to represent the weighted sum of  $w$  and  $x$  is using the **dot product** of  $w^T$  and  $x$ . The transpose of  $w$  is  $w^T = [w_1 \ w_2 \ \dots \ w_k]$  - a row vector of size  $1 \times k$ . Taking the dot product of  $w^T$  with  $x$ :

$$w^T \cdot x = [w_1 \ w_2 \ \dots \ w_k] \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = w_1 x_1 + w_2 x_2 + \dots + w_k x_k$$

Upon adding bias to  $w^T \cdot x$ , we have  $w^T \cdot x + b = w_1 x_1 + w_2 x_2 + \dots + w_k x_k + b$ . Hence,

$$\text{Cumulative Input} = w^T \cdot x + b = w_1 x_1 + w_2 x_2 + \dots + w_k x_k + b$$

Upon applying the step function, if this cumulative sum of input is  $> 0$ , the output is 1/yes else 0/no.

Perceptron Output	Cumulative Input
<p>You have the following vectors: <math>w = \begin{bmatrix} 3 \\ 1 \\ 5 \\ 7 \\ 4 \end{bmatrix}</math> and <math>x = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}</math>. The bias value is -2. Following the discussion above, what is the output of the perceptron? Consider the step function as defined above in the discussion.</p> <p>0</p> <p>1</p> <p>Q Feedback: As <math>w^T \cdot x + b &gt; 0</math>, output is 1.</p>	<p>You have the following vectors: <math>w = \begin{bmatrix} 3 \\ 1 \\ 5 \\ 7 \\ 4 \end{bmatrix}</math> and <math>x = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}</math>. The bias value is -2. Following the discussion above, what is the cumulative input to the perceptron?</p> <p>18</p> <p>11</p> <p>9</p> <p>Q Feedback: We have <math>w^T \cdot x + b = 9</math>.</p> <p>16</p>

## Perceptron as a Classifier

How a perceptron can be used for simple learning tasks. To start with, consider a simple **binary classification** task and spend a few minutes thinking about how the perceptron can work as a classifier.

Following lecture covers how the perceptron can act as a classifier.

NEURAL NETWORKS	
<p style="text-align: center;"><u>Perceptron</u></p> <p>Perceptron → classifier</p> <p> <math>(\vec{x}, \vec{y}) \begin{matrix} R \\ B \\ G \end{matrix}</math> <math>\vec{x} \Rightarrow \text{triangle} \rightarrow \begin{matrix} +1 \\ -1 \end{matrix}</math> </p> <p> <math>(\vec{x}_1, \vec{y}_1)</math> <math>\Rightarrow h(\vec{x}) = \vec{y}</math> <math>\text{sign}(\omega^T \cdot \vec{x} + b) = y</math> <math>(\omega_1 x_1 + \omega_2 x_2 + \dots + \omega_n x_n)</math> </p> <p> <math>(\vec{x}_n, \vec{y}_n)</math> </p> <p>binary classification labels, <math>y \in \{+1, -1\}</math></p>	<p>How the perceptron works as a classifier. The weights represent the importance of the corresponding feature for classification. We used a <b>sign function</b>. The 'sign function' is similar to the step function - it outputs +1 when the input is greater than 0 and -1 otherwise. In a binary classification setting, +1 and -1 represent the two classes.</p>

A simple exercise to help better understand how a perceptron works.

Consider the decision of whether to go to the sushi place being taken by a perceptron model. Following factors affect the decision to go/not go: Distance, Cost and Company. These three variables are inputs to the perceptron. Suppose the inputs can be only 0/1 and the weights assigned to each variable add up to 1.

A sample set of weights can be  $\begin{bmatrix} 0.5 & 0.3 & 0.2 \end{bmatrix}$  (Column matrix not row)

For each of the inputs, the rules for deciding 1 and 0 are as follows - these are arbitrary mappings decided to make model simpler:

Factor	1	0
Distance	< 8 km	> = 8 km
Cost	=< Rs 2000 for two	> Rs 2000 for two
Company	> 2 friends	< 2 friends

Assume that the **bias** value is -0.7. The sushi place is 5 km away and 4 friends are ready to go. Also, the cost for 2 is INR 2500.

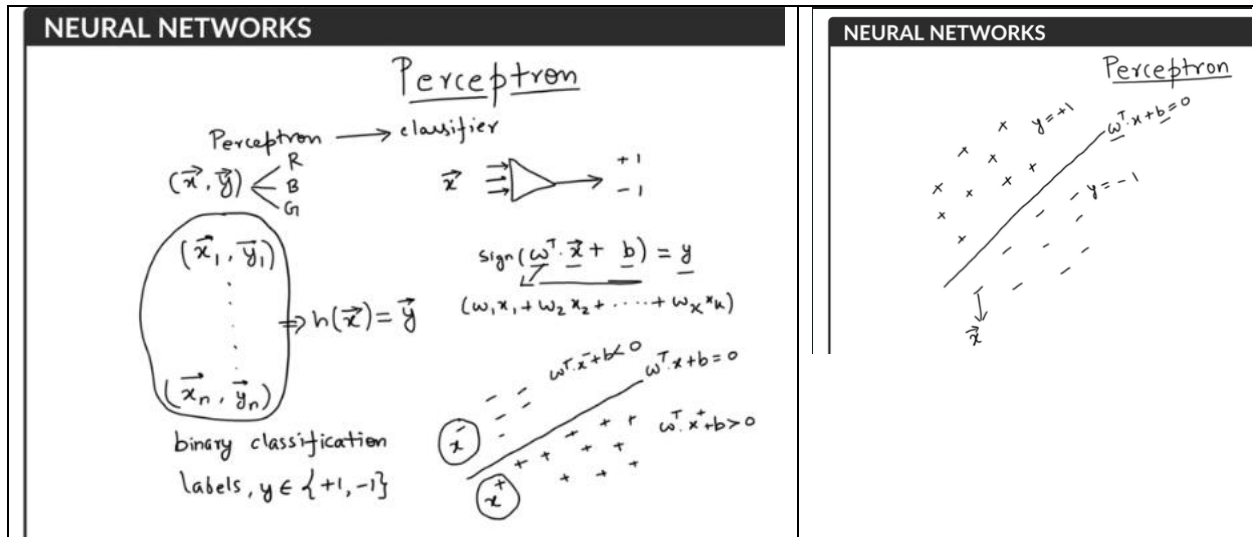
<p><b>Input</b></p> <p>What is the input vector for the above-mentioned sushi place?</p> <p>Input vector = <math>\begin{bmatrix} \text{Distance} \\ \text{Cost} \\ \text{Company} \end{bmatrix}</math></p> <p><math>\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}</math></p> <p><math>\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}</math></p> <p>Q Feedback :  <i>The place is within 8km, hence, 1 for distance. It's cost &gt; 2500. Hence, 0 for the cost. Since 3 friends are accompanying me, the company is 1. Hence, the vector is <math>\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}</math>.</i></p> <p><math>\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}</math></p> <p><math>\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}</math></p>	<p>From this exercise, realise that the weighted sum of inputs, <math>w_1x_1+w_2x_2+w_3x_3</math>, when crosses a <b>threshold</b> (that is 0.7 here), decide to go to the restaurant. else wouldn't go.</p>
--	--

Next segment covers how a perceptron can perform binary classification in detail.



## Binary Classification using Perceptron

Learn how perceptrons can be trained to perform certain tasks. Let's formally define the problem statement and fix some notations.



The perceptron problem statement is defined as follows:

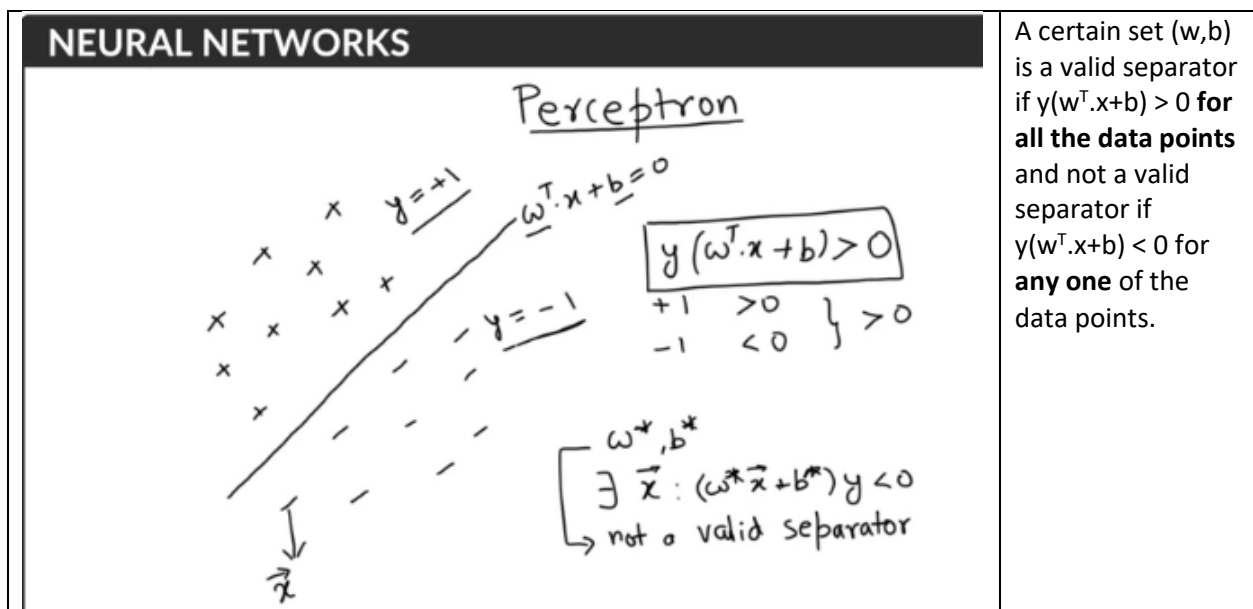
Find  $w$  and  $b$  such that  $w^T \cdot x + b > 0$  for points where  $y = +1$  and  $w^T \cdot x + b < 0$  for points where  $y = -1$ .

Step function used is defined as follows:

$$y = 1 \text{ if } x > 0$$

$$y = -1 \text{ if } x \leq 0$$

Let's represent the above statement in a much concise way.



Let's now solve some questions to concretize these concepts. Following data points with their corresponding ground truth values.

Data points, x	Ground Truth, y
(0,3)	1
(5,9)	1
(-1,-2)	-1

Vector  $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$  can be represented as  $(x_1, x_2, x_3)$ .

#### Valid Separator

Which of the following combination of  $(w, b)$  are valid separators?

$$w = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, b = -2$$

❏ Feedback :  
 $y(w^T \cdot x + b) > 0$  for all data points

$$w = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, b = -5$$

$$w = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, b = -1$$

❏ Feedback :  
 $y(w^T \cdot x + b) > 0$  for all data points

$$w = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, b = 2$$

❏ Feedback :  
 $y(w^T \cdot x + b) > 0$  for all data points

Before we move on, let us first tweak our representation a little to **homogenous coordinates** which will help us in formulating the perceptron solution more neatly.

## NEURAL NETWORKS

$$\vec{x} \rightarrow (\vec{x}, 1)$$

$$\rightarrow \vec{w}^T \vec{x} + b \equiv \vec{w}^T \vec{x}$$

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{pmatrix}$$

### Homogeneous Coordinates

$\forall (x, y) \in \text{Dataset}$

$y \vec{w}^* \cdot \vec{x} > 0$

$\Rightarrow \vec{w}^*$  is a valid separator

Algorithm  
Iterative Proc.

So what homogeneous coordinates mean is as follows:

$x$  earlier represented as this  $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$  transforms to this  $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{bmatrix}$ .

$w$  earlier represented as this  $\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$  transforms to this  $\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{bmatrix}$ .

This new representation does not explicitly state the existence of a bias term though it intrinsically includes it.

### Homogeneous Dimension

Consider an input vector  $x$  which has dimensions (5,1). What is the dimension of  $w$  without transforming to homogenous coordinates?

(1,5)

(5,1)

Feedback :  
The dimension of  $w$  has to be the same as  $x$ .

(6,1)

(1,6)

### Homogeneous Dimension

After converting to homogeneous coordinates, what should be the new dimension of  $x$  and  $w$ ?

(6,1)

Feedback :  
There 1 additional input hence the dimension increases by 1. Hence (6,1).

(1,6)

### Homogeneous Dimension

Suppose that  $x$  is  $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$  and  $w$  is  $\begin{bmatrix} 4 \\ 5 \\ 6 \\ 1 \end{bmatrix}$  and  $b = 3$ . After homogenous transformation, what are new  $x_{new}$  and  $w_{new}$ ?

$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 4 \\ 5 \\ 6 \\ 1 \end{bmatrix}$

$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 4 \\ 5 \\ 6 \\ 1 \end{bmatrix}$

$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 4 \\ 5 \\ 6 \\ 3 \end{bmatrix}$

Feedback :  
1 is added in  $x$  and 3 that is bias is added in the  $w$  vector.

$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}$  and  $\begin{bmatrix} 4 \\ 5 \\ 6 \\ 3 \end{bmatrix}$

<p><b>Homogeneous Dimension</b></p> <p>Suppose that <math>x</math> is <math>\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}</math> and <math>w</math> is <math>\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}</math> and <math>b = 3</math>. What is <math>(w^T \cdot x + b)</math>?</p> <div> <div>35</div> <div> Q Feedback :  <math>(w^T \cdot x + b) = 35</math> </div> </div> <div>32</div>	<p><b>Homogeneous Dimension</b></p> <p>You got the <math>x_{new}</math> and <math>w_{new}</math> as <math>\begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}</math> and <math>\begin{bmatrix} 4 \\ 5 \\ 6 \\ 3 \end{bmatrix}</math>.</p> <p>What is <math>w_{new}^T \cdot x_{new}</math>?</p> <div> <div>32</div> <div> Q Feedback :  <math>w_{new}^T \cdot x_{new} = 35</math> </div> </div> <div>35</div>
---	---

Homogeneous coordinates to represent the perceptron more concisely help us in illustrating some of the wonderful tasks a set of perceptrons can do.

## Perceptrons - Training

Let's look at the iterative solution suggested by Rosenblatt for **training the perceptron**.

### NEURAL NETWORKS

Homogeneous Coordinates

$\vec{x} \xrightarrow{d+1} (\vec{x}, 1)$   
 $\rightarrow \vec{w}^T \vec{x} + b \equiv \vec{w}^T \vec{x}$

$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{pmatrix}$

$\forall (x, y) \in \text{Dataset}$   
 $y \vec{w}^* \cdot \vec{x} > 0$   
 $\Rightarrow \vec{w}^*$  is a valid separator

**Algorithm**  
 Iterative Proc.  
 $\vec{w}_0 \leftarrow$  arbitrary initialization  
 $\vec{w}_{t+1} \leftarrow \vec{w}_t + y_{i_t} \cdot \vec{x}_{i_t}$   
 (Error term  $y_{i_t} \cdot \vec{x}_{i_t}$  is labeled **error**)  
 misclassified datapoint

### NETWORKS

$\vec{w}_{t+1} \leftarrow \vec{w}_t + y_{i_t} \cdot \vec{x}_{i_t}$   
 (Error term  $y_{i_t} \cdot \vec{x}_{i_t}$  is labeled **error**)

Rosenblatt suggested an elegant iterative solution to train the perceptron (i.e. to learn the weights):

$$\vec{w}_{t+1} \leftarrow \vec{w}_t + y_{i_t} \cdot \vec{x}_{i_t}$$

(The term  $y_{i_t} \cdot \vec{x}_{i_t}$  is labeled **error**)

### Preceptron Algo

where  $y_{it}, x_{it}$  is the error term.  $x_{it}$  in this iterative procedure is a **misclassified data point** and  $y_{it}$  is the corresponding true label. Dot in  $y_{it}, x_{it}$  is not a dot product.

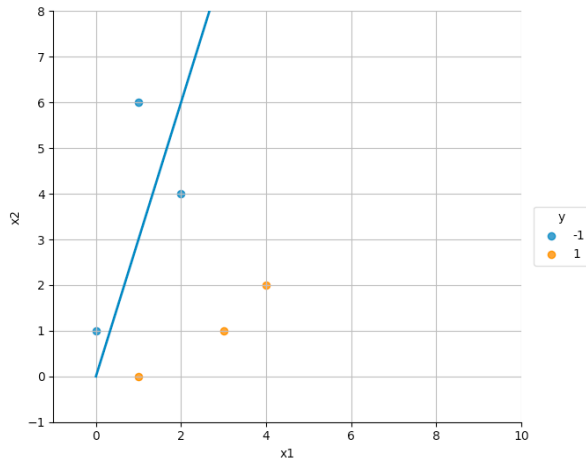
Consider the following figure with 6 data points and the separator. These are represented by numbers on a scale. The blue points belong to class '-1' and the orange points belong to the class '+1'.

The coordinates of the points and the labels are given in the following table:

Data Points	x1	x2	True Label(y)	Homogeneous coordinates
0	1	0	1	(1,0,1)
1	3	1	1	(3,1,1)
2	4	2	1	(4,2,1)
3	0	1	-1	(0,1,1)
4	1	6	-1	(1,6,1)
5	2	4	-1	(2,4,1)

Last column has the homogeneous coordinates of the data points.

The initial classifier is  $(3, -1, 0)$  which when expressed algebraically is  $3x_1 - 1x_2 = 0$ .



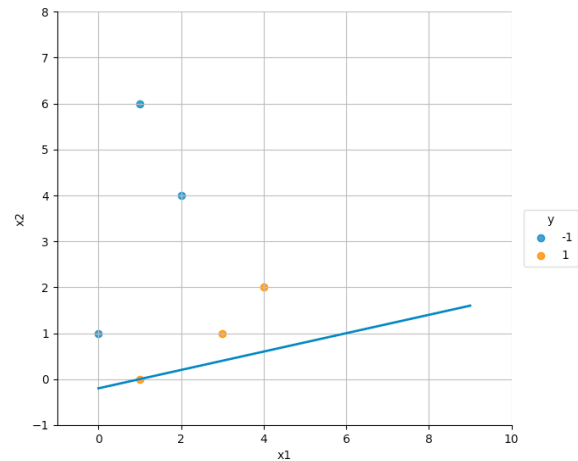
### Setting Algo

For the first iteration, the misclassified data point is '5':  $(2, 4, 1)$ .

In the formula  $w_{t+1} = w_t + y_{it} \cdot x_{it}$ ,  $x_{it}$  is  $(2, 4, 1)$  and  $y_{it}$  is the true label '-1'.

We get  $w_1 = \begin{bmatrix} 3 \\ -1 \\ 0 \end{bmatrix} + (-1) * \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -5 \\ -1 \end{bmatrix}$

$w_1 = (1, -5, -1)$  which is  $1 * x_1 - 5 * x_2 = -1$  shown in the figure below.



### Iteration 2

We performed the 1st iteration to get  $w_1$ . Line moves in the right direction, though it misclassifies two orange points now (and passes through one).

#### Misclassified Points

Which of the following data points are misclassified after the 1<sup>st</sup> iteration?

☐ 0

☒ 1

Feedback :  
You are correct.

☒ 2

Feedback :  
You are correct.

#### Misclassified Points

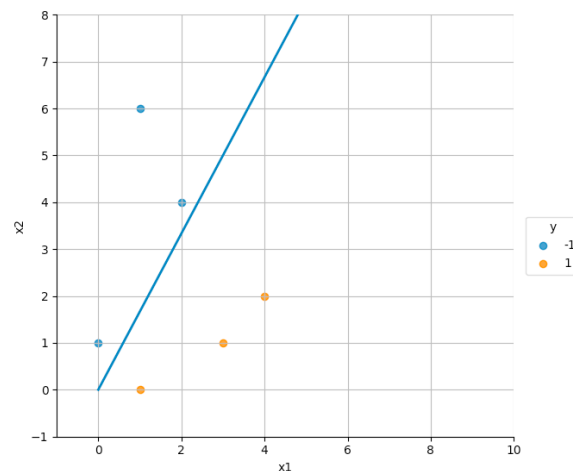
What is the true label  $y_{it}$  for the misclassified points?

☒ +1

Feedback :  
We can see from the table that the misclassified points have true label +1.

☐ -1

The  $w_2$  we get is  $\begin{bmatrix} 5 \\ -3 \\ 0 \end{bmatrix}$ , algebraically expressed as  $5x_1 - 3x_2 = 0$ . It classifies all the data points correctly.



Perceptron Algorithm	W2
<p>Let us consider the data point '2': <math>\begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}</math> for the next update to get <math>w_2</math>. What is <math>x_{it}, y_{it}</math> for the data point '2'?</p> <hr/> <p><input type="radio"/> <math>\begin{bmatrix} -4 \\ -2 \\ -1 \end{bmatrix}</math></p> <p><input checked="" type="radio"/> <math>\begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}</math> ✓</p> <p>Feedback :             As <math>y_{it}</math> is true label which is +1. Hence the answer is <math>\begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}</math>.</p>	

This is a simple way to understand the intuition behind the algorithm. Go through the mathematics of the proof in the additional reading section.

A perceptron performs binary classification, but wouldn't it be amazing if these simple devices could do something more complex? Next segment covers how a group of perceptrons can do multiclass classification.

#### Additional Readings:

- Proof of learning algorithm of the perceptron [here](#).

## Multiclass Classification using Perceptrons

We saw how a perceptron performs binary classification. But more tasks can be done by a perceptron (or a collection of them). They can do much more complex things, such as **multiclass classification**.

**WORKS**

$$f(\vec{x}) = \vec{y}$$

Input  $\vec{x}$  →  $f$  → Output  $\vec{y}$   
Universal  $f^n$  Approximators

**AL NETWORKS** Perceptron Up

$w_1^T x \geq 0$   
 $w_2^T x \geq 0$   
 $w_3^T x \geq 0$   
 $w_4^T x \geq 0$   
AND → Sum  $\geq 4$

**OR gate**

Think about how you'd design a perceptron which works as an OR gate. Consider the setting shown in the lecture - the perceptron gets 4 inputs (each of which can either be -1 or 1) and that it has weights  $w_1, w_2, w_3, w_4$  and bias  $b$ . Consider that all the weights are 1 i.e.  $w_1 = w_2 = w_3 = w_4 = 1$ . The perceptron works as an OR gate when:

Note that OR gate outputs 1 when at least one of the four inputs is 1.

- $b = 0$
- $b = -4$
- $b = 4$

Feedback :

The cumulative input to the OR perceptron =  $w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$ . Let's take the worst case for the output to be 1 when either of the 4 inputs is 1 for it to function as an OR. Hence,  $w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 = -2$  as all weights are 1. Hence,  $b$  can be any value  $> 2$ . But at the same time, it should not give an output 1 when all 4 inputs are -1, where  $w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 = -4$ . Hence,  $b < 4$ . Hence,  $b$  can be 3 and 4.

- $b = 1$

**NETWORKS** P

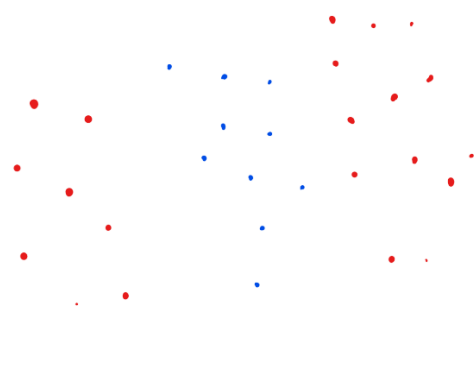
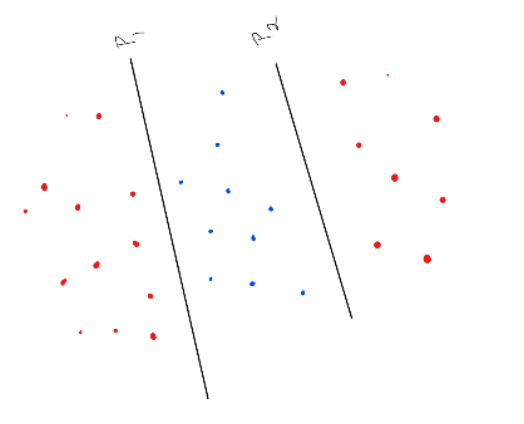
Input  $\vec{x}$  →  $P_1, P_2, P_3, P_4$  → OR → output  
Sum  $> -4$   
sets of perceptrons

**NETWORKS**



A network of perceptrons can act as a **universal function approximator**. A single layer of perceptron in combination with an AND gate leads to an enclosure in a polygon, and multiple such AND outputs using an OR gate lead to an enclosure in multiple polygons. In the most extreme case, this can be extended to finding a polygon for every single data point.

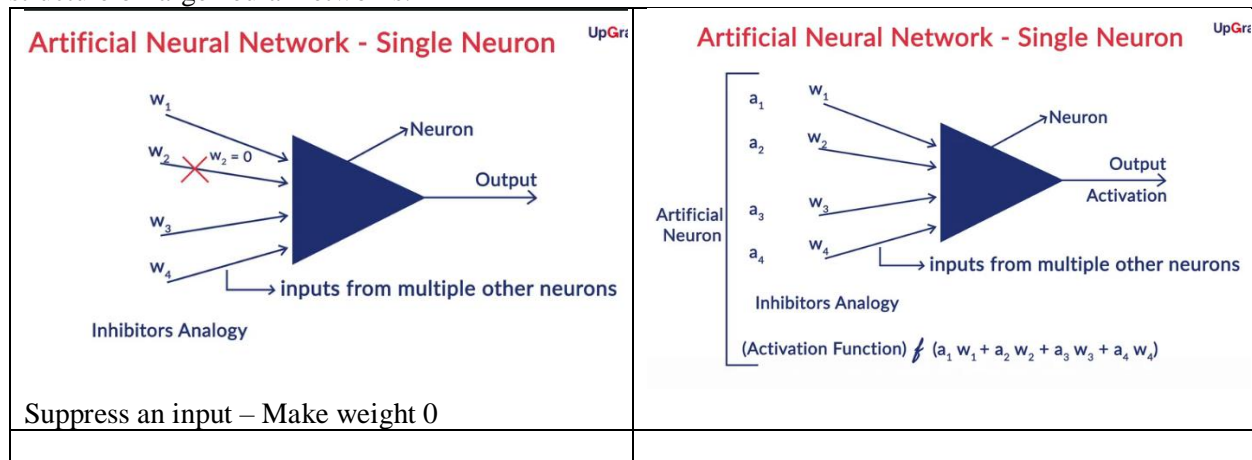
In the following figure, there are two classes shown in two colours.

 <p>2 classes</p>	<p>A single perceptron is a binary classifier and that it can be defined in any of the following shown ways. You have the freedom to decide which one to use to answer the following questions.</p>  <p>Hint</p>
<p><b>Number of Perceptrons</b></p> <p>What is the minimum number of perceptrons required to correctly classify the points?</p> <p><input type="radio"/> 1</p> <p><input checked="" type="radio"/> 2</p> <p>Feedback : You need just 2 perceptrons as shown in the above image.</p> <p><input type="radio"/> 4</p> <p><input type="radio"/> More than 4</p>	<p><b>AND/OR</b></p> <p>We will use an OR gate for the two perceptrons we choose. True or False?</p>  <p><input type="radio"/> True</p> <p><input checked="" type="radio"/> False</p> <p>Feedback : We'll use the AND of the two perceptrons as shown in the image.</p>

## Working of a Neuron

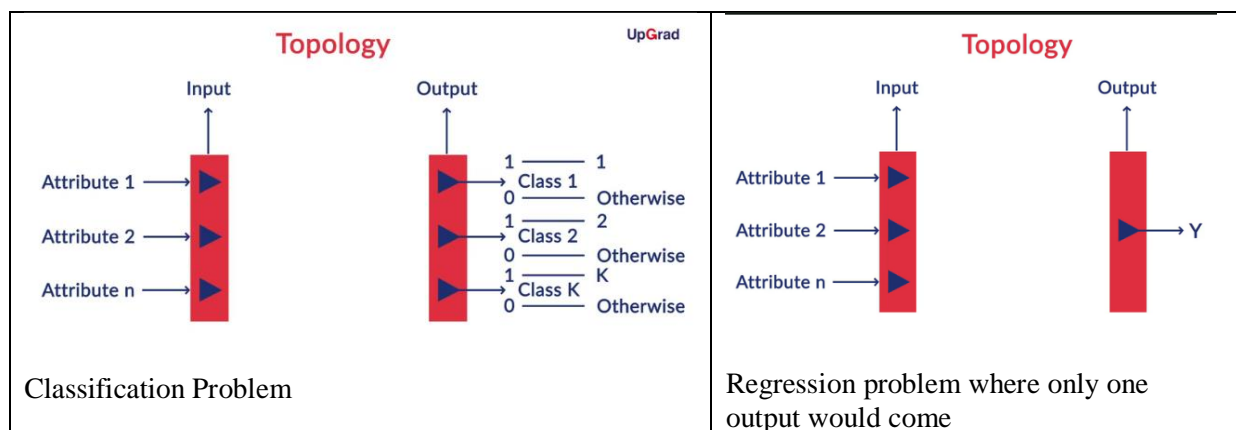
Understanding of perceptrons equips to study the design of **Artificial Neural Networks (ANNs)**.

Neural networks are a collection of artificial neurons arranged in a specific structure. This segment covers how an artificial neuron works i.e. how it converts inputs into outputs. It also covers the **topology** or the structure of large neural networks.



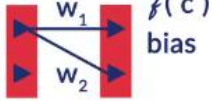
A neuron is very similar to a perceptron, the only difference being that there is an **activation function** applied to the weighted sum of inputs. In perceptrons, the activation function is the **step function**, though, in artificial neural networks, it can be any non-linear function.

Let's now look at how large neural networks are designed using multiple individual neurons.



Neurons in a neural network are arranged in **layers**. The first and the last layer are called the **input and output** layers. Input layers have as many neurons as the number of attributes in the data set and the output layer has as many neurons as the number of classes of the target variable (for a classification problem). For a regression problem, the number of neurons in the output layer would be 1 (a numeric variable).

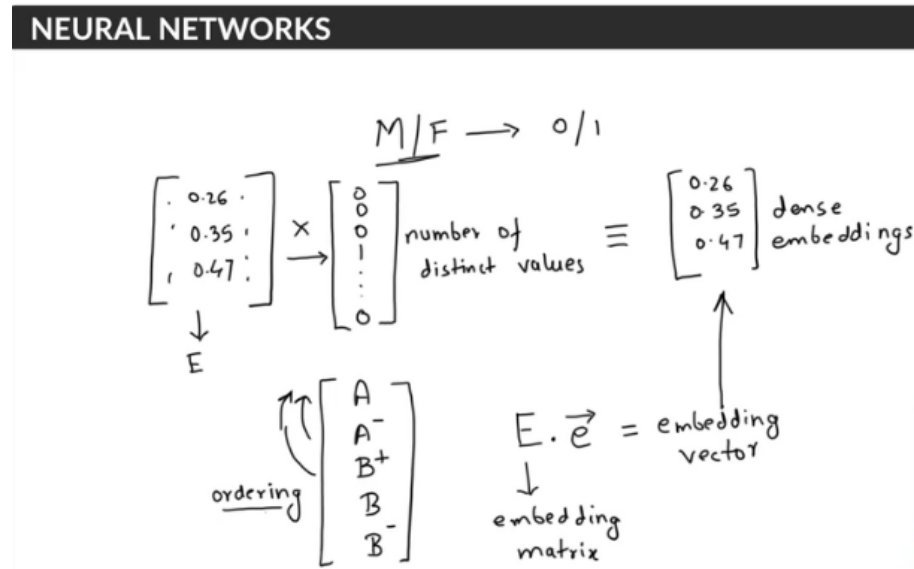
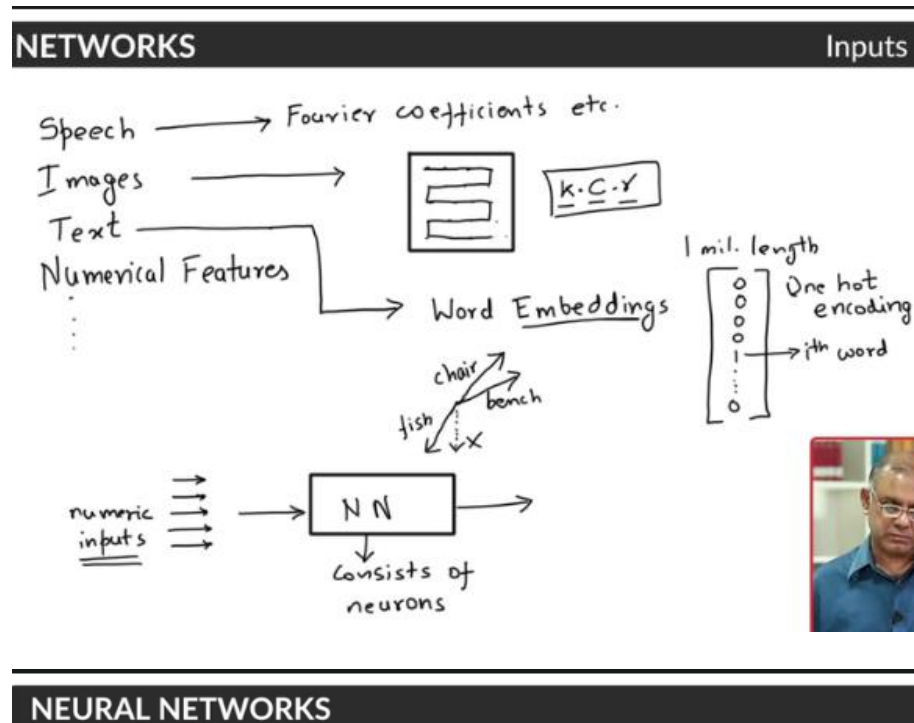
What it means to specify a neural network completely, i.e. what all we need to specify to completely describe a neural network?

<p style="text-align: right;">UpGrad</p> <h2 style="color: red;">Specifications of Neural Network</h2> <ol style="list-style-type: none"> <li>1. Structure / Topology</li> <li>2. Graph — Nodes - Neurons Edges - Interconnections</li> <li>3. Specify the Input layer</li> <li>4. Specify the Output layer</li> <li>5. Specify the Weights</li> <li>6. </li> <li>7. Specify the activation function</li> <li>8. Specify the bias</li> </ol>	<h2 style="color: red;">Cumulative Input</h2> $f \left[ \sum_{i=1}^k w_i a_i + b_i \right]_{\text{Out}}$ <h2 style="color: red;">Specifications for Neural Networks</h2> <ol style="list-style-type: none"> <li>1. Network Topology</li> <li>2. Input Layer</li> <li>3. Output Layer</li> <li>4. Weights</li> <li>5. Activation Functions</li> <li>6. Biases</li> </ol>
---	---

Next segments will cover "how to decide the number of neurons in the input layer", "how are weights and biases specified", etc.

## Inputs and Outputs of a Neural Network I

Number of neurons in the input layer is determined by the input given to the network and that the number of neurons in the output layer is equal to the number of classes (for a classification task) and equal to one (for regression tasks). Let's look at some example use cases to understand the input and outputs of ANNs better.

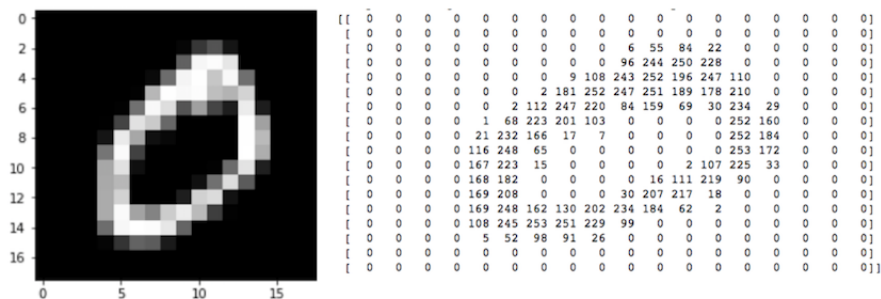


Inputs can only be numeric. For different types of input data, use different ways to convert the inputs to a numeric form.

In case of **text data**, either use a **one-hot vector** or **word embeddings** corresponding to a certain word. For example, if the vocabulary size is  $|V|$ , then represent the word  $w_n$  as a one-hot vector of size  $|V|$  with a

'1' at the nth element while all other elements being zero. The problem with one-hot representation is that usually the vocabulary size  $|V|$  is huge, in tens of thousands at least, and hence it is often better to use word embeddings which are a lower dimensional representation of each word.

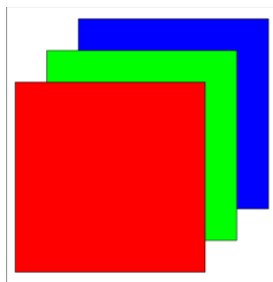
Feeding **images** (or videos) is straightforward since images are naturally represented as **arrays of numbers**. These numbers are the raw **pixels** of the image. Pixel is short for picture element. In images, pixels are arranged in rows and columns (an array of pixel elements). The figure below shows an image of a handwritten 'zero' in the MNIST dataset (black and white) and its corresponding representation in Numpy as an array of numbers. The pixel values are high where the **intensity** is high, i.e. the colour is white-ish, while they are low in the black regions.



**MNIST Zero Numpy Array**

In a neural network, each **pixel** of the input image is a **feature**. For example, the image above is an 18 x 18 array. Hence, it will be fed as a **vector of size 324** to the network.

Image above is a **black and white** image (also called **greyscale image**), and thus, each pixel has only one 'channel'. If it were a **colour image** (called an RGB image - Red, Green, Blue), each pixel would have **three channels** - one each for red, blue and green as shown below. Hence, the number of neurons in the input layer would be  $18 \times 18 \times 3 = 972$ . In detail in the next module on Convolution Neural Networks.



**RGB**

<p><b>Input and Output Layers of a Neural Network</b></p> <p>In a classification problem with 12 attributes and 3 class labels, the number of neurons in input and output layers will be:</p> <p><input type="radio"/> 12, 2</p> <p><input checked="" type="radio"/> 12, 3</p> <p>Q Feedback : The input layer has 12 neurons corresponding to 12 attributes and the output layer has 3 neurons corresponding to probability of class labels 1, 2 and 3.</p> <p><input type="radio"/> 2, 12</p> <p><input type="radio"/> 1, 12</p>	<p><b>Input and Output Layers of a Neural network</b></p> <p>Neural networks are quite popular in image recognition problems. The task is to classify a given black and white image (say a Google image) into categories like nature, animal, and sports. An image is just a collection of pixels. In a 720 x 1080 image, for example, there are 720 pixels along the vertical of the image and 1080 along the horizontal. Each pixel acts as an attribute and contains a 'value' which may represent the color, shade etc. at that point on the image.</p> <p>To classify an image into the three categories mentioned above, the number of neurons in the input and output layers are respectively:</p> <p><input type="radio"/> 720, 1080</p> <p><input type="radio"/> 720 X 1080, 1</p> <p><input checked="" type="radio"/> 720 X 1080, 3</p> <p>Q Feedback : The 720 x 1080 pixels each act as an attribute. The three output neurons contain the probability of an image being from nature, animal or sports.</p> <p><input type="radio"/> 1080, 720</p>
<p><b>Inputs to the Network</b></p> <p>What would be the number of neurons in the input layer if the above 720 x 1080 image was RGB instead of black and white?</p> <p><input type="radio"/> 720 x 1080</p> <p><input checked="" type="radio"/> 720 x 1080 x3</p> <p>Q Feedback : 720 x 1080 x3</p>	

After looking into how to feed input vectors to neural networks, let's see how the **output layers** are specified.

## NEURAL NETWORKS

collection as a vector

softmax output multiclass logistic f

$$p_0 = \frac{e^{w_0 \cdot x'}}{e^{w_0 \cdot x'} + e^{w_1 \cdot x'} + e^{w_2 \cdot x'}}$$

$$0 \leq p_0, p_1, p_2 \leq 1$$

A softmax output is a **multiclass logistic function** commonly used to compute the 'probability' of an input belonging to one of the multiple classes. It is defined as follows:

$$p_i = \frac{e^{w_i \cdot x'}}{\sum_{t=0}^{c-1} e^{w_t \cdot x'}}$$

where c is the number of neurons in the output layer.

If the output layer has 3 neurons and all of them have the same input  $x'$  (coming from the previous layers in the network), then the probability of the input belonging to class 0 is:

$$p_0 = \frac{e^{w_0 \cdot x'}}{e^{w_0 \cdot x'} + e^{w_1 \cdot x'} + e^{w_2 \cdot x'}}$$

Although, it seems here that the  $x'$  and all the  $w_i$ s here are scalars but that is not the case in a neural network. In the neural network, the input  $x'$  is a large vector and the  $w_i$ s are rows of a weight matrix which is present between the layers. More about the weight matrices and the representation in the next session.

The above expression means that the sum  $p_0 + p_1 + p_2 = 1$  and that  $p_0, p_1, p_2 \in (0,1)$ .

<p><b>Softmax expression</b></p> <p>Looking at the above equation, what will be the expression for <math>p_1</math>?</p> <div style="margin-top: 10px;"> <input type="radio"/> <math>\frac{e^{w_0 \cdot x'}}{e^{w_0 \cdot x'} + e^{w_1 \cdot x'} + e^{w_2 \cdot x'}}</math> </div> <div style="margin-top: 10px; background-color: #e0ffe0; padding: 5px;"> <input checked="" type="radio"/> <math>\frac{e^{w_1 \cdot x'}}{e^{w_0 \cdot x'} + e^{w_1 \cdot x'} + e^{w_2 \cdot x'}}</math> <p><b>Q Feedback :</b> It is corresponding to <math>i = 1</math> in <math>\frac{e^{w_i \cdot x'}}{\sum_{t=0}^{c-1} e^{w_t \cdot x'}}</math> where <math>c = 3</math>.</p> </div> <div style="margin-top: 10px;"> <input type="radio"/> <math>\frac{e^{w_2 \cdot x'}}{e^{w_0 \cdot x'} + e^{w_1 \cdot x'} + e^{w_2 \cdot x'}}</math> </div>	<p><b>Range of Softmax</b></p> <p>In the above softmax output layer, if <math>p_0 = 0.5</math>, what is the range of <math>p_1</math>?</p> <div style="margin-top: 10px;"> <input type="radio"/> 0 to 1         </div> <div style="margin-top: 10px; background-color: #e0ffe0; padding: 5px;"> <input checked="" type="radio"/> 0 to 0.5  <b>Q Feedback :</b>            Maximum value <math>p_1</math> can have is <math>1 - 0.5 = 0.5</math>. Minimum value is 0 when <math>p_2 = 0.5</math>.         </div> <div style="margin-top: 10px;"> <input type="radio"/> 0 to 0.25         </div> <div style="margin-top: 10px;"> <input type="radio"/> 0.5 to 1         </div>
--	--

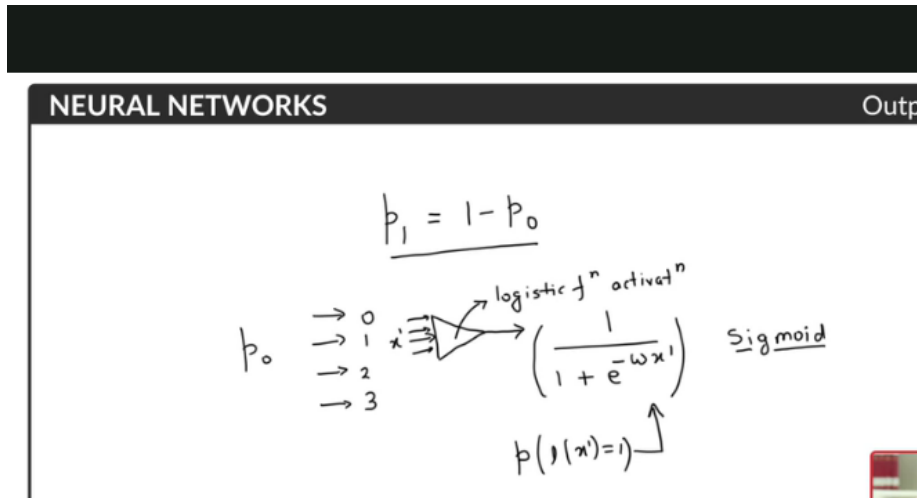
The softmax function stated above is a general case for multiclass classification. Next is how the softmax function translates to a **sigmoid function** in the special case of **binary classification** in the next segment.

### Additional Reading:

- Go through [this book](#) to understand more about recognizing handwritten digits

## Inputs and Outputs of a Neural Network II

We can define the inputs and outputs of a neural network. This segment covers how we can define the inputs and outputs for the famous MNIST dataset for the multiclass classification. First, let's see how the softmax function translates to a **sigmoid function** in the special case of **binary classification**:



For **sigmoid output**, there is only **one neuron** in the output layer since if there are two classes with probabilities  $p_0$  and  $p_1$ ,  $p_0 + p_1 = 1$ . Hence, we need to compute the value of only one of  $p_0$  or  $p_1$ . Sigmoid function is just a special case of the softmax function (since binary classification is a special case of multiclass classification).

We can derive sigmoid function from softmax function. Assume that the softmax function has two neurons with the following outputs:

$$p_0 = \frac{e^{w_0 \cdot x'}}{e^{w_0 \cdot x'} + e^{w_1 \cdot x'}} \text{ and } p_1 = \frac{e^{w_1 \cdot x'}}{e^{w_0 \cdot x'} + e^{w_1 \cdot x'}}$$

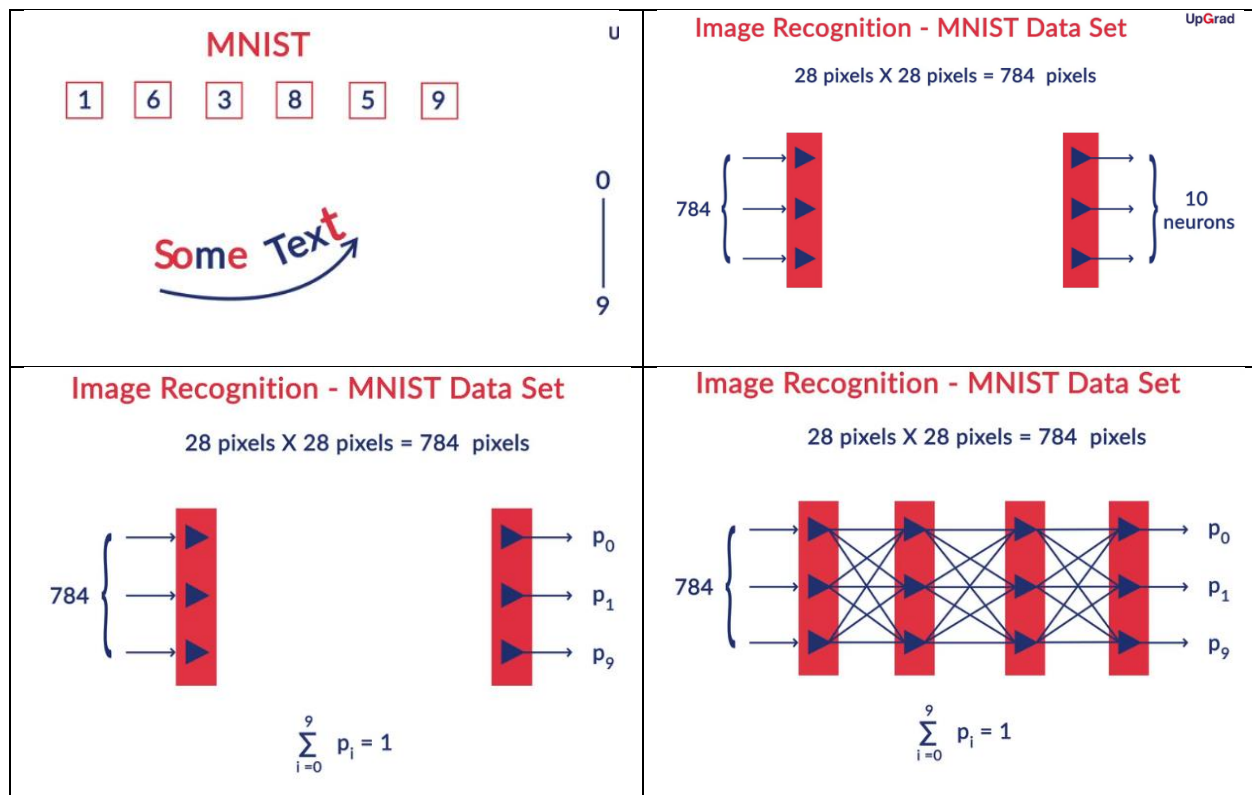
Dividing by the numerator, we can rewrite  $p_1$  as:

$$p_1 = \frac{1}{1 + \frac{e^{w_0 \cdot x'}}{e^{w_1 \cdot x'}}} = \frac{1}{1 + e^{(w_0 - w_1) \cdot x'}}$$

And if we replace  $w_1 - w_0 = \text{some } w$ , we get the sigmoid function.







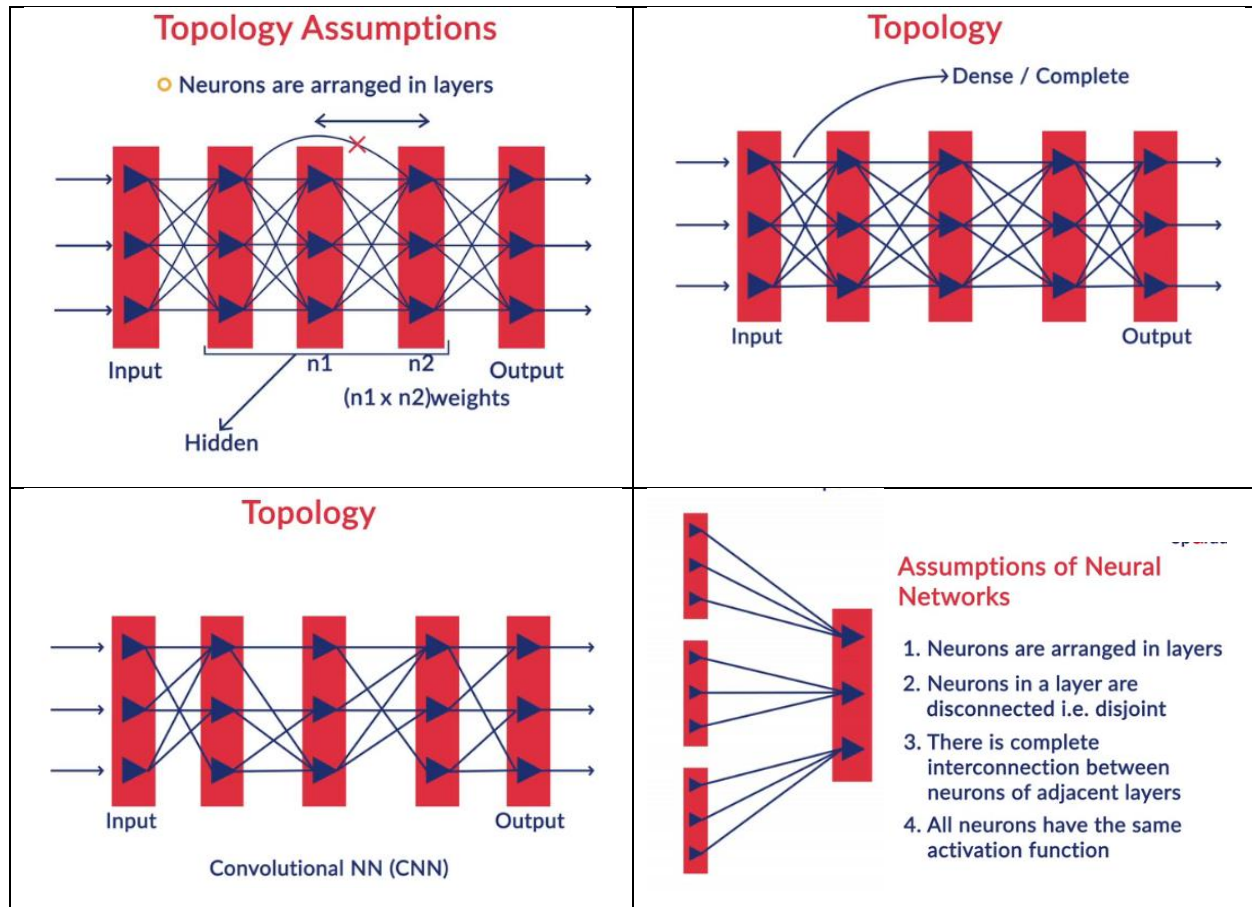
There are various problems while trying to recognise handwritten text using an algorithm such as:

- Noise in the image
- The orientation of the text
- Non-uniformity in the spacing of text
- Non-uniformity in handwriting.

The MNIST dataset takes care of some of these problems since the digits are written in a box. Now the only problem the network needs to take care of the non-uniformity in handwriting. Since the images in the MNIST dataset are 28 X 28 pixels, the input layer has 784 neurons (each neuron takes 1 pixel as input) and the output layer has 10 neurons each giving the probability of the input image belonging to any of the 10 classes. The image is classified to the class represented by the neuron with the highest probability.

## Assumptions made to Simplify Neural Networks

Since large neural networks can potentially have extremely complex structures, certain assumptions are made to simplify the way information flows in them. Let's see some of the most common assumptions.



Commonly used neural network architectures make the following simplifying assumptions:

1. Neurons are **arranged in layers** and the layers are arranged **sequentially**.
2. Neurons **within the same layer do not interact** with each other.
3. All the inputs enter the network through the **input layer** and all the outputs go out of the network through the **output layer**.
4. Neurons in consecutive layers are **densely connected**, i.e. all neurons in layer  $l$  are connected to all neurons in layer  $l+1$ .
5. **Every interconnection** in the neural network has a **weight** associated with it, and **every neuron has a bias** associated with it.
6. All neurons in a layer use the **same activation function**.

After basic assumptions in the architecture of ANNs, next is how neural networks are trained and used to make predictions. Next segment covers the **hyperparameters and parameters** of neural networks.

## Parameters and Hyperparameters of Neural Networks

Neural networks require rigorous training. But what does it mean to train neural networks? What are the **parameters** which the network learns during training, and what are the **hyperparameters** which you (as the network designer) need to specify beforehand?

Models such as linear regression, logistic regression, SVMs etc. are trained on their coefficients, i.e. the training task is to find the optimal values of the coefficients to minimise some cost function.

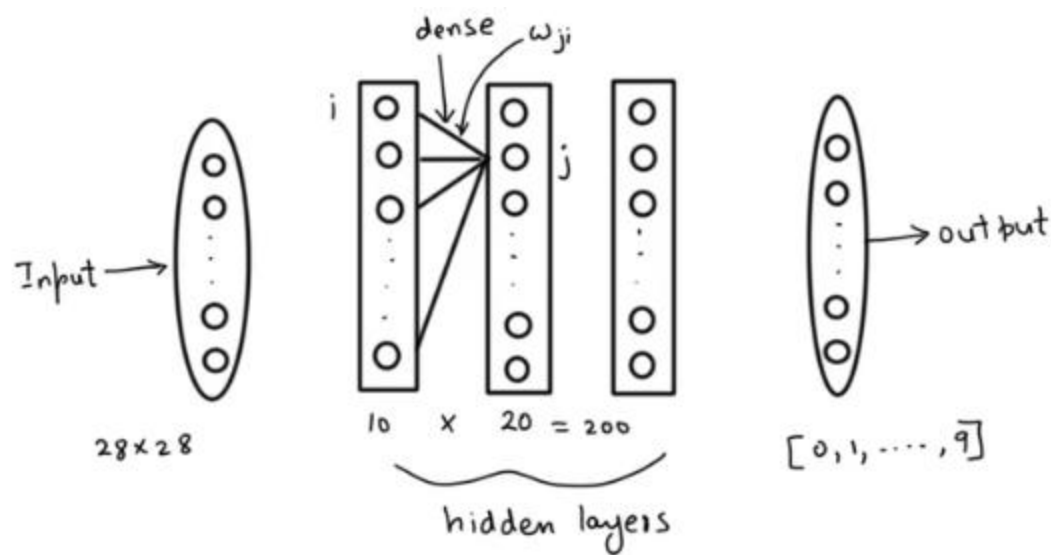
Neural networks are no different - they are **trained on weights and biases**.

In this segment, cover the parameters which are learnt during neural network training. Also, a broad understanding of how the learning algorithm is trained.

<b>Neural Networks – Learning Algorithm</b>  <ol style="list-style-type: none"><li>1. Fits multiple neural networks</li><li>2. Selects the best network for predictions</li></ol>	<b>Working with Neural Networks</b>  <ol style="list-style-type: none"><li>1. Known topology - Network structure is fixed</li><li>2. Activation Functions are fixed</li><li>3. Input and Output layers are fixed</li></ol>
<b>Training a Neural Network</b>  <ol style="list-style-type: none"><li>1. Trained on weights and biases</li><li>2. Weights and Biases - Influences the model produced</li></ol>	<b>Optimal Neural Networks</b>  <ol style="list-style-type: none"><li>1. Fixed – Topology, Activation Functions, Inputs and Outputs</li><li>2. Tune weights and biases to obtain best model</li></ol>

During training, the neural network learning algorithm fits various models to the training data and selects the best model for prediction. The learning algorithm is trained with a **fixed set of hyperparameters** - the network structure (number of layers, number of neurons in the input, hidden and output layers etc.). It is trained on the **weights and the biases**, which are the **parameters of the network**.

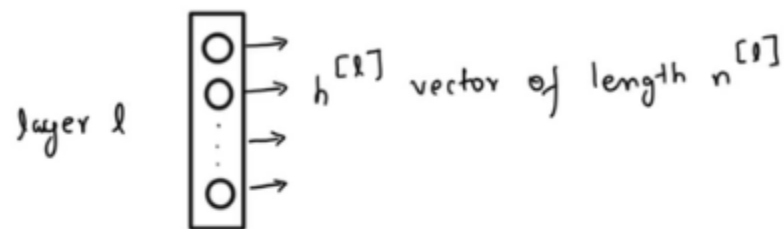
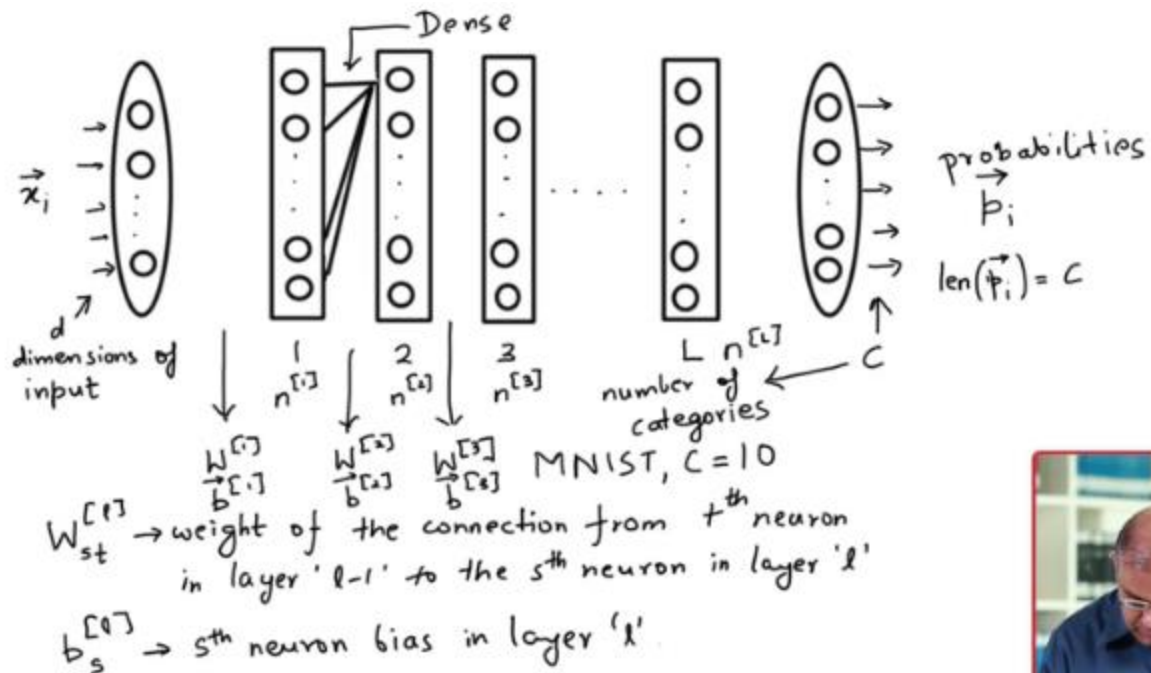
Let's have a closer look at the **hyperparameters** - the number of neurons in the input and the output layers, activation functions, the number of layers etc. Also fix some notations to use throughout the upcoming lectures.



Define the symbols using the MNIST dataset as an example.

$W^{[l]}$  → weights for connections between layer 'l' & 'l-1'  
 $\vec{x}_i$  → input vector,  $i$  → index (ith input datapoint)  
 $\vec{y}_i$  → label for ith datapoint (ground truth)  
 $\vec{p}_i$  → network output for input  $\vec{x}_i$   
 $\vec{b}^{[l]}$  → bias vector for a layer 'l'

$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$  → one hot encoding 7

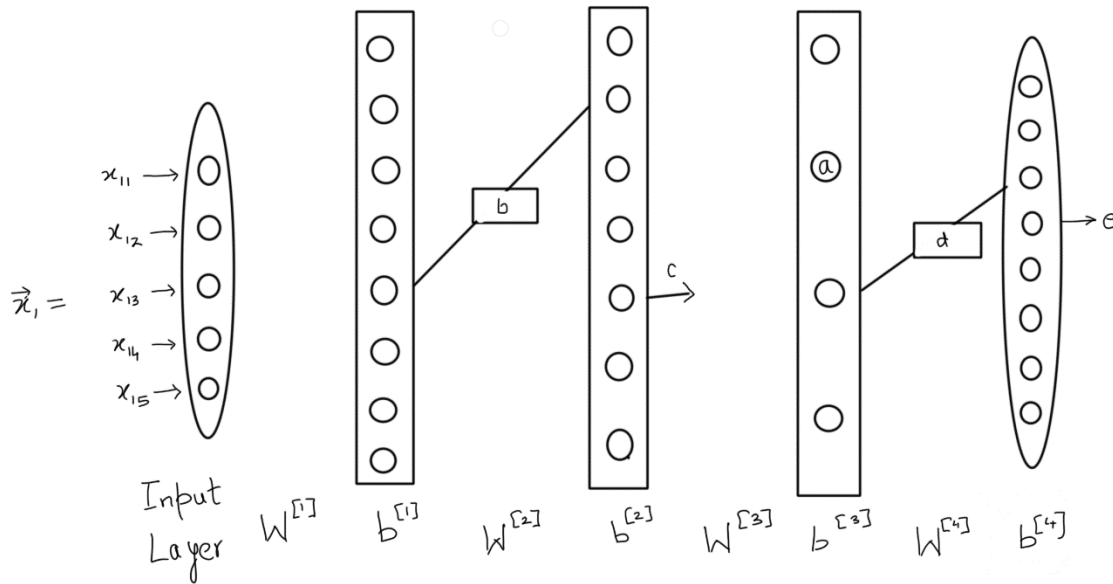


$|A|$  → matrix  
 superscript → layer      subscript → values of individual neuron  
 $x$  → input  
 $y$  → output  
 $b$  → bias  
 $h$  → hidden layer output  
 $\vec{x}, \vec{y}, \vec{b}, \vec{h}, \vec{p}$  → vectors

The notations are

1. **W** is for weight matrix
2. **b** shall stand for the bias
3. **x** stands for input

4.  $\mathbf{y}$  is the ground truth label
5.  $\mathbf{p}$  is the probability vector of the predicted output
6.  $\mathbf{h}$  is the output of the hidden layers
7. superscript stands for layer number
8. subscript stands for the index of the individual neuron



## Notation Check

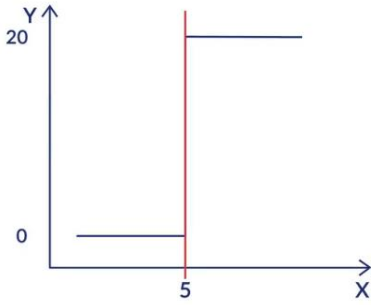
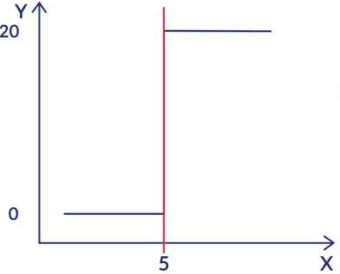
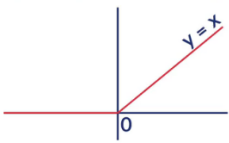
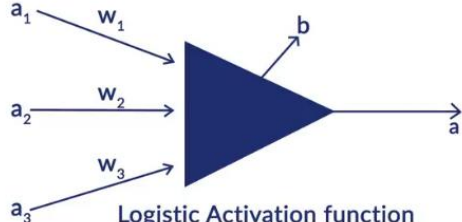
<p><b>Notations</b></p> <p>How will you represent the probability output coming out of the last layer represented as <math>e</math>?</p> <p><input checked="" type="radio"/> <math>p_{41}</math></p> <p>Q Feedback : We know that the input vector is <math>x_1</math>. Hence, the corresponding output vector will be <math>p_1</math>. Notice that we represented the input into the 4th neuron as <math>x_{14}</math>.</p> <p><input type="radio"/> <math>p_{14}</math></p> <p>Q Feedback : We know that the input vector is <math>x_1</math>. Hence, the corresponding output vector will be <math>p_1</math>. Like we represented the input into the 4th neuron as <math>x_{14}</math>, hence the element of <math>p_1</math> for the 4th neuron will be <math>p_{14}</math>.</p> <p><input type="radio"/> <math>p_1</math></p> <p><input type="radio"/> <math>p_4</math></p>	<p><b>Notations</b></p> <p>How will you represent the element of the weight matrix between layer 1 and 2 represented as <math>b</math> in the figure (do not confuse this with the bias)?</p> <p><input checked="" type="radio"/> <math>W_{25}^2</math> ✓</p> <p>Q Feedback : The elements are of the matrix <math>W^2</math>. It is connecting to 2nd neuron in layer 2 from 5th neuron in the layer 1. Hence, <math>W_{25}^2</math>.</p> <p><input type="radio"/> <math>W_{25}^1</math></p> <p><input type="radio"/> <math>W_{52}^1</math></p> <p><input type="radio"/> <math>W_{52}^2</math></p>
--	---



<p><b>Notations</b></p> <p>On similar lines, how will you represent the element of the weight matrix represented as <math>d</math>?</p> <hr/> <p><input type="radio"/> <math>W_{33}^3</math></p> <p><input checked="" type="radio"/> <math>W_{33}^4</math></p> <p>Q Feedback : The elements are of the matrix <math>W^4</math>. It is connecting to 3rd neuron in layer 4 from 3rd neuron in the layer 3. Hence, <math>W_{33}^4</math>.</p>	<p><b>Notations</b></p> <p>How will you represent the bias of the neuron denoted by <math>a</math>?</p> <hr/> <p><input checked="" type="radio"/> <math>b_2^3</math></p> <p>Q Feedback : It is the bias of the 3rd layer for the 2nd neuron. Hence, <math>b_2^3</math>.</p> <p><input type="radio"/> <math>b_2</math></p>
<p><b>Notations</b></p> <p>How will you represent the output of the neuron denoted by <math>c</math>?</p> <hr/> <p><input checked="" type="radio"/> <math>h_5^2</math></p> <p>Q Feedback : It is the output of the 2nd layer for the 5th neuron. Hence, <math>h_5^2</math>.</p> <p><input type="radio"/> <math>b_5^2</math></p>	

## Activation Functions

Previous segments covered the topology, the underlying simplifying assumptions and the hyperparameters and parameters of neural networks. Next is how the output is calculated from a single neuron using an **activation function** and various types and properties of common activation functions.

<p><b>Choosing an Activation Function</b></p>  <p> <math>X = 4.999</math>  <math>= 0</math>  <math>X = 5.001</math>  <math>= 20</math> </p> <p>Bad Activation function</p>	<p><b>Choosing an Activation Function</b></p>  <ul style="list-style-type: none"> <li>Activation functions should be smooth</li> <li>Injects non linearity between input and output</li> </ul>
<p><b>Types of Activation functions</b></p> <ul style="list-style-type: none"> <li>Sigmoid / Logistic.</li> </ul> $f(y) = \frac{1}{1 + e^{-y}}$ $\tanh(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$ <p>RELU (Rectilinear Unit)</p> 	<p><b>Calculating Output of a Neuron</b></p>  <p>Logistic Activation function</p> $a = \frac{1}{1 + e^{-y}}$ $y = (a_1 w_1 + a_2 w_2 + a_3 w_3 + b)$

The activation function could be any function, though it should have some important properties such as:

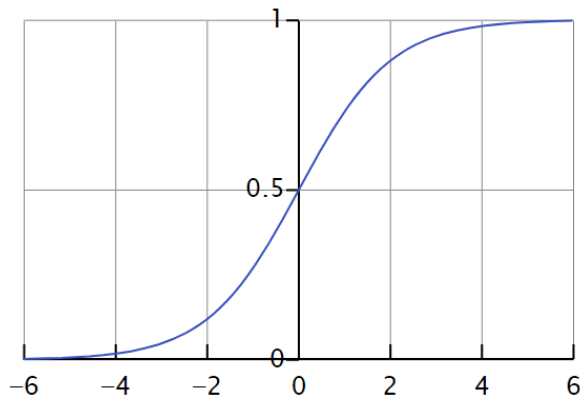
- Activation functions should be **smooth** i.e. they should have no abrupt changes when plotted.
- They should also make the inputs and outputs **non-linear** with respect to each other to some extent. This is because non-linearity helps in making neural networks more compact.

Most popular activation functions used for neural networks are:

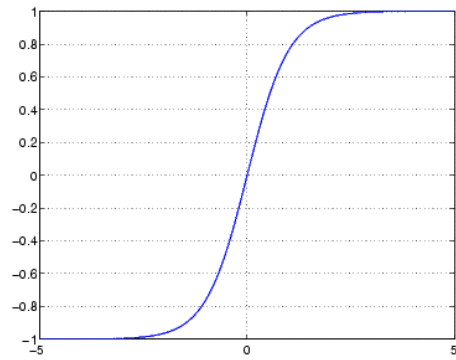
- Logistic function -  $output = f(x) = \frac{1}{1 + e^{-x}}$

Hyperbolic tangent function

$$output = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



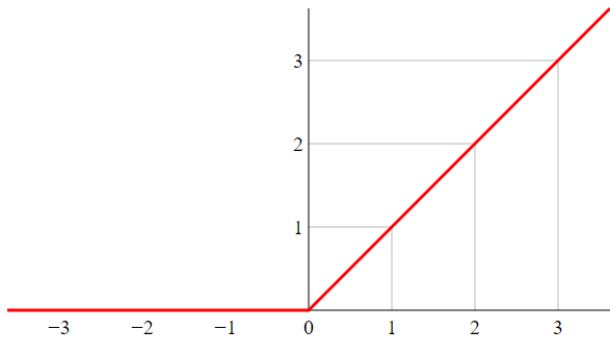
Sigmoid



tanh

It is similar to the sigmoid function.

Rectilinear Unit - output=x for  $x \geq 0$  and 0 otherwise.



The output of a neuron is basically the activation function applied to the cumulative input to that neuron. If the cumulative input to the neuron is  $y = w_1a_1 + w_2a_2 + \dots + w_k a_k + b$  then using the logistic/sigmoid activation function the output of that neuron will be

$$a = \frac{1}{1 + e^{-(w_1a_1 + w_2a_2 + \dots + w_k a_k + b)}}$$

There are some other activation functions like leaking Relu which is defined as

output=x for  $x \geq 0$  output= $\alpha x$  otherwise.

There are some other activation functions like leaking Relu which is defined as

output=x for  $x \geq 0$  output= $\alpha x$  otherwise.

<p><b>Cumulative Input</b></p> <p>Consider only a single neuron with the following weight vector, <math>w = \begin{bmatrix} 2 \\ -6 \\ 3 \end{bmatrix}</math>, the input vector <math>x = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}</math> and the bias <math>b = -1</math>.</p> <p>What is the cumulative input into the neuron?</p> <p> <input type="radio"/> 4  <input checked="" type="radio"/> -4  <input type="radio"/> -1         </p> <p>Q Feedback: Cumulative input = <math>w_1x_1 + w_2x_2 + w_3x_3 + b = -4</math></p>	<p><b>ReLU</b></p> <p>Consider only a single neuron with the following weight vector, <math>w = \begin{bmatrix} 2 \\ -6 \\ 3 \end{bmatrix}</math>, the input vector <math>x = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}</math> and the bias <math>b = -1</math>.</p> <p>What is the output of the neuron if you use the ReLU activation function? (up to 3 decimal places)</p> <p> <input type="radio"/> -4  <input type="radio"/> 4  <input checked="" type="radio"/> 0         </p> <p>Q Feedback: Output = <math>y = x</math> for <math>x &gt; 0</math> and 0 otherwise. Since, <math>y = -4</math> which is <math>&lt; 0</math>. Hence output = 0.</p>
<p><b>Leaky ReLU</b></p> <p>Consider only a single neuron with the following weight vector, <math>w = \begin{bmatrix} 2 \\ -6 \\ 3 \end{bmatrix}</math>, the input vector <math>x = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}</math> and the bias <math>b = -1</math>.</p> <p>What is the output of the neuron if you use the Leaky ReLU activation function with <math>\alpha = 0.2</math>? (up to 3 decimal places)</p> <p> <input type="radio"/> 0.8  <input checked="" type="radio"/> -0.8  <input type="radio"/> 4         </p> <p>Q Feedback: For <math>y &lt; 0</math>, Leaky ReLU output = <math>\alpha x = -0.8</math></p>	<p><b>Sigmoid activation</b></p> <p>Consider only a single neuron with the following weight vector, <math>w = \begin{bmatrix} 2 \\ -6 \\ 3 \end{bmatrix}</math>, the input vector <math>x = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}</math> and the bias <math>b = -1</math>.</p> <p>What is the output of the neuron if you use the sigmoid/ logistic activation function? (up to 3 decimal places)</p> <p> <input type="radio"/> 0.018  <input checked="" type="radio"/> 0.982         </p> <p>Q Feedback: <math>f(y) = \frac{1}{1+e^{-y}}</math>, <math>y = -4</math>, Hence, output = 0.018</p> <p>Q Feedback: <math>f(y) = \frac{1}{1+e^{-y}}</math>, <math>y = -4</math>, Hence, output = 0.018</p>

In a sigmoid neuron, if you multiply the weights and biases by a positive constant  $c > 0$ , as the limit as  $c \rightarrow \infty$  the behaviour of this sigmoid neurons is the same as that of a perceptron, given  $w^*x + b \neq 0$  for the input  $x$ .

In the next segment, how a neural network can be made on a very popular dataset called the MNIST dataset which contains images of handwritten digits between 0-9.

## Summary

How the architecture of Artificial Neural Networks draws inspiration from the human brain.

How **the perceptron** acts as a binary classifier and can perform complex classification tasks.

**The architecture of ANNs** - the topology, the parameters (weights and biases) on which the neural network is trained, the hyperparameters, etc.

Some **simplifying assumptions** in the architecture.

ANNs only take **numerical inputs**, and hence, you need to convert all types of data to a numeric format so that neural networks can process it. Next, we fixed **the notations** that we would use in the future segments and modules.

1.  $W$  is for weight matrix
2.  $b$  shall stand for the bias
3.  $x$  stands for input
4.  $y$  is the ground truth label
5.  $p$  is the probability vector of the predicted output
6.  $h$  is the output of the hidden layers
7. superscript stands for layer number
8. subscript stands for the index of the individual neuron

Also introduced to the most **common activation functions** such as sigmoid, relu, tanh and leaky relu.



