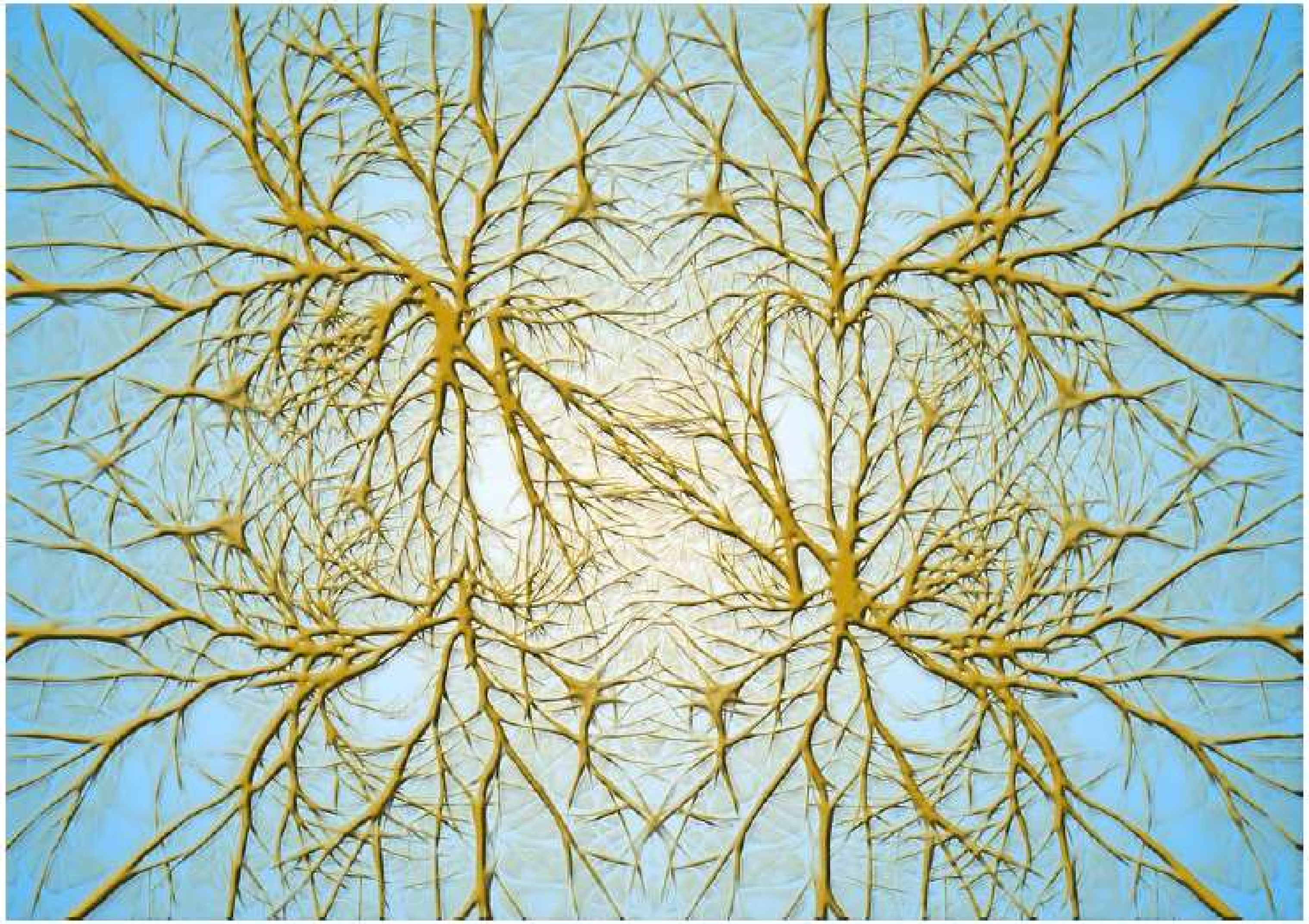
ANN : Multi Layered Perceptron (MLP)

Sourav Karmakar

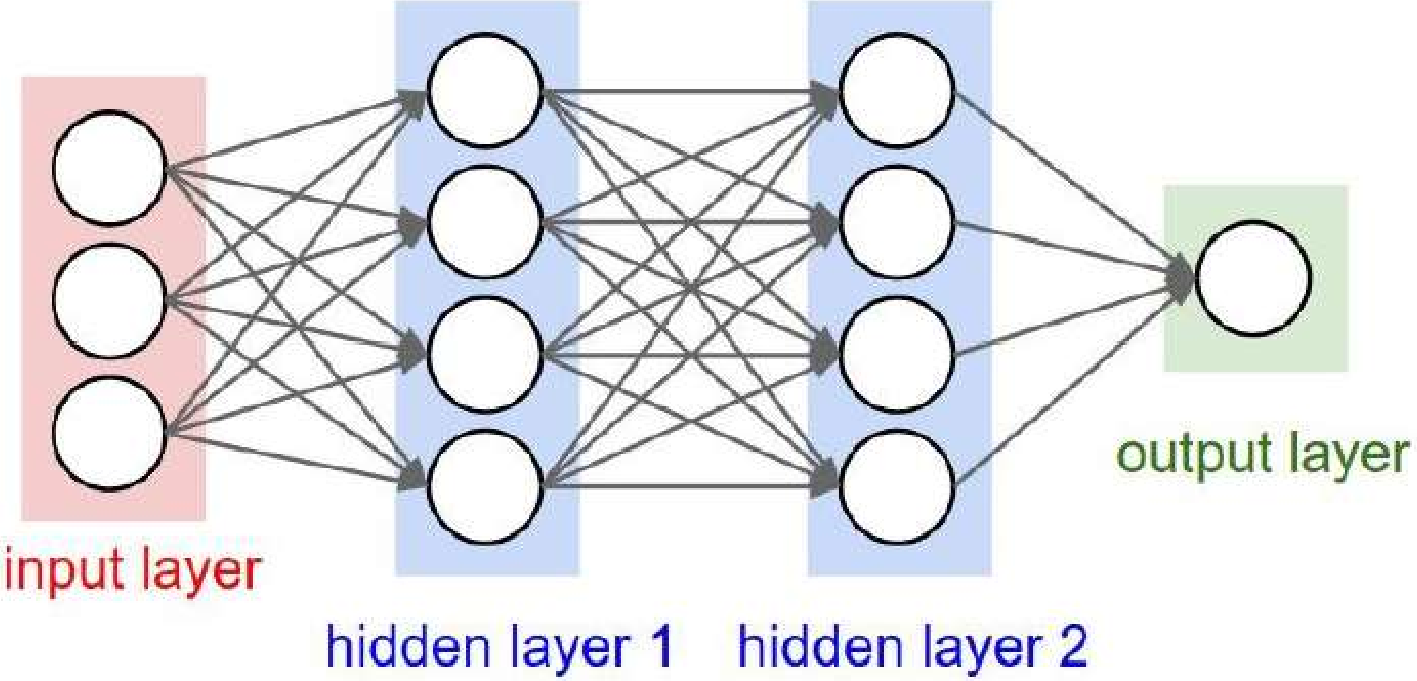
souravkarmakar29@gmail.com

Artificial Neural Networks

Biological Neurons:
Complex connectivity patterns

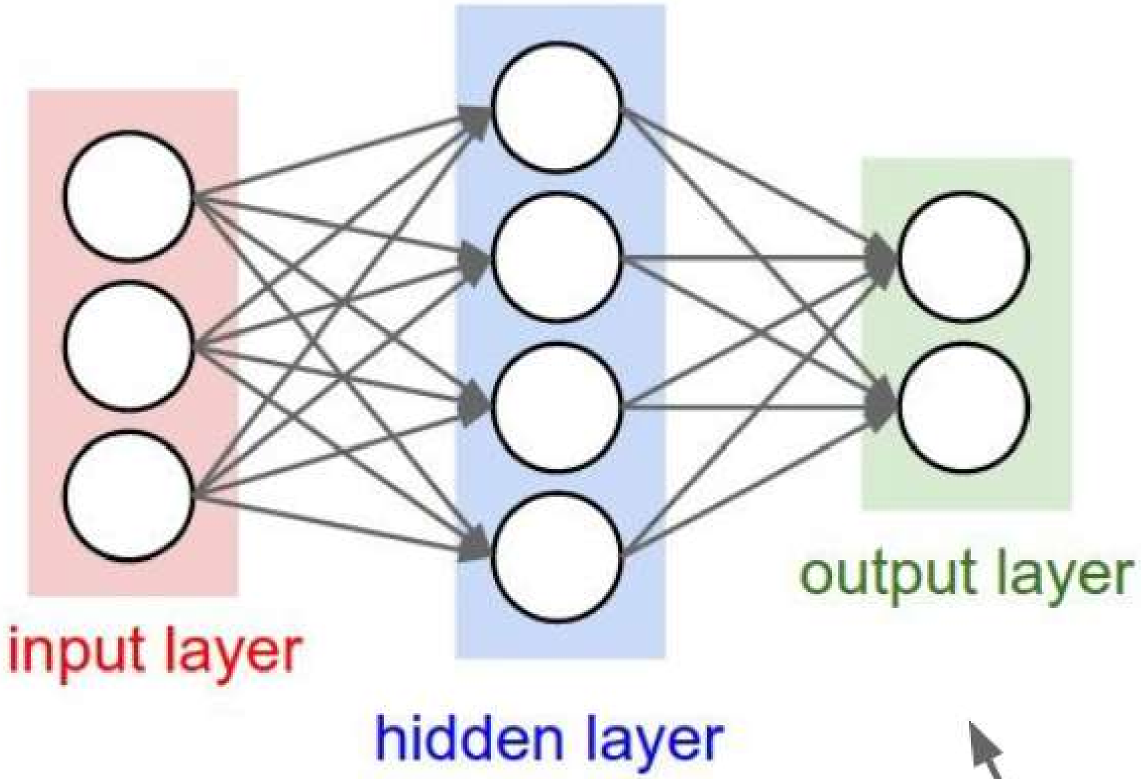


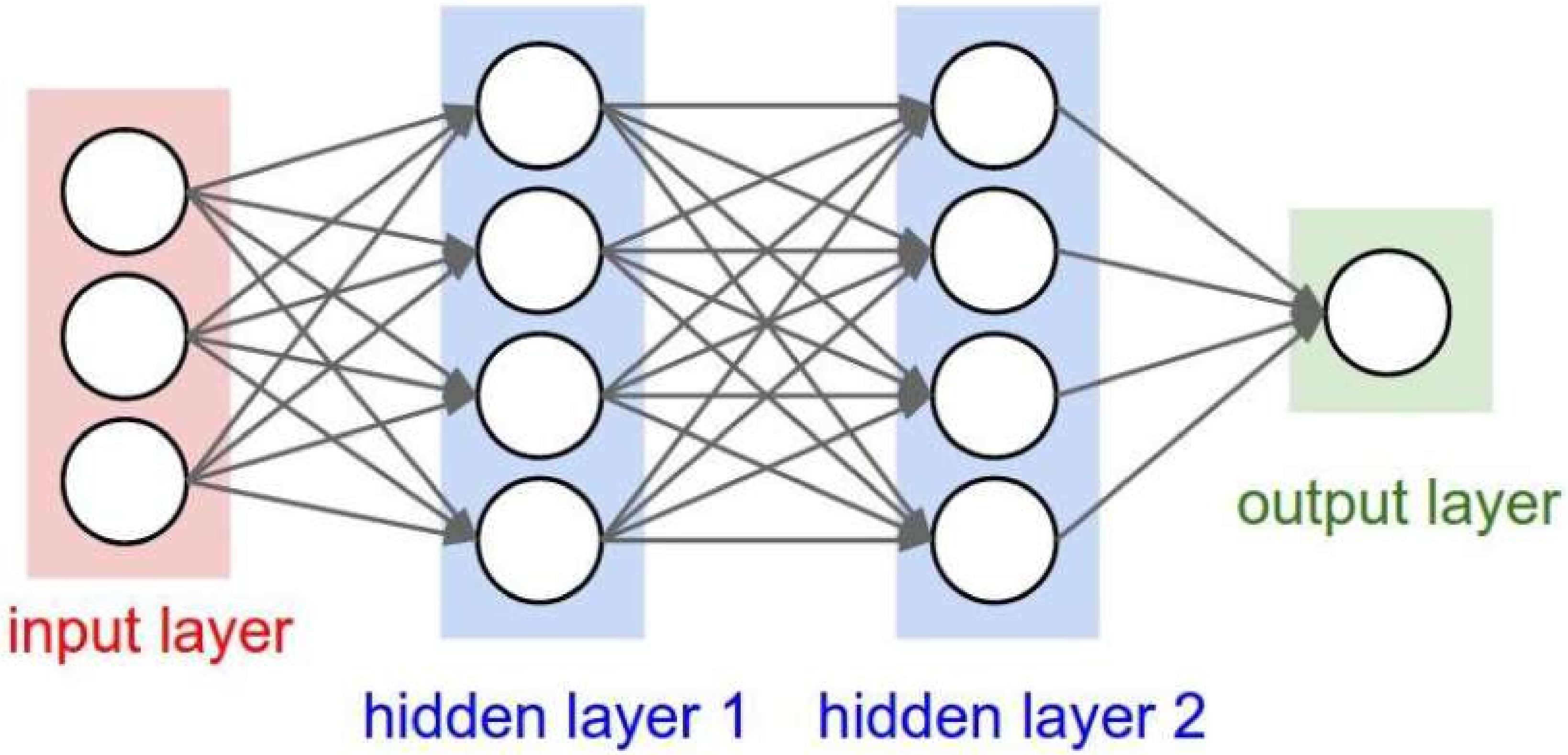
Neurons in a neural network:
Organized into regular layers for computational efficiency



This type of architectures are called Multi-Layered Perceptron (MLP)

Multi Layered Perceptron (MLP)





“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

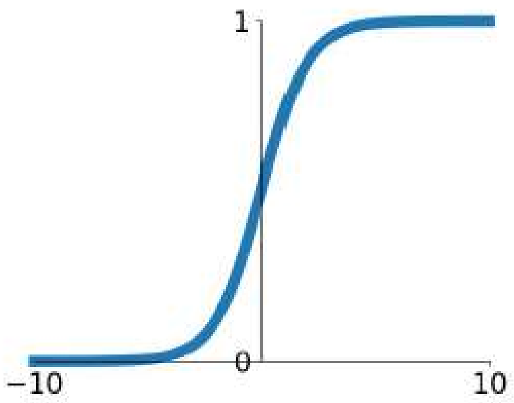
“Fully-connected” layers

Fully-connected layers are also called **Dense Layers**

Different Activation Functions

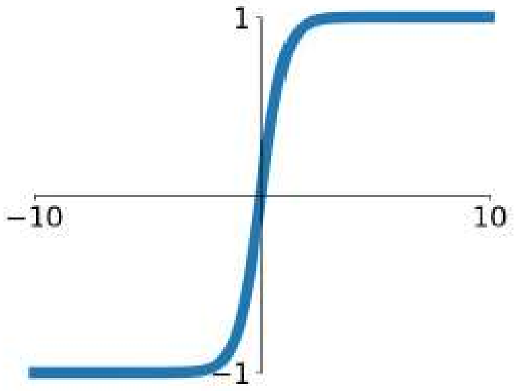
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



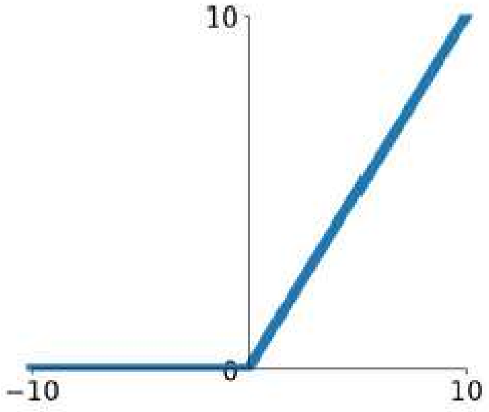
tanh

$$\tanh(x)$$



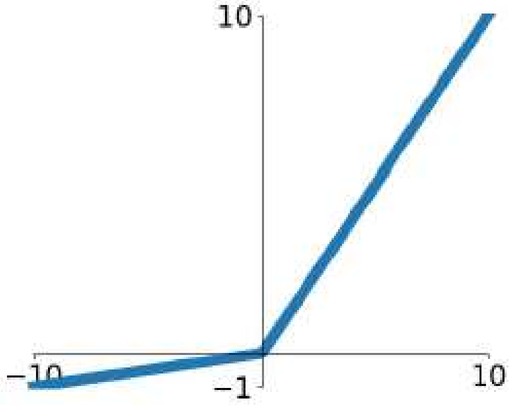
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

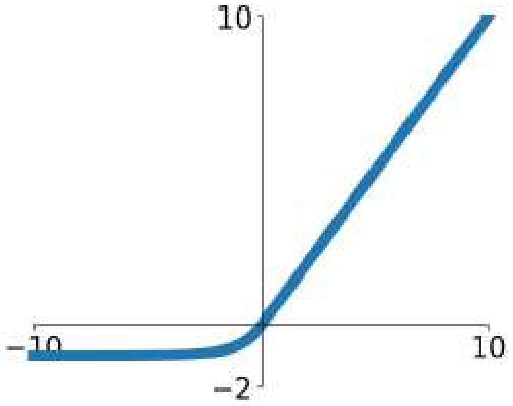


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ReLU is a good practical choice for most problems

Softmax Regression

Softmax Regression is a generalization of Logistic Regression for multiclass classification problems.

- **Training Set:** $\{ \langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle, \langle \boldsymbol{x}^{(2)}, y^{(2)} \rangle, \dots, \langle \boldsymbol{x}^{(m)}, y^{(m)} \rangle \}$,

Where m = Number of training data, the feature vector $\boldsymbol{x}^{(i)} \in \mathbb{R}^{n+1}$ $(x_0 = 1)$, and the labels $y^{(i)} \in \{1, 2, 3, \dots, K\}$ (K = Number of classes).

- **Hypothesis:** $h_\theta\left(\boldsymbol{x}^{(i)}\right) = \frac{1}{\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}^{(i)}}} \begin{bmatrix} e^{\boldsymbol{\theta}_1^T \boldsymbol{x}^{(i)}} \\ e^{\boldsymbol{\theta}_2^T \boldsymbol{x}^{(i)}} \\ \vdots \\ e^{\boldsymbol{\theta}_K^T \boldsymbol{x}^{(i)}} \end{bmatrix}$

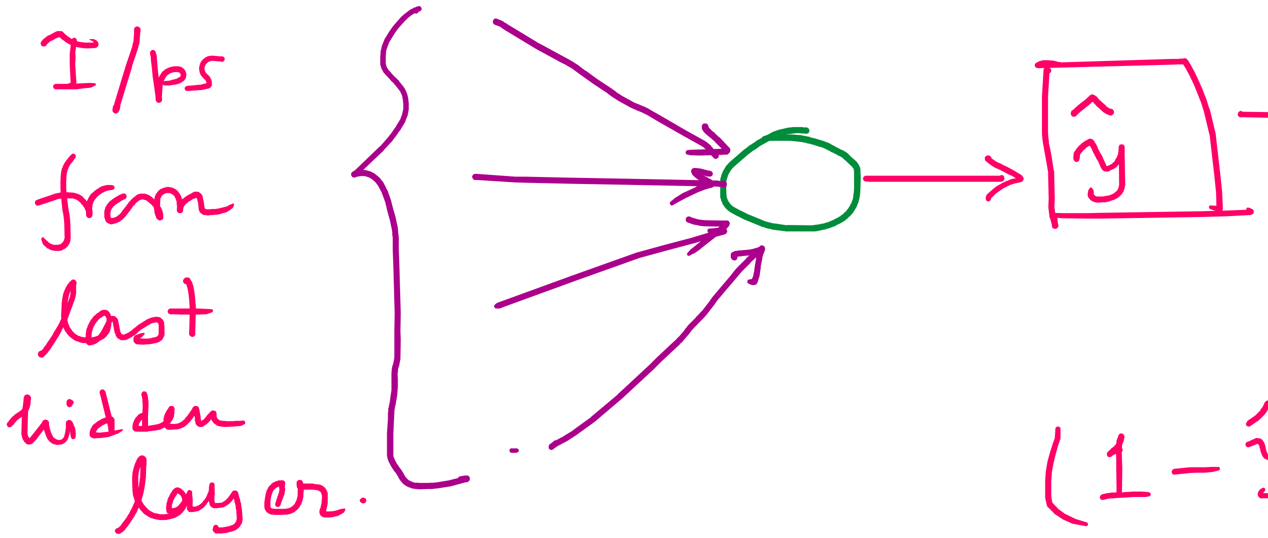
Where $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_K \in \mathbb{R}^{n+1}$, are the parameters of our model. The denominator $\sum_{j=1}^{K} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}^{(i)}}$ normalizes the distribution, so that it sums to one.

Usually Softmax regression is used at the output / classification layer of the ANN. Where it is also referred to as **Softmax Activation function**.

Softmax Activation :-

Binary classification → class-1
→ class-2

In case of binary classification there is only one o/p node.



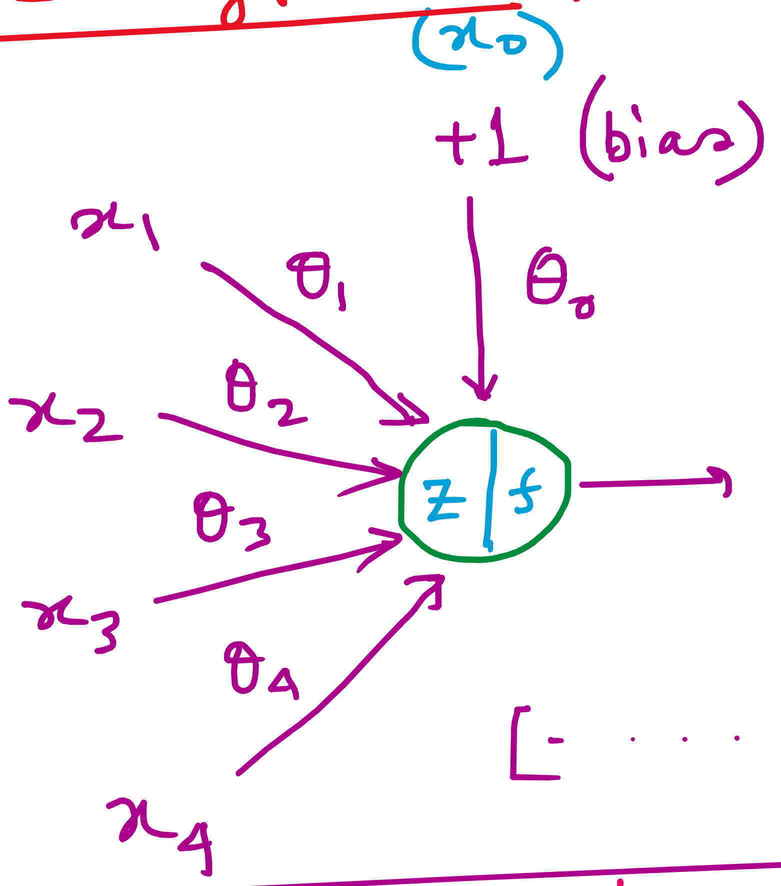
→ This o/p gives the probability of the observation belonging to class-1.

$(1-\hat{y})$ → probability that the observation (i/p) belongs to class-2.

$\hat{y} = 0.7$ → This means the i/p has 0.7 (70%) probability to belong to class-1.

→ It has 30% probability to belong to class-2.

Logistic Regression:-



$$\hat{y} = \frac{1}{1+e^{-z}}$$

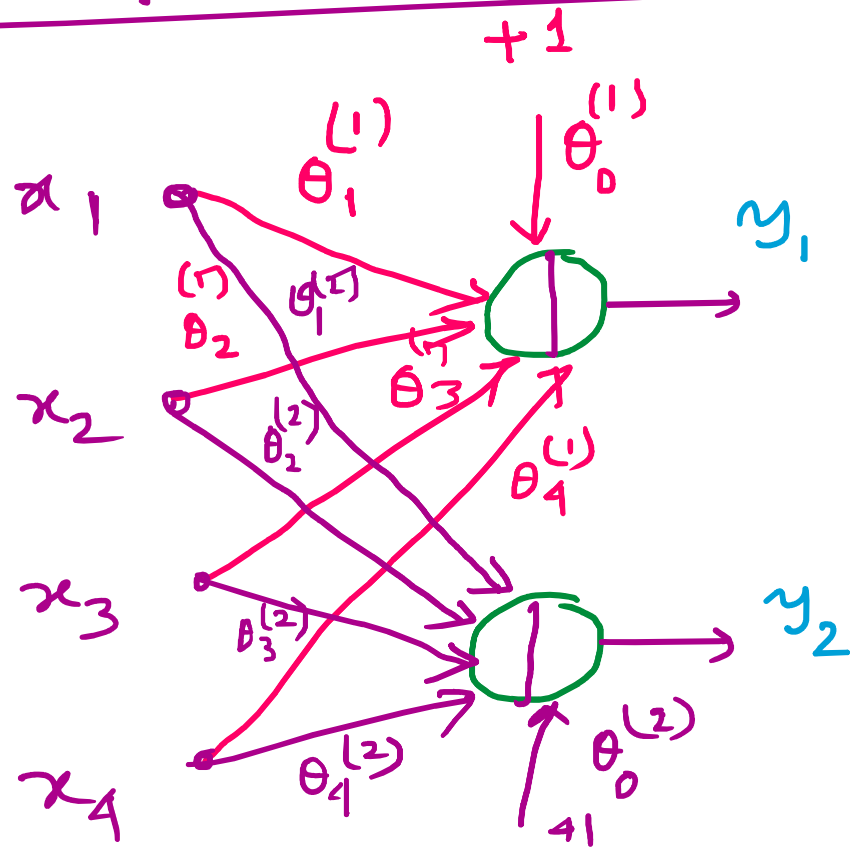
$$f(\cdot) = \text{sigmoid} \rightarrow f(x) = \frac{1}{1+e^{-x}}$$

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$= \sum_{i=0}^{4} \theta_i x_i$$

$[\cdot \; . \; . \; . \; . \; . \; -] \begin{bmatrix} : \\ : \end{bmatrix} = -$

$$= \frac{1}{1+e^{-\sum_{i=0}^{4}\theta_i x_i}} = \frac{1}{1+e^{-\vec{\theta}^T \vec{x}}}$$



$$\left[\theta_0^{(1)}, \theta_1^{(1)}, \theta_2^{(1)}, \theta_3^{(1)}, \theta_4^{(1)}\right]^T \rightarrow \vec{\theta}_1$$

$$\left[\theta_0^{(2)}, \theta_1^{(2)}, \theta_2^{(2)}, \theta_3^{(2)}, \theta_4^{(2)}\right]^T \rightarrow \vec{\theta}_2$$

$$y_1 = \frac{1}{1+e^{-\vec{\theta}_1^T \vec{x}}} \qquad y_2 = \frac{1}{1+e^{-\vec{\theta}_2^T \vec{x}}}$$

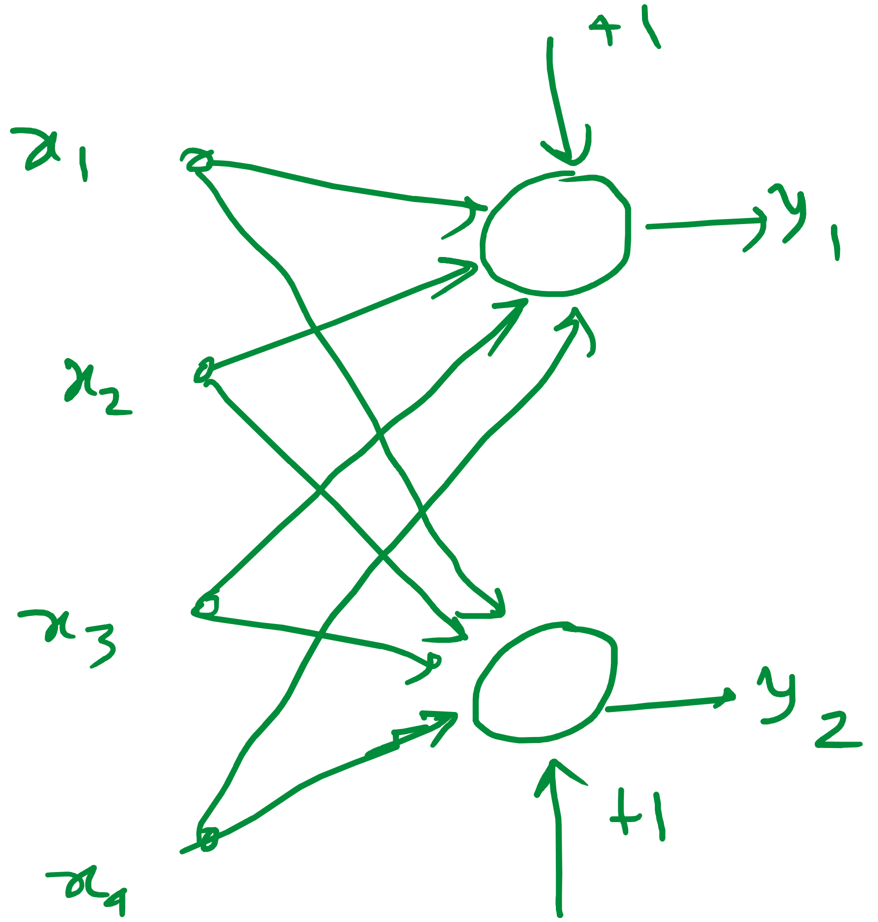
✗ $\boxed{y_1 + y_2 = 1}$

$$y_1 = \frac{1}{1+e^{-\vec{\theta}_1^T \vec{x}}} \quad , \quad y_2 = \frac{1}{1+e^{-\vec{\theta}_2^T \vec{x}}}$$

$$\left(\frac{e^{\vec{\theta}_1^T \vec{x}}}{1+e^{\vec{\theta}_1^T \vec{x}}}\right) + \left(\frac{e^{\vec{\theta}_2^T \vec{x}}}{1+e^{\vec{\theta}_2^T \vec{x}}}\right)$$

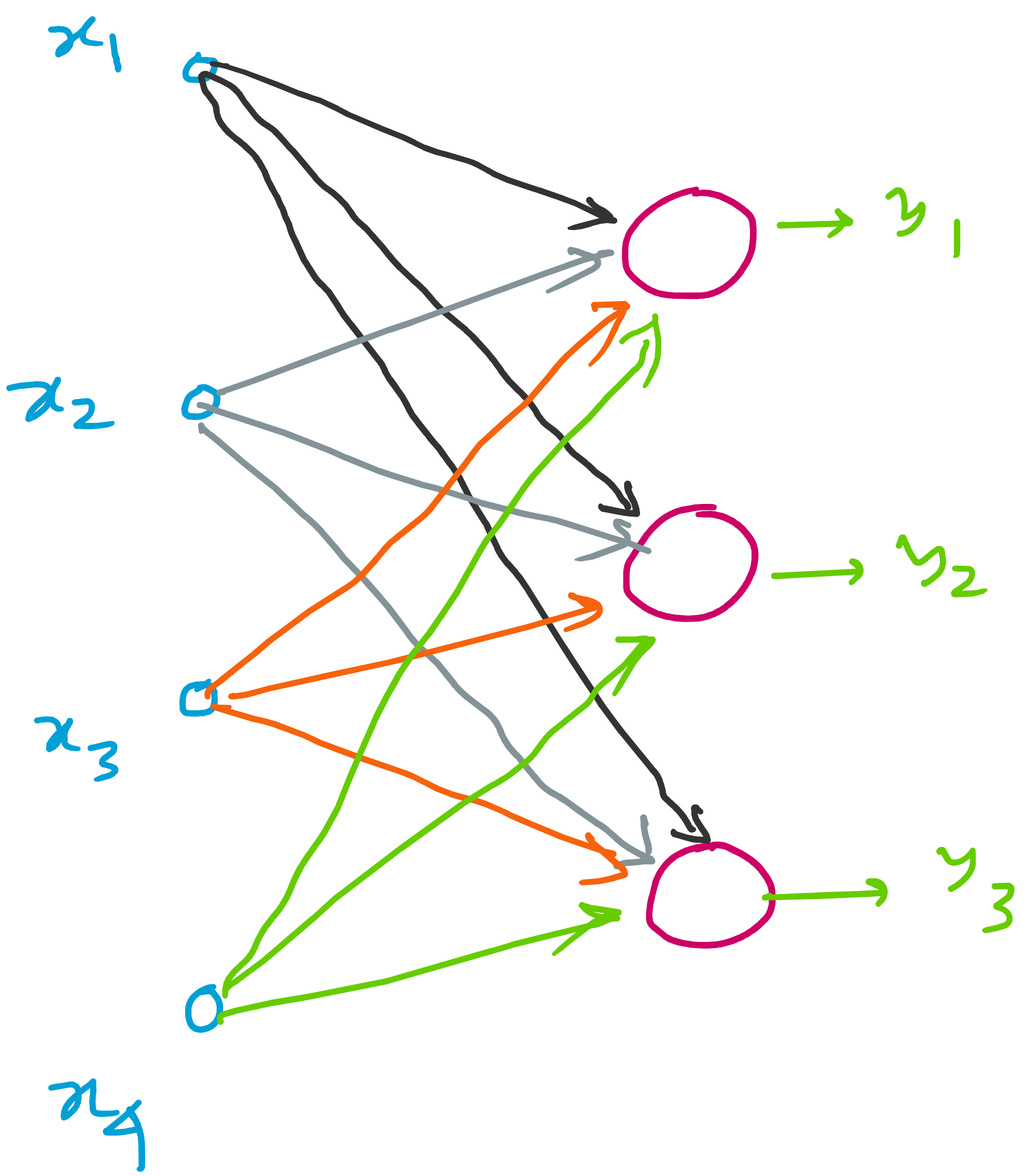
$$\frac{1}{1+e^{-\vec{\theta}_1^T \vec{x}}} + \frac{1}{1+e^{-\vec{\theta}_2^T \vec{x}}} = 1$$

using two separate neurons (both of them using sigmoid).



$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{e^{\vec{\theta}_1^T \vec{x}} + e^{\vec{\theta}_2^T \vec{x}}} \begin{bmatrix} e^{\vec{\theta}_1^T \vec{x}} \\ e^{\vec{\theta}_2^T \vec{x}} \end{bmatrix}$$

$$y_1 = \frac{e^{\vec{\theta}_1^T x}}{e^{\vec{\theta}_1^T \vec{x}} + e^{\vec{\theta}_2^T \vec{x}}} \quad , \quad y_2 = \frac{e^{\vec{\theta}_2^T \vec{x}}}{e^{\vec{\theta}_1^T \vec{x}} + e^{\vec{\theta}_2^T \vec{x}}}$$



$$y_1 = \frac{e^{\vec{\theta}_1^{\,T} \vec{x}}}{e^{\vec{\theta}_1^{\,T} \vec{x}} + e^{\vec{\theta}_2^{\,T} \vec{x}} + e^{\vec{\theta}_3^{\,T} \vec{x}}}$$

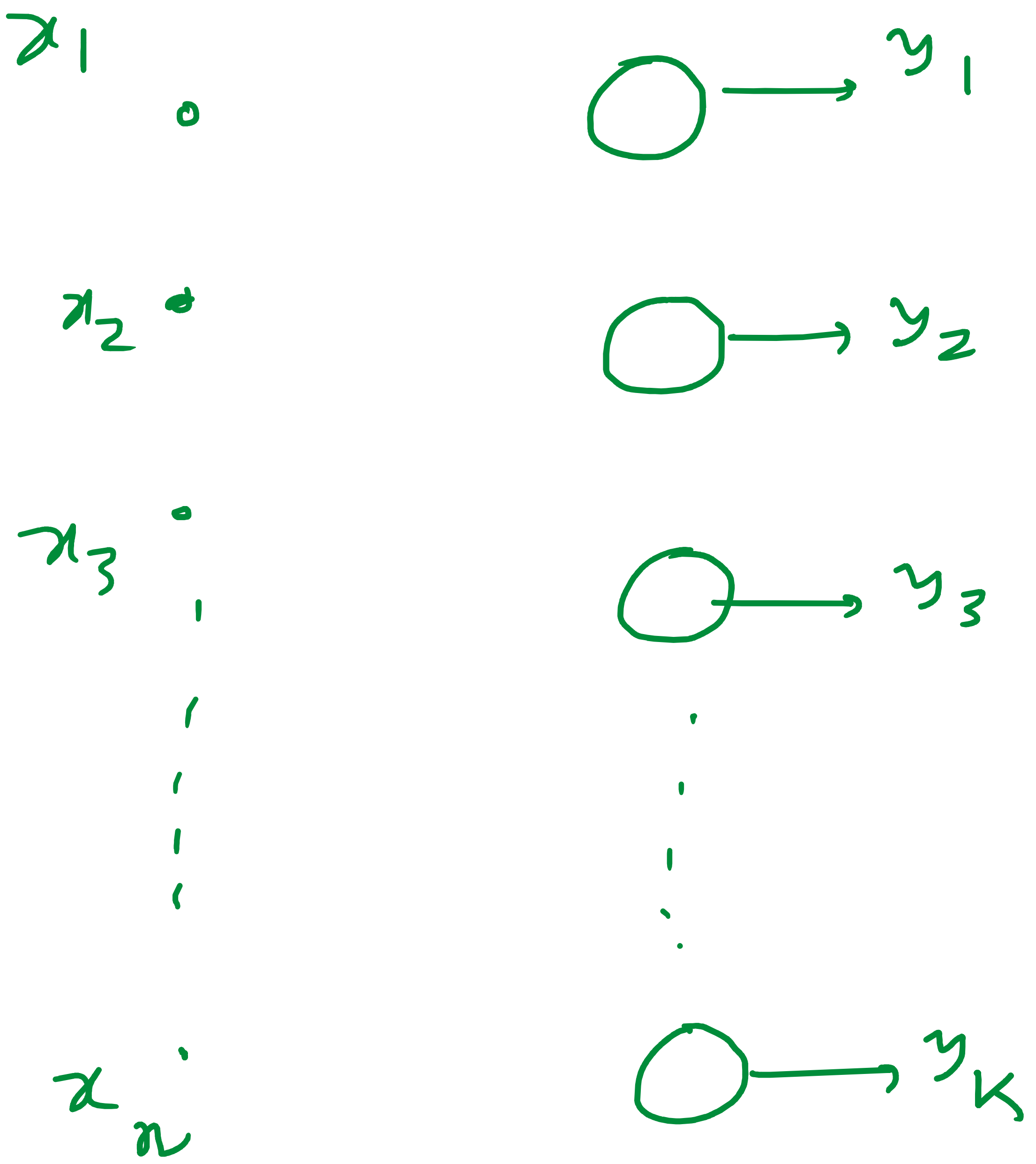
$$y_2 = \frac{e^{\vec{\theta}_2^{\,T} \vec{x}}}{e^{\vec{\theta}_1^{\,T} \vec{x}} + e^{\vec{\theta}_2^{\,T} \vec{x}} + e^{\vec{\theta}_3^{\,T} \vec{x}}}$$

$$y_3 = \frac{e^{\vec{\theta}_3^{\,T} \vec{x}}}{e^{\vec{\theta}_1^{\,T} \vec{x}} + e^{\vec{\theta}_2^{\,T} \vec{x}} + e^{\vec{\theta}_3^{\,T} \vec{x}}}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \frac{1}{e^{\vec{\theta}_1^{\,T} \vec{x}} + e^{\vec{\theta}_2^{\,T} \vec{x}} + e^{\vec{\theta}_3^{\,T} \vec{x}}} \begin{bmatrix} e^{\vec{\theta}_1^{\,T} \vec{x}} \\ e^{\vec{\theta}_2^{\,T} \vec{x}} \\ e^{\vec{\theta}_3^{\,T} \vec{x}} \end{bmatrix}$$

for k class classification

There will k output nodes.



$$\vec{x} = [1, x_1, x_2, \ldots\ldots, x_n]^T$$

$$\underset{\text{(o|p vector)}}{\vec{y}} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_K \end{bmatrix} = \underbrace{\frac{1}{\sum_{i=1}^{K} e^{\vec{\theta}_i^T \vec{x}_i}} \begin{bmatrix} e^{\vec{\theta}_1^T \vec{x}} \\ e^{\vec{\theta}_2^T \vec{x}} \\ \vdots \\ e^{\vec{\theta}_K^T \vec{x}} \end{bmatrix}}_{\text{Softmax Regression/ Softmax Activation.}}$$

Multi Layered Perceptron (MLP)

Architecture:

- The no. of nodes in the input layer = no. of features or attributes of the data.
- The no. of nodes in the output layer = no. of classes of the dataset.
- No. of Hidden layers and No. of nodes in each hidden layer is still user choice and there is no general guideline for the choice of no. of hidden layers or the no. of nodes in each hidden layers.
- There should be at least one hidden layer.
- Usually we use **'ReLU'** activation for all nodes in hidden layer and **'softmax'** activation in output layer for multi-class classification problem or **'sigmoid'** activation in output layer for binary classification problem. However we can also use different activation functions in the hidden layer.

For example: To classify the simple IRIS dataset using ANN

- The number of input nodes = 4 (as the dataset has four features / attributes)
- The number of output nodes = 3 (as there are three classes)
- We can build an ANN with one hidden layer containing 4 nodes (say)

ANN (MLP) Architecture :-

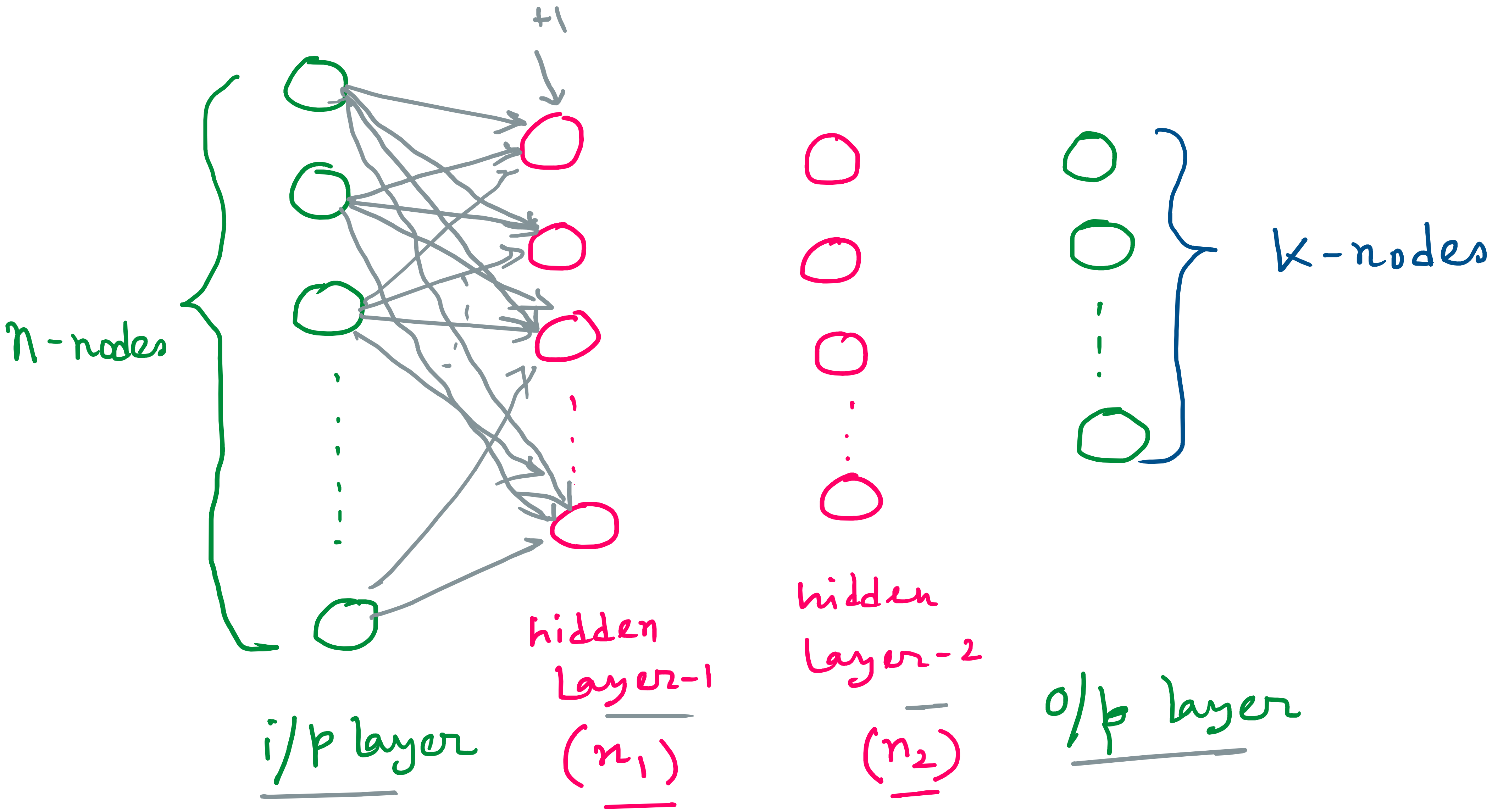
Dataset :- n - attributes / features (model ready)

m

x_1	x_2	x_3	$\cdots$	x_n	y

$y \in \{1, 2, 3, \ldots, k\}$

k - classes.



There should be atleast one hidden layer.

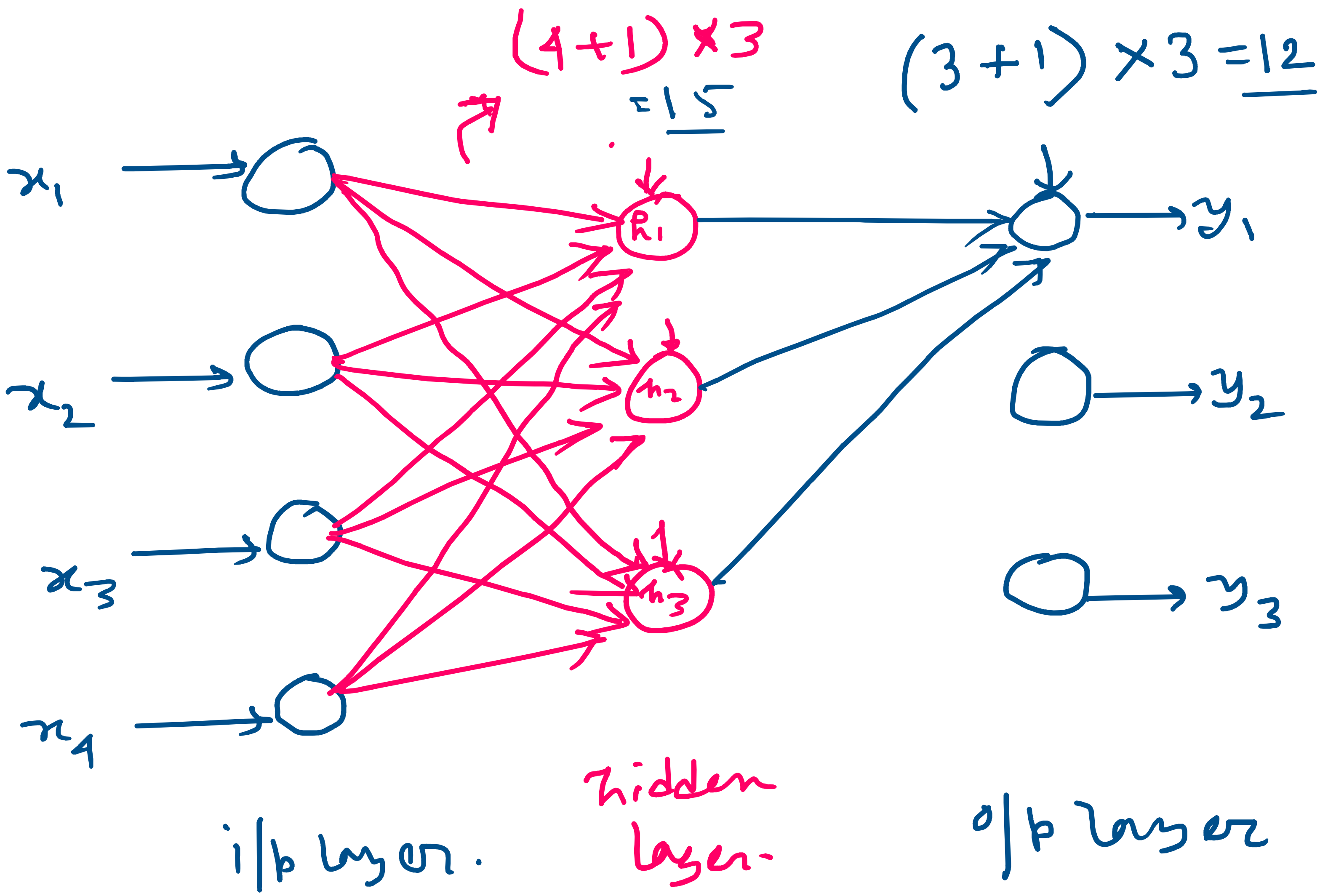
1) Number of hidden layers

2) Number of nodes in each hidden layers

3) Activation functions in the hidden layer.

} user choice (Hyperparameters)

For IRIS dataset:- There are 4 attributes & 3 classes.



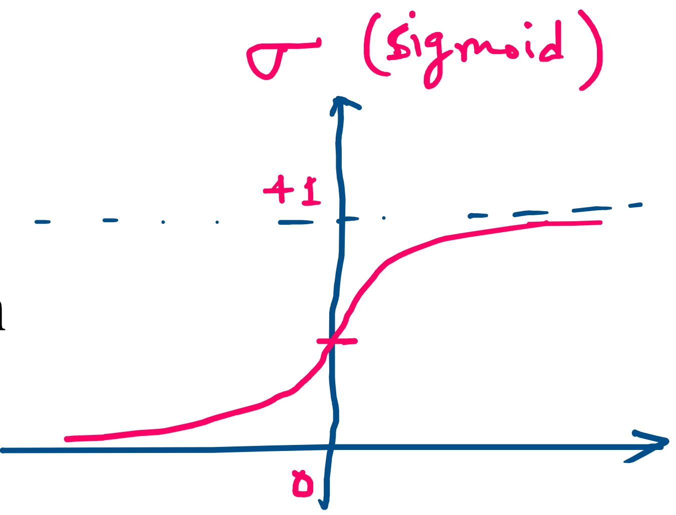
Multi Layered Perceptron (MLP)

Advantages:

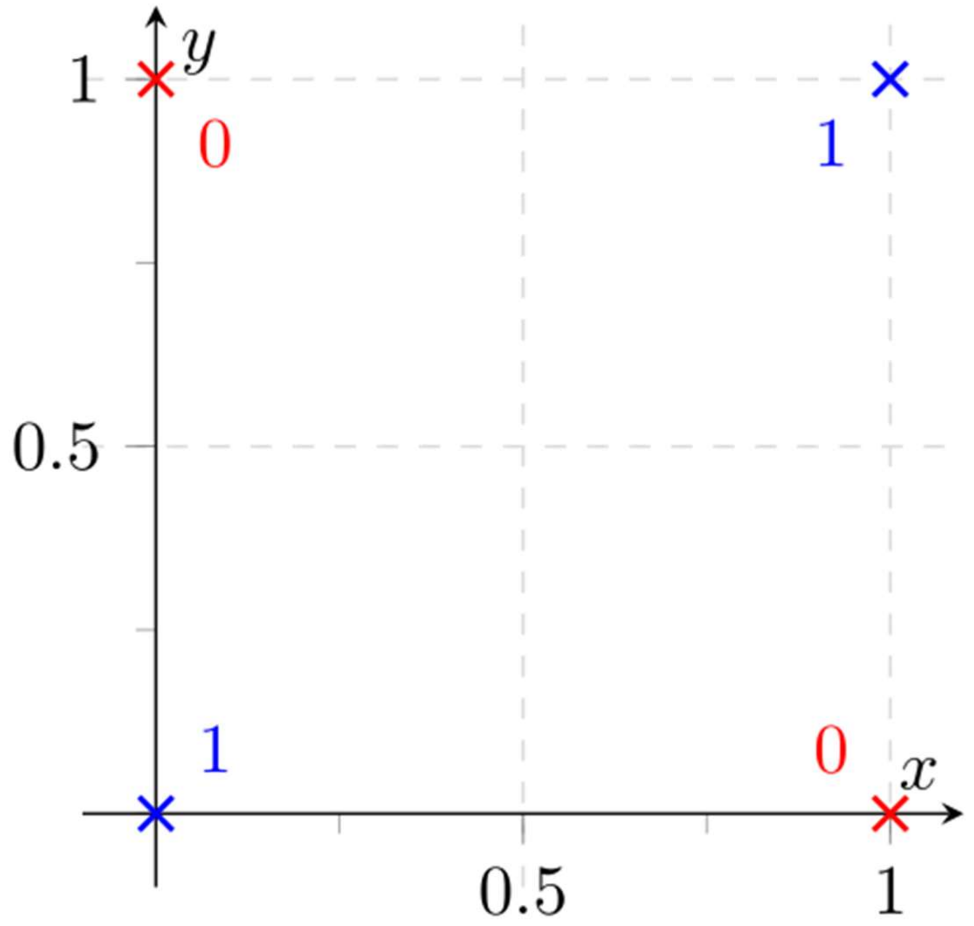
- **Non-linearity:** MLP is very suitable to model a physical phenomenon which is in general non-linear.
- **Adaptivity:** It can cope with the change in dataset distribution.
- **Parallelism:** In can be implemented in multi-core parallel architecture for faster computation.
- **Robustness:** It has inherent ability to handle confusing / noisy / missing datapoints
- **Fault tolerance:** It has the ability to work to some extent even in case of component (neuron) failure.
- **Complex decision boundary:** It has the ability to have a complex decision boundary for classification.
- **VLSI implementability:** It can be implemented quite easily in VLSI chips.

We shall use MLP with supervised learning for complex classification tasks

How MLP Solved The XOR Problem



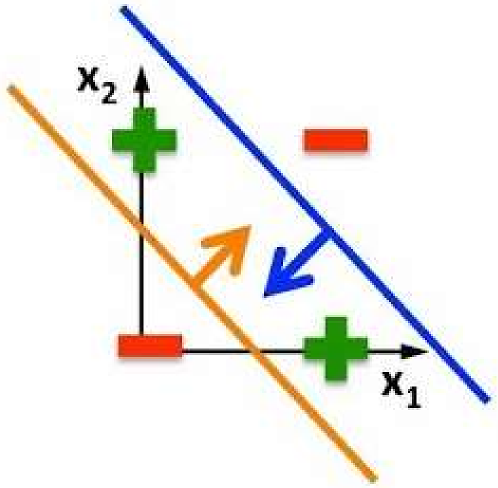
XOR Problem

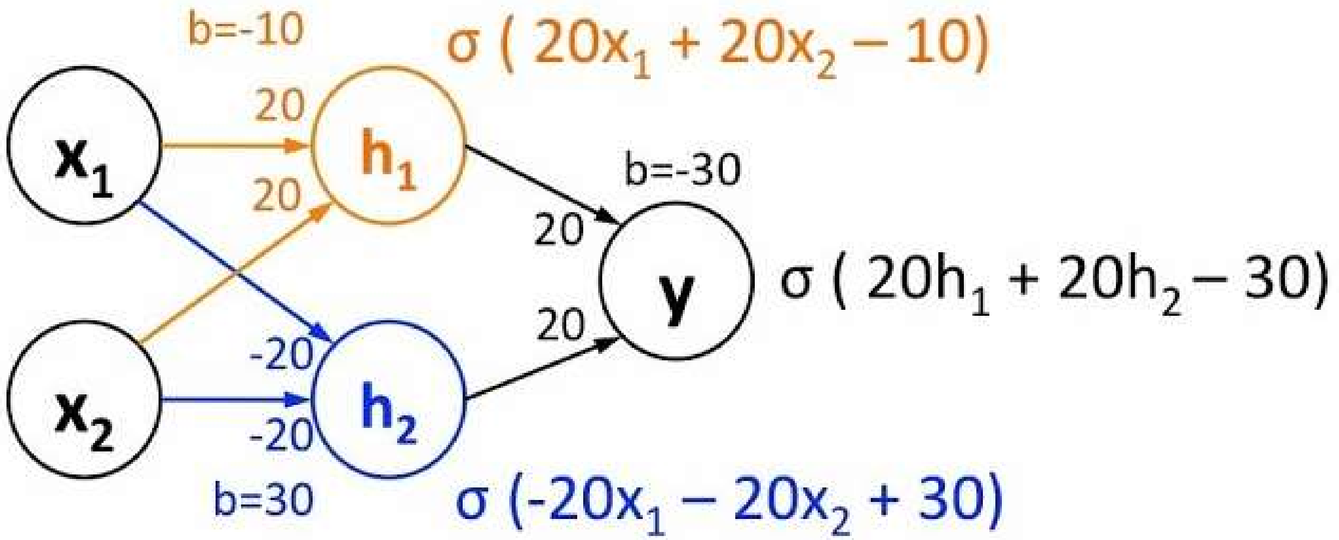


Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

Multi Layered Perceptron

Linear classifiers cannot solve this




b=-10
$\sigma (20x_1 + 20x_2 - 10)$
20
20
x_1
h_1
b=-30
20
20
y
$\sigma (20h_1 + 20h_2 - 30)$
x_2
-20
-20
h_2
b=30
$\sigma (-20x_1 - 20x_2 + 30)$

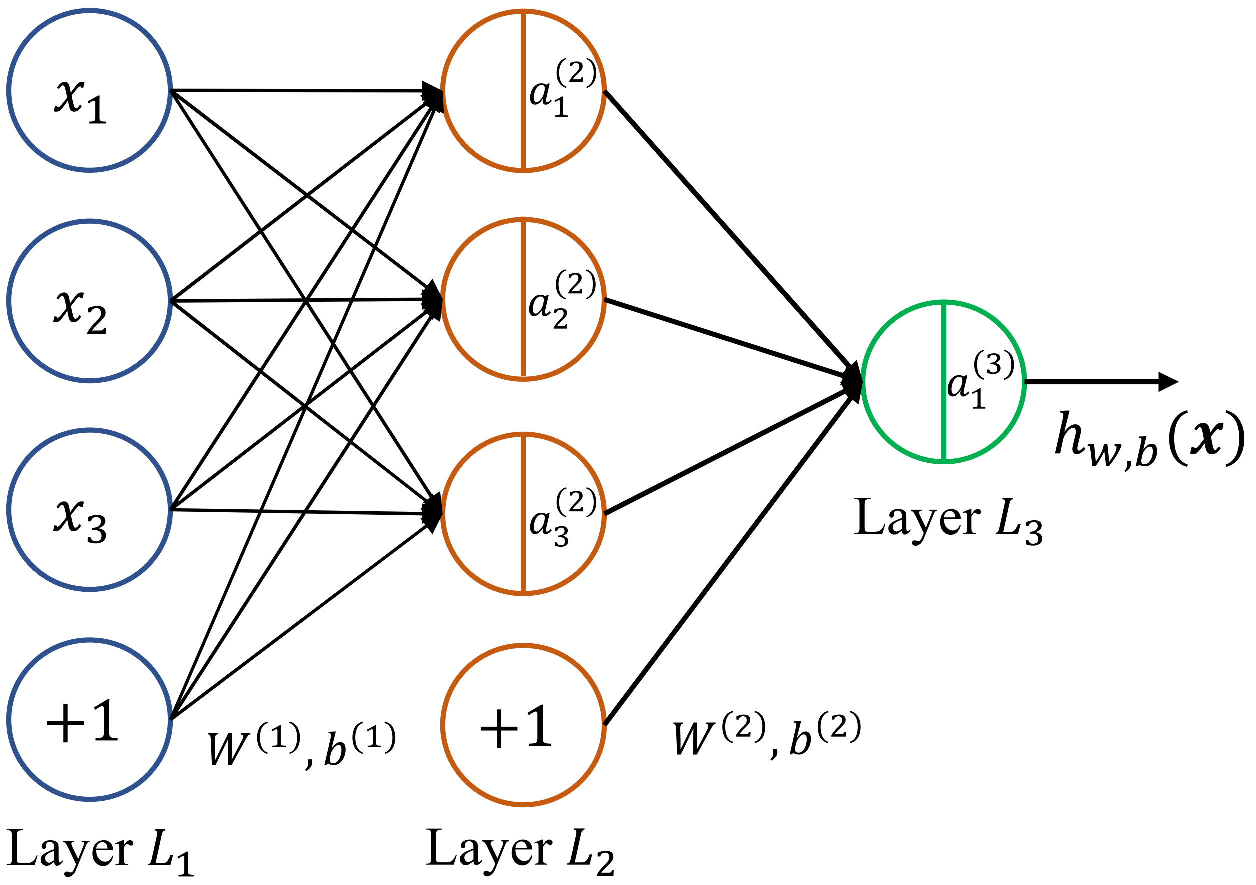
$\sigma(20*0 + 20*0 - 10) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$
$\sigma(20*0 + 20*1 - 10) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$

$\sigma (-20*0 - 20*0 + 30) \approx 1$
$\sigma (-20*1 - 20*1 + 30) \approx 0$
$\sigma (-20*0 - 20*1 + 30) \approx 1$
$\sigma (-20*1 - 20*0 + 30) \approx 1$

$\sigma (20*0 + 20*1 - 30) \approx 0$
$\sigma (20*1 + 20*0 - 30) \approx 0$
$\sigma (20*1 + 20*1 - 30) \approx 1$
$\sigma (20*1 + 20*1 - 30) \approx 1$

Multi Layered Perceptron (MLP)

Key Notations:



- n_l denotes the number of layers in our network; thus in the figure beside $n_l = 3$.
- s_l denotes the number of nodes in layer l (not counting the bias unit)
- L_1 is the input layer and L_{n_l} is the output layer.
- $b_i^{(l)}$ is the bias associated with unit i in layer $l + 1$

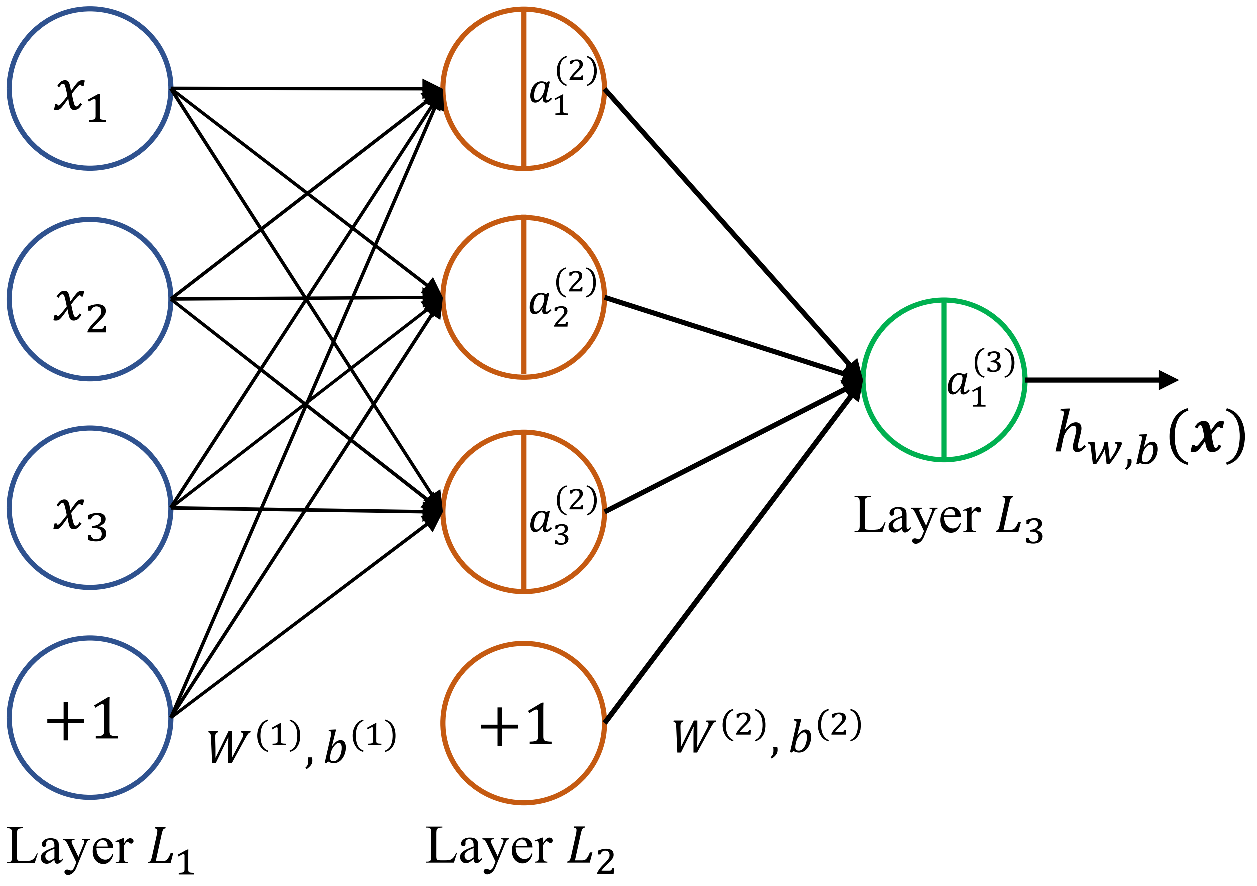
- The neural network as shown above has parameters $\langle W, b\rangle = \langle (W^{(1)}, b^{(1)}), (W^{(2)}, b^{(2)})\rangle$,

Where $W_{ij}^{(l)}$ denotes the parameter (or weight) associated with the connection between unit j in layer l, and unit i in layer $l + 1$. Note that bias units don't have inputs or connections going into them, since they always output value $+1$.

- In this example we have $W^{(1)} \in \mathbb{R}^{3\times3}$, $b^{(1)} \in \mathbb{R}^{1\times3}$, $W^{(2)} \in \mathbb{R}^{1\times3}$, and $b^{(2)} \in \mathbb{R}$

Multi Layered Perceptron (MLP)

Forward Propagation:



- We will write $a_i^{(l)}$ to denote the activation of unit i in layer l. For $l = 1$, we also use $a_i^{(1)} = x_i$ to denote i^{th} input.

$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right) = f\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_2^{(1)}\right) = f\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = f\left(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}\right) = f\left(z_3^{(2)}\right)$$

Here $f(.)$ denotes the activation function.

The final output $h_{w,b}(\boldsymbol{x}) = a_1^{(3)} = f\left(W_{11}^{(2)}\, a_1^{(2)} + W_{12}^{(2)}\, a_2^{(2)} + W_{13}^{(2)}\, a_3^{(2)} + b_1^{(2)}\right) = f\left(z_1^{(3)}\right)$

All the equations shown above can be written in simplified fashion in vector matrix notation:

$$\boldsymbol{a}^{(l+1)} = f\left(\boldsymbol{z}^{(l+1)}\right) = f\left(W^{(l)}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}\right)$$

Multi Layered Perceptron (MLP)

Loss / Cost Function:

Given a training set of ‘m’ samples, we define the overall cost function to be:

Mean Squared Error Loss:

$$J(W,b) = \frac{1}{m}\left[\sum_{i=1}^{m}\frac{1}{2}\left(h_{W,b}(x^{(i)}) - y^{(i)}\right)^{2}\right]$$

Multi Class Log Loss or Categorical Cross-entropy Loss:

$$J(W,b) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_{W,b}(x^{(i)}))_k + (1-y_k^{(i)}) log(1-h_{W,b}(x^{(i)}))_k\right]$$

Usually for classification task Categorical Cross-entropy Loss is more preferred

Multi Layered Perceptron (MLP)

Back Propagation Learning Algorithm:

The weights are updated using Backpropagation Algorithm where each weights are updated using Gradient Descent update rule. For that we have to calculate the partial derivatives of the loss function.

User has to specify:

- Number of layers: n_l.
- Number of nodes in each hidden layer: $s_l \;\; for \;\; l = 2, 3, \ldots, n_l - 1$
- Number of epochs for which the network to be trained.
- Type of gradient descent:
  - ***Batch Gradient Descent:*** in each epoch the error in the final layer is average of errors due to all the training samples in the dataset and update the parameters / weights using backpropagation after all the training samples are fed to the network.
  - ***Minibatch Gradient Descent:*** Compute error in the final layer for a batch (a fraction) of inputs and backpropagation to update weight after each batch is completed. An epoch is said to be complete if all the batches of input data are fed to the network. The batch size should be specified by user.
  - ***Stochastic Gradient Descent:*** in each epoch update weight after each of the training sample is fed.

Multi Layered Perceptron (MLP)

Back Propagation Learning Algorithm:

1. For each epoch : (Number of epochs should be specified by user)
2. Perform a feedforward pass, computing the activations for layers $L_2, L_3,$ and so on up to the output layer L_{n_l}.
3. For each output unit i in layer n_l (the final/output layer), set

$$\delta_i^{(n_l)} = -\left(y_i - a_i^{(n_l)}\right).f'\left(z_i^{(n_l)}\right).$$

4. For $l = (n_l - 1), (n_l - 2), (n_l - 3), \ldots, 2$

For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right).f'\left(z_i^{(l)}\right)$$

5. Compute the desired partial derivatives which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = a_j^{(l)}.\ \delta_i^{(l+1)} \quad and \quad \frac{\partial}{\partial b_i^{(l)}} J(W,b) = \delta_i^{(l+1)}$$

5. Finally compute the changed parameters:

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \eta \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \quad and \quad b_i^{(l)} := b_i^{(l)} - \eta \frac{\partial}{\partial b_i^{(l)}} J(W,b); \quad \eta \text{ is the learning rate}$$

Thank You