Machine Learning (19CSE305)

Backpropagation

Dr. Peeta Basa Pati
Ms. Priyanka V
Department of Computer Science & Engineering,
Amrita School of Engineering, Bengaluru

Topics

- Backpropagation

Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.

Contd..

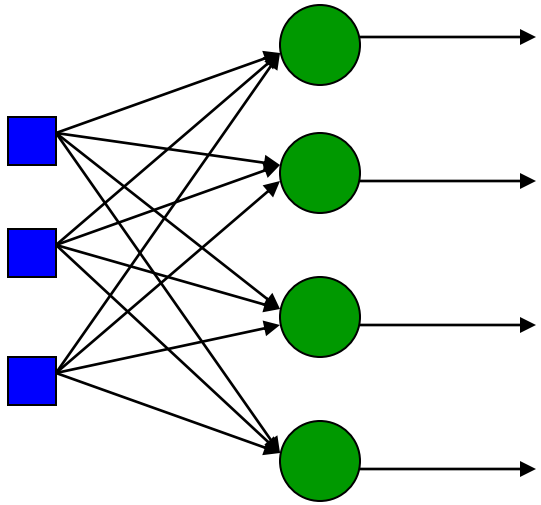
- An Artificial Neural Network is specified by:
  - **neuron model**: the information processing unit of the NN,
  - **an architecture**: a set of neurons and links connecting neurons. Each link has a weight,
  - **a learning algorithm**: used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.
- The aim is to obtain a NN that is trained and generalizes well.
- It should behaves correctly on new instances of the learning task.

Network Architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

Input layer of source nodes

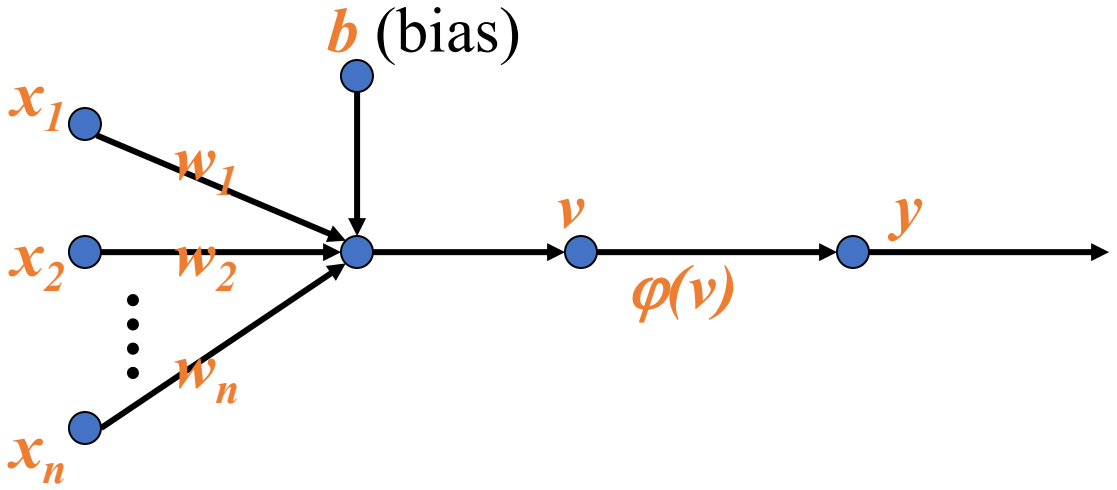


Output layer of neurons

Perceptron: Neuron Model
(Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$

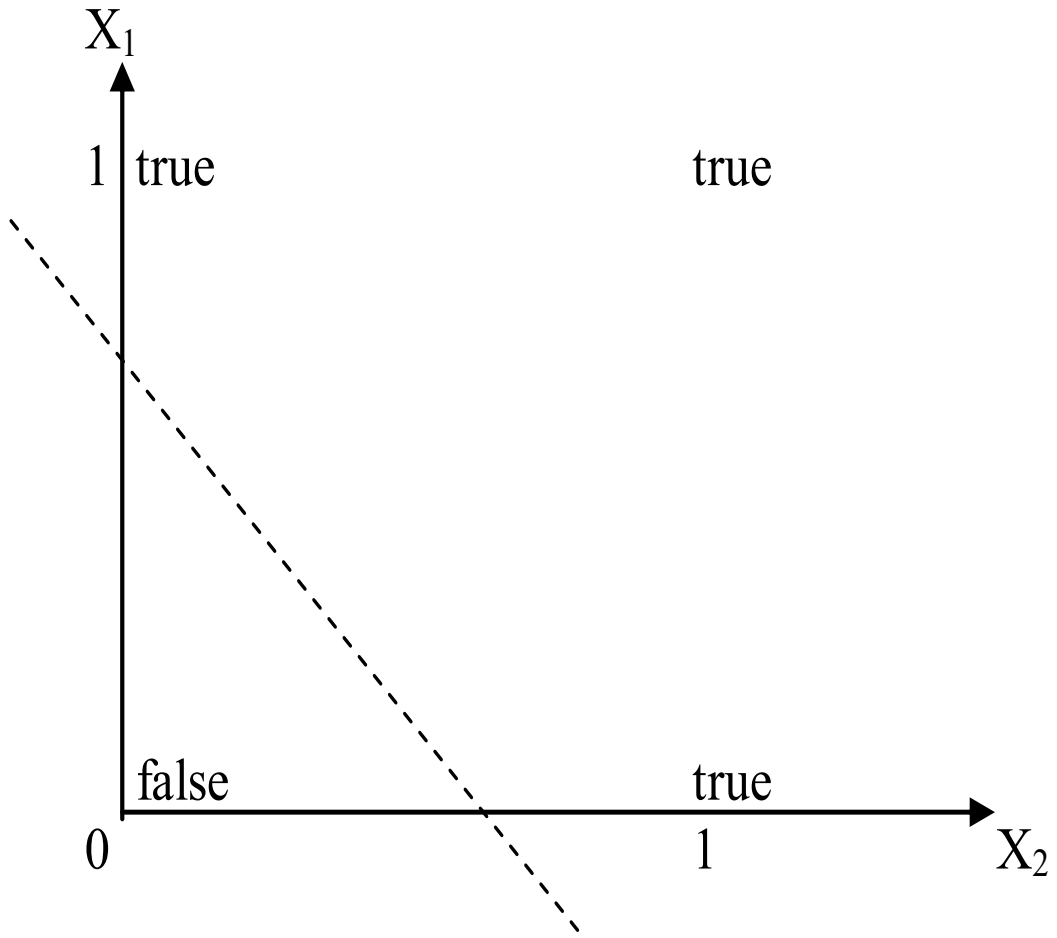


Perceptron for Classification

- used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

- Given training examples of classes C_1, C_2 train the perceptron in such a way that :
  - *If the output of the perceptron is +1 then the input is assigned to class* C_1
  - *If the output is -1 then the input is assigned to* C_2

Boolean function OR – Linearly separable



Perceptron: Limitations

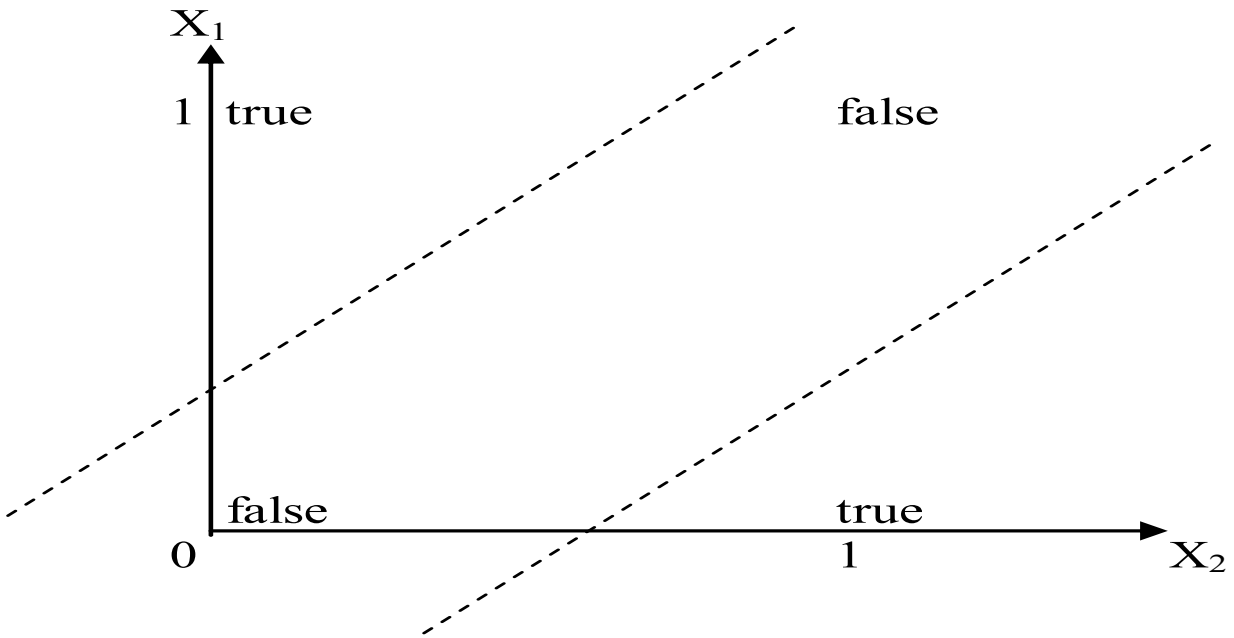
- The perceptron can only model linearly separable functions,
  - those functions which can be drawn in 2-dim graph and single straight line separates values in two part.
- Boolean functions given below are linearly separable:
  - AND
  - OR
  - COMPLEMENT
- It cannot model XOR function as it is non linearly separable.
  - When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

XOR – Non linearly separable function

- A typical example of non-linearly separable function is the XOR that computes the logical **exclusive or.**.
- This function takes two input arguments with values in {0,1} and returns one output in {0,1},
- Here 0 and 1 are encoding of the truth values **false** and **true**,
- The output is **true** if and only if the two inputs have different truth values.
- XOR is non linearly separable function which can not be modeled by perceptron.
- For such functions we have to use multi layer feed-forward network.

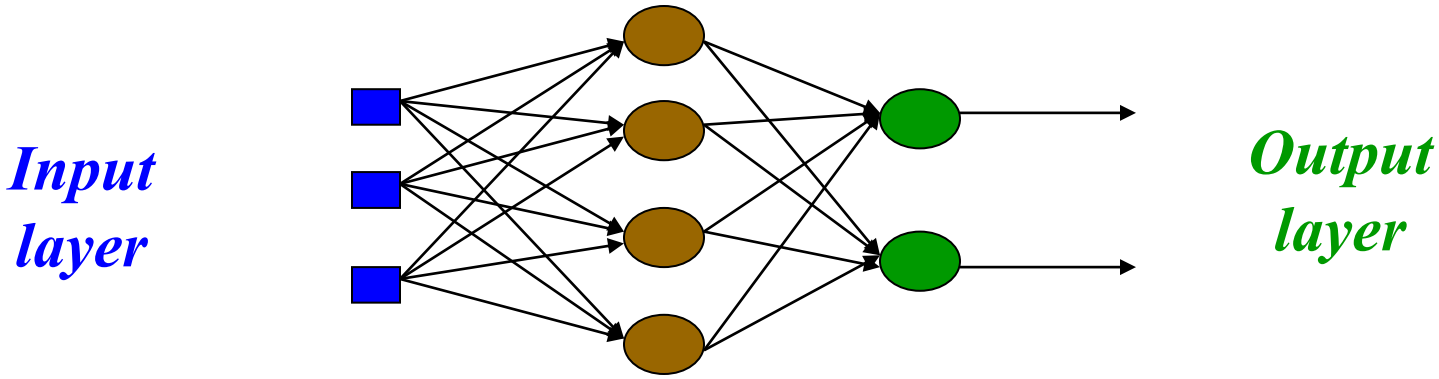
Input		Output
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

These two classes (true and false) cannot be separated using a line. Hence XOR is non linearly separable.



Multi layer feed-forward NN (FFNN)

- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning tasks.



3-4-2 Network

FFNN NEURON MODEL

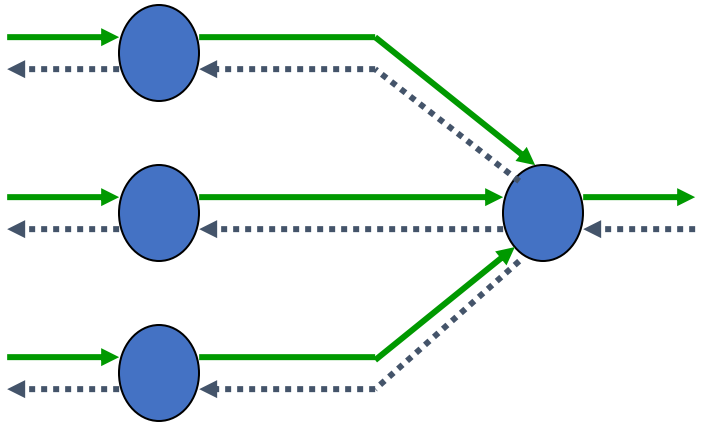
- The classical learning algorithm of FFNN is based on the gradient descent method.
- For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- Eg. **sigmoid function**

Training Algorithm: Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

Backpropagation

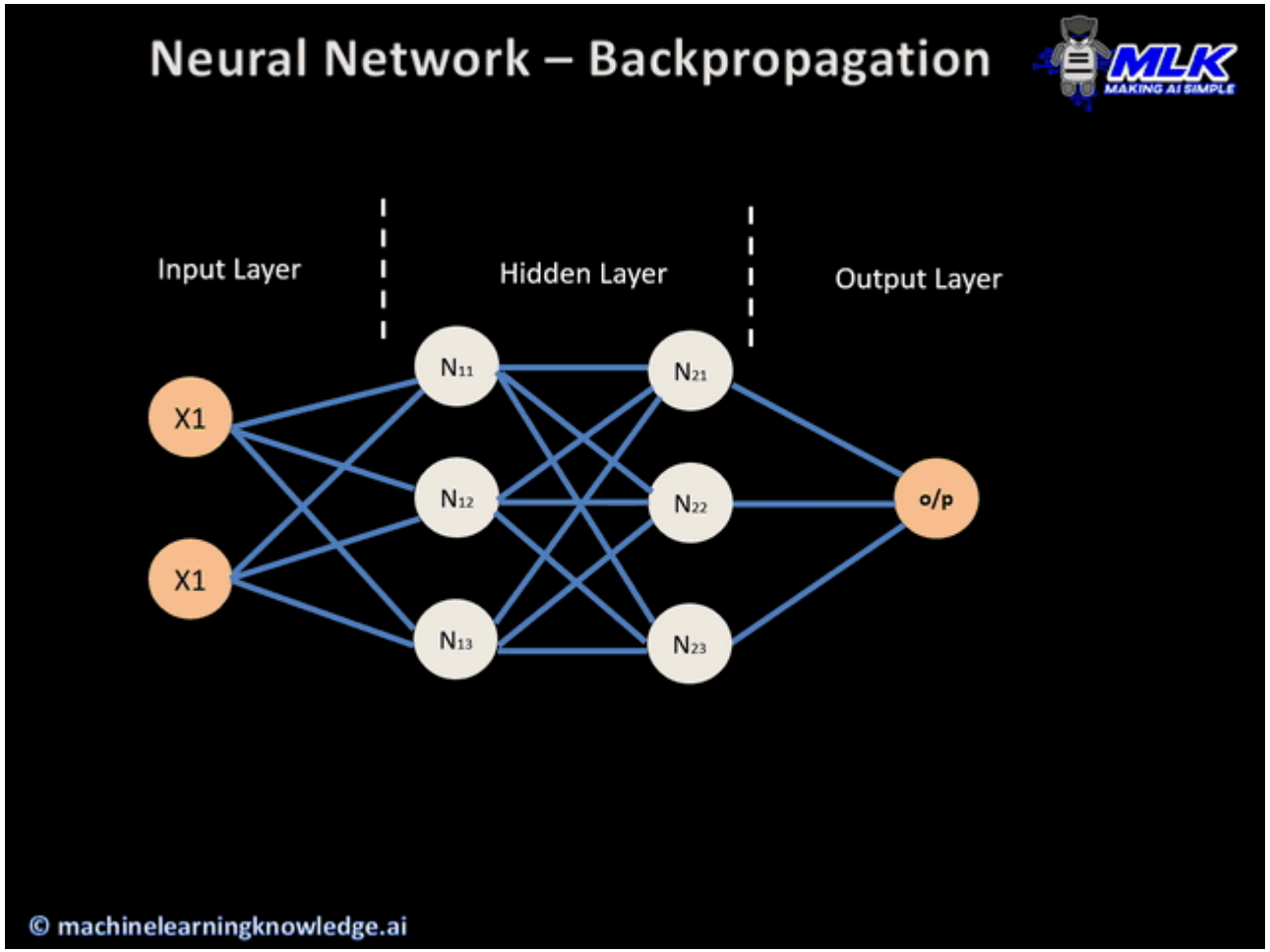
- Back-propagation training algorithm



Network activation
Forward Step

Error propagation
Backward Step

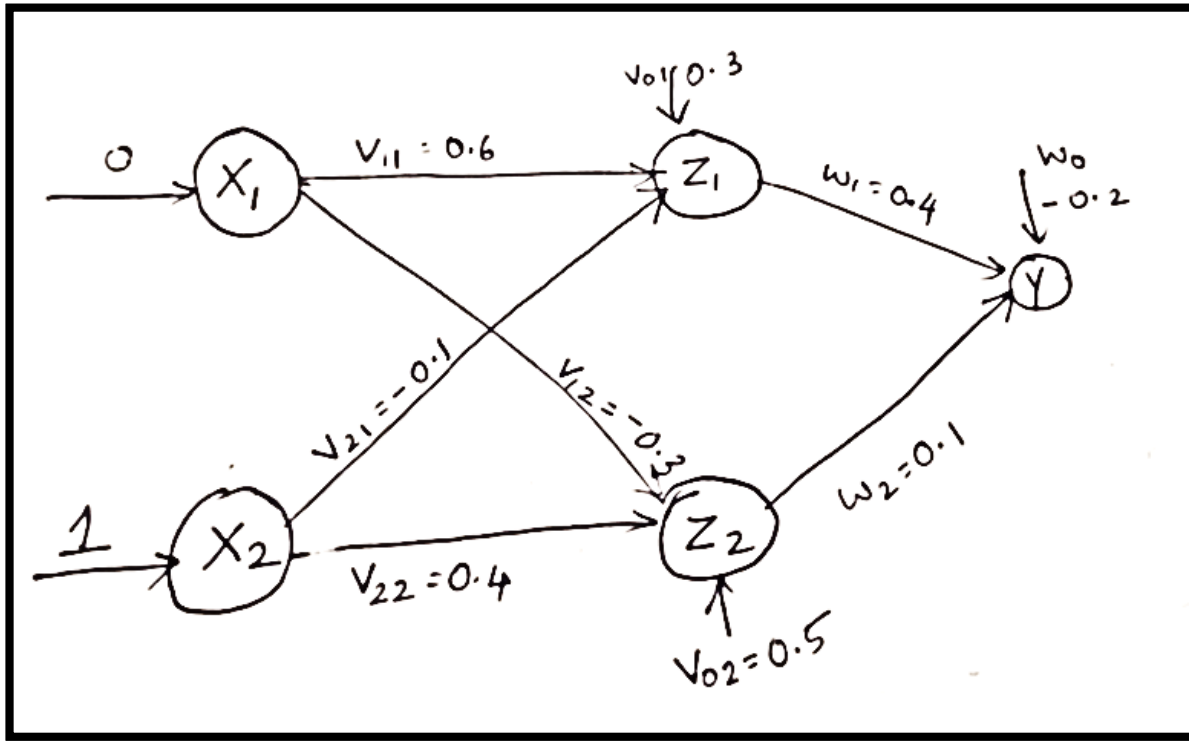
- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.


Neural Network – Backpropagation
MLK
MAKING AI SIMPLE
Input Layer
Hidden Layer
Output Layer
X1
X1
N11
N12
N13
N21
N22
N23
o/p
© machinelearningknowledge.ai

- Using back-propagation network, find the weights for the net. It is presented with the input pattern [0, I] and the target output is I. Use a learning rate α = 0.25 and sigmoidal activation function. initial weights are $[v_{11}\ \ v_{21}\ \ v_{01}]$ = [0.6 -0.1 0.3]

$[v_{12}\ v_{22}\ v_{02}] = [-0.3\ 0.4\ 0.5]$ and $[w_1\ w_2\ w_0] = [0.4$
$0.1\ -0.2]$, and the learning rate is $\alpha = 0.25$. Acti-
vation function used is binary sigmoidal activation
function and is given by

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



Calculate the net input: For z_1 layer

$$z_{in1} = v_{01} + x_1 v_{11} + x_2 v_{21}$$
$$= 0.3 + 0 \times 0.6 + 1 \times -0.1 = 0.2$$

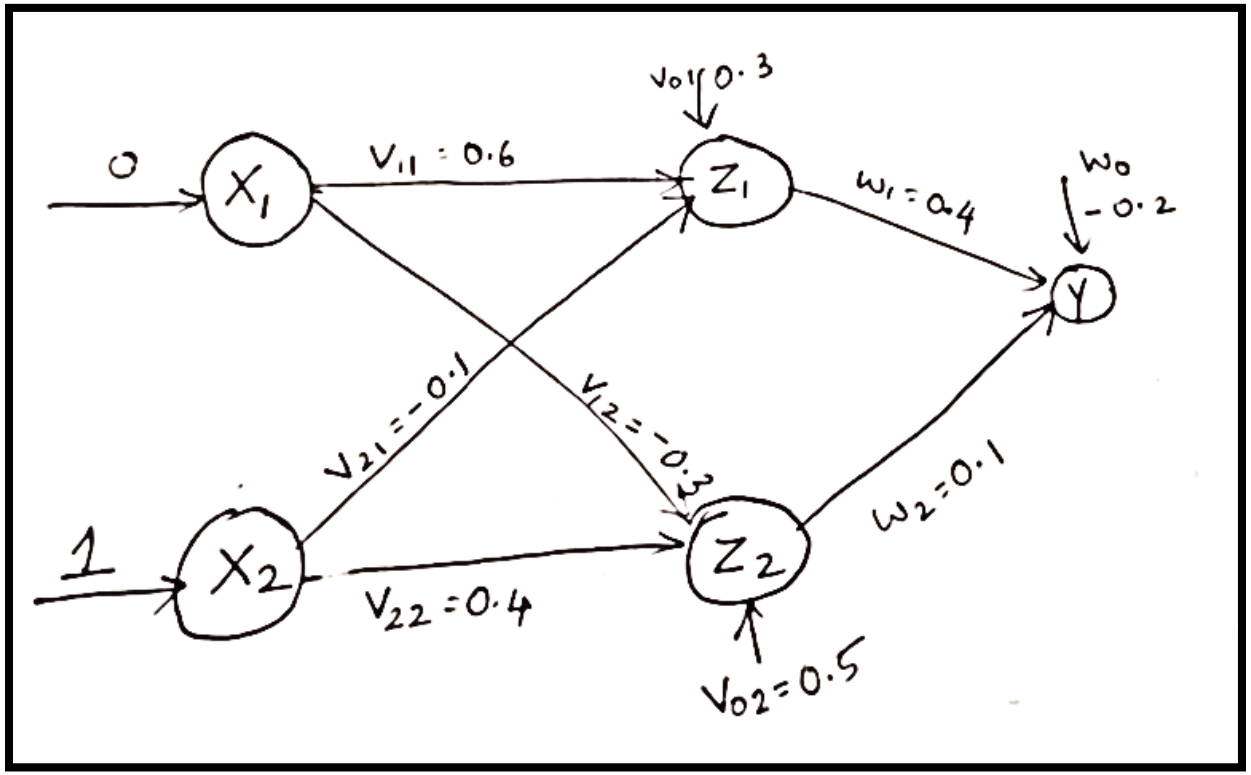
For z_2 layer

$$z_{in2} = v_{02} + x_1 v_{12} + x_2 v_{22}$$
$$= 0.5 + 0 \times -0.3 + 1 \times 0.4 = 0.9$$

Applying activation to calculate the output, we obtain

$$z_1 = f(z_{in1}) = \frac{1}{1 + e^{-z_{in1}}} = \frac{1}{1 + e^{-0.2}} = 0.5498$$

$$z_2 = f(z_{in2}) = \frac{1}{1 + e^{-z_{in2}}} = \frac{1}{1 + e^{-0.9}} = 0.7109$$

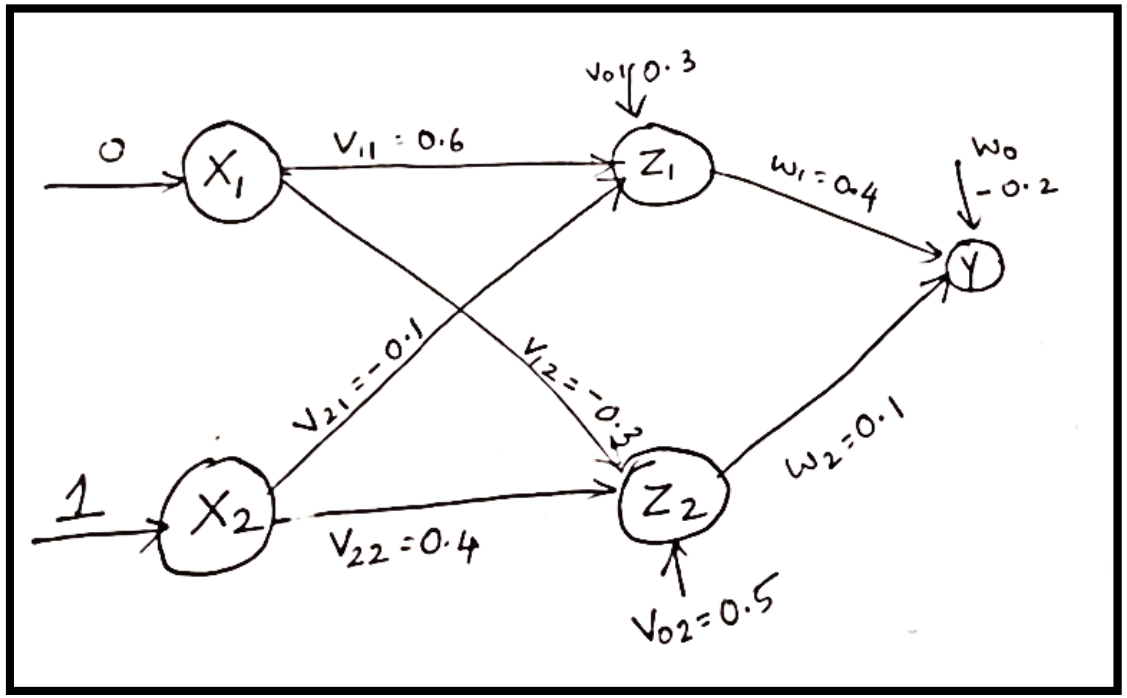


Calculate the net input entering the output layer.
For y layer

$$y_{in} = w_0 + z_1 w_1 + z_2 w_2$$
$$= -0.2 + 0.5498 \times 0.4 + 0.7109 \times 0.1$$
$$= 0.09101$$

Applying activations to calculate the output, we obtain

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-0.09101}} = 0.5227$$



- Compute the error portion δ_k:

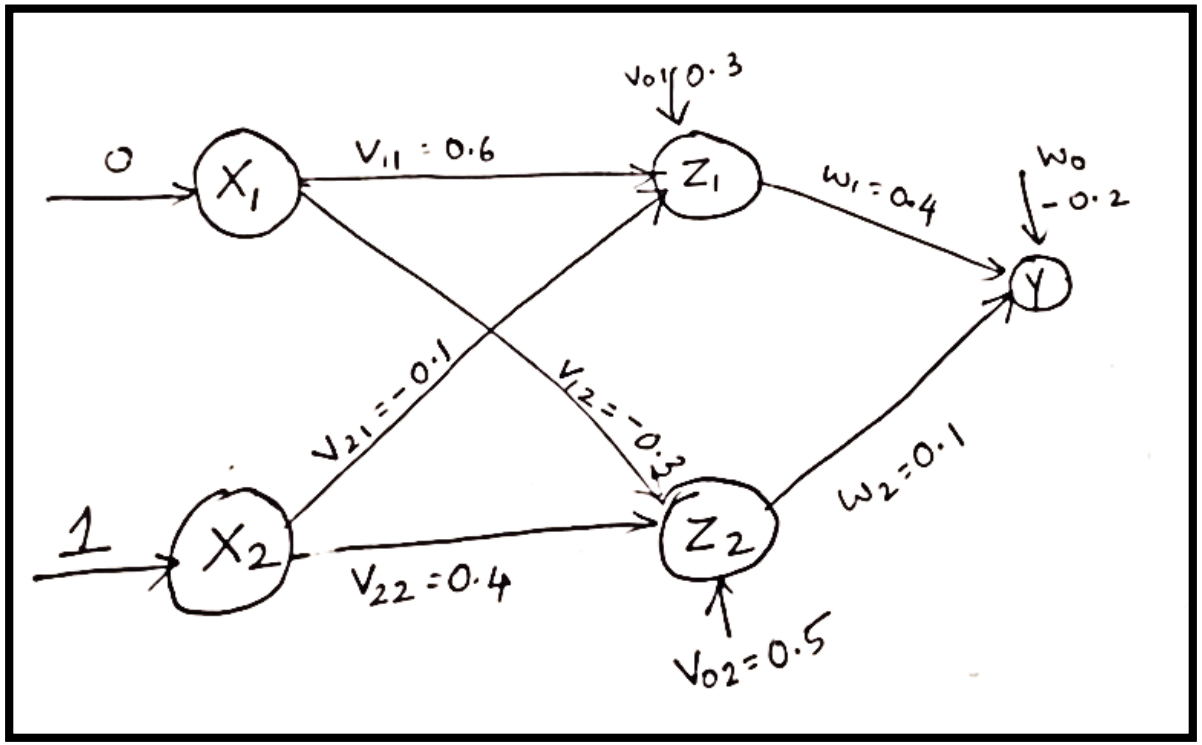
$$\delta_k = (t_k - y_k) f'(y_{ink})$$

Now

$$f'(y_{in}) = f(y_{in})[1 - f(y_{in})] = 0.5227[1 - 0.5227]$$

$$f'(y_{in}) = 0.2495$$

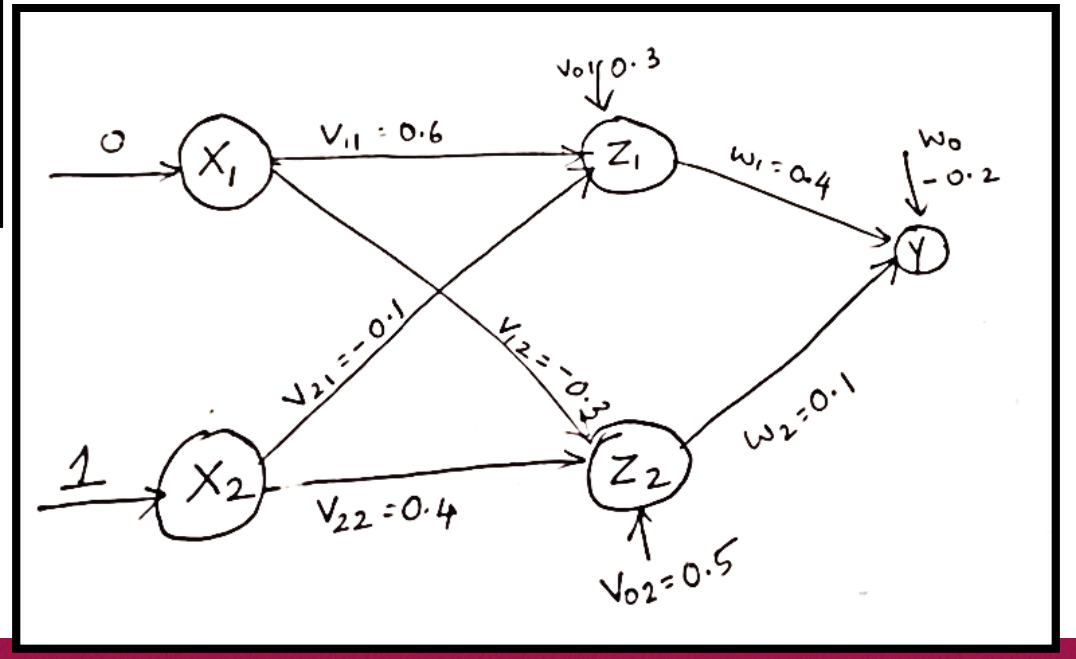
$$\delta_1 = (1 - 0.5227)\,(0.2495) = 0.1191$$



$$\Delta w_1 = \alpha \delta_1 z_1 = 0.25 \times 0.1191 \times 0.5498$$
$$= 0.0164$$

$$\Delta w_2 = \alpha \delta_1 z_2 = 0.25 \times 0.1191 \times 0.7109$$
$$= 0.02117$$

$$\Delta w_0 = \alpha \delta_1 = 0.25 \times 0.1191 = 0.02978$$



- Compute the error portion δ_j between input and hidden layer ($j = 1$ to 2):

$$\delta_j = \delta_{inj} f'(z_{inj})$$

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

$$\delta_{inj} = \delta_1 w_{j1} \quad [\because \text{only one output neuron}]$$

$$\Rightarrow \delta_{in1} = \delta_1 w_{11} = 0.1191 \times 0.4 = 0.04764$$

$$\Rightarrow \delta_{in2} = \delta_1 w_{21} = 0.1191 \times 0.1 = 0.01191$$

Error, $\delta_1 = \delta_{in1} f'(z_{in1})$

$$f'(z_{in1}) = f(z_{in1})[1 - f(z_{in1})]$$
$$= 0.5498[1 - 0.5498] = 0.2475$$
$$\delta_1 = \delta_{in1} f'(z_{in1})$$
$$= 0.04764 \times 0.2475 = 0.0118$$

Error, $\delta_2 = \delta_{in2} f'(z_{in2})$

$$f'(z_{in2}) = f(z_{in2})[1 - f(z_{in2})]$$
$$= 0.7109[1 - 0.7109] = 0.2055$$
$$\delta_2 = \delta_{in2} f'(z_{in2})$$
$$= 0.01191 \times 0.2055 = 0.00245$$

Now find the changes in weights between input and hidden layer:

$$\Delta v_{11} = \alpha \delta_1 x_1 = 0.25 \times 0.0118 \times 0 = 0$$

$$\Delta v_{21} = \alpha \delta_1 x_2 = 0.25 \times 0.0118 \times 1 = 0.00295$$

$$\Delta v_{01} = \alpha \delta_1 = 0.25 \times 0.0118 = 0.00295$$

$$\Delta v_{12} = \alpha \delta_2 x_1 = 0.25 \times 0.00245 \times 0 = 0$$

$$\Delta v_{22} = \alpha \delta_2 x_2 = 0.25 \times 0.00245 \times 1 = 0.0006125$$

$$\Delta v_{02} = \alpha \delta_2 = 0.25 \times 0.00245 = 0.0006125$$

- Compute the final weights of the network:

$$v_{11}(\text{new}) = v_{11}(\text{old}) + \Delta v_{11} = 0.6 + 0 = 0.6$$

$$v_{12}(\text{new}) = v_{12}(\text{old}) + \Delta v_{12} = -0.3 + 0 = -0.3$$

$$v_{21}(\text{new}) = v_{21}(\text{old}) + \Delta v_{21}$$
$$= -0.1 + 0.00295 = -0.09705$$

$$v_{22}(\text{new}) = v_{22}(\text{old}) + \Delta v_{22}$$
$$= 0.4 + 0.0006125 = 0.4006125$$

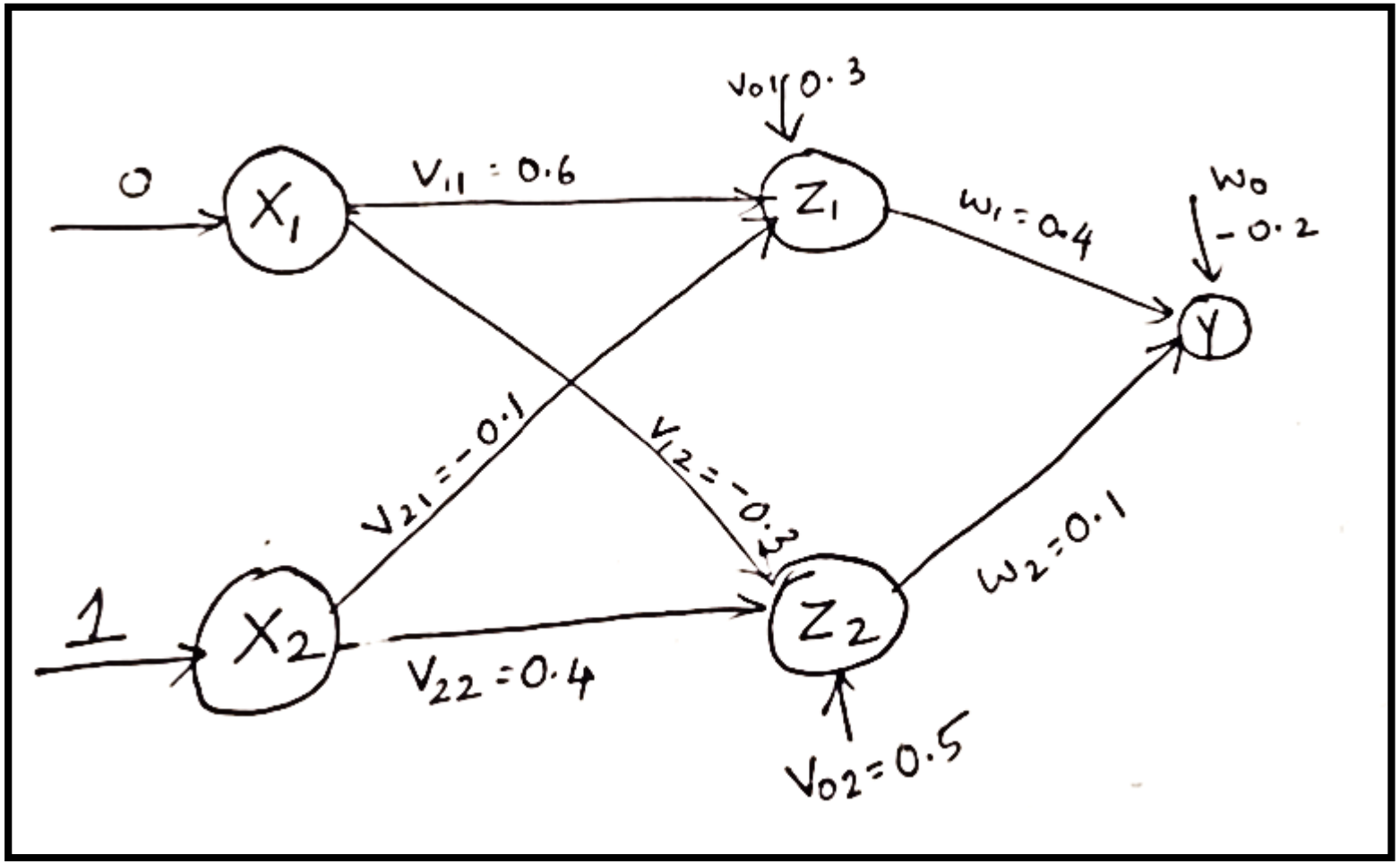
$$w_1(\text{new}) = w_1(\text{old}) + \Delta w_1 = 0.4 + 0.0164$$
$$= 0.4164$$

$$w_2(\text{new}) = w_2(\text{old}) + \Delta w_2 = 0.1 + 0.02117$$
$$= 0.12117$$

$$v_{01}(\text{new}) = v_{01}(\text{old}) + \Delta v_{01} = 0.3 + 0.00295$$
$$= 0.30295$$

$$v_{02}(\text{new}) = v_{02}(\text{old}) + \Delta v_{02}$$
$$= 0.5 + 0.0006125 = 0.5006125$$

$$w_0(\text{new}) = w_0(\text{old}) + \Delta w_0 = -0.2 + 0.02978$$
$$= -0.17022$$



Thank you !!!!!