

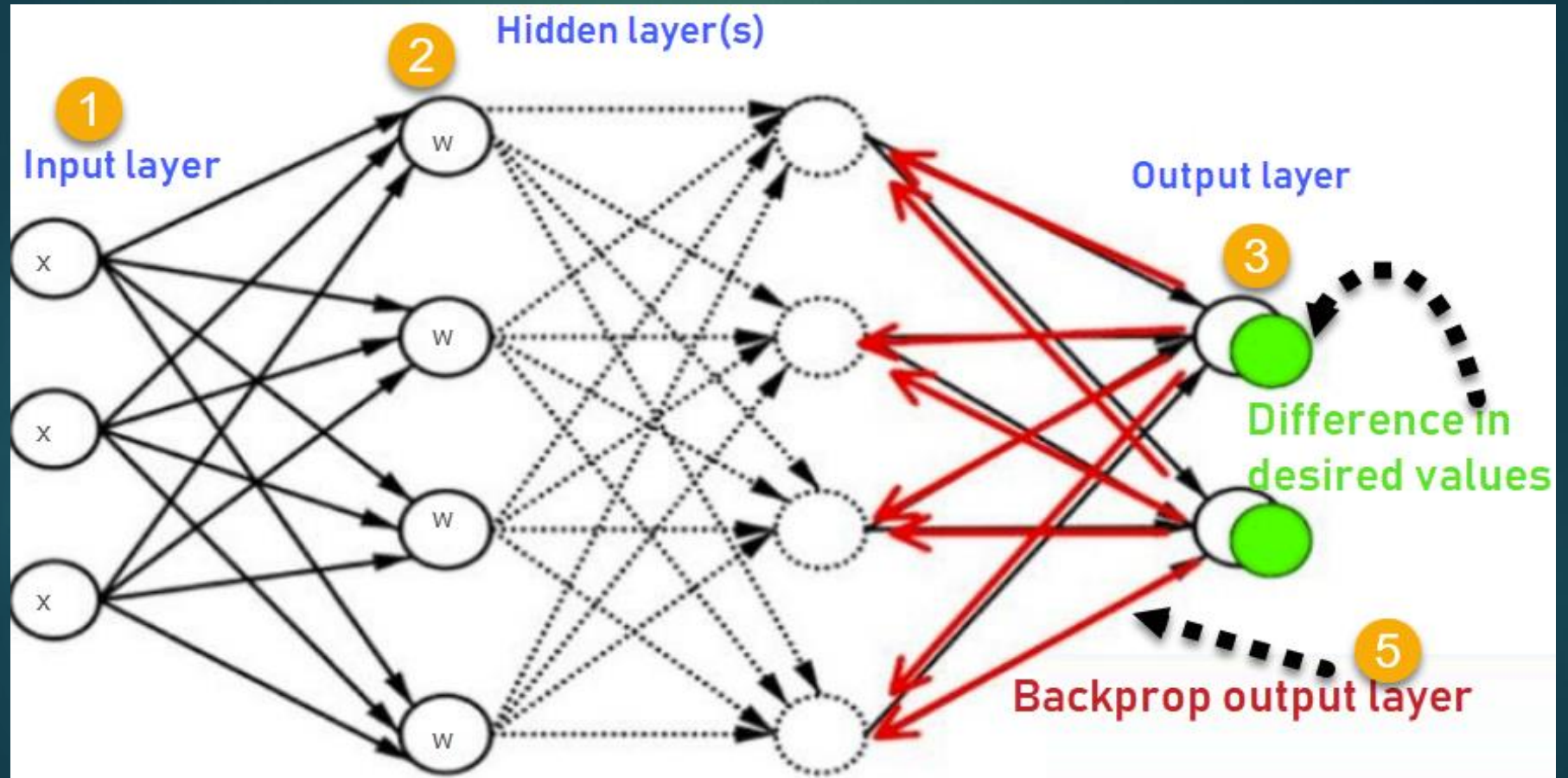


BACK PROPAGATION NETWORK

BY

ASST. PROF. N. L. MUDEGOL

How bPN works?

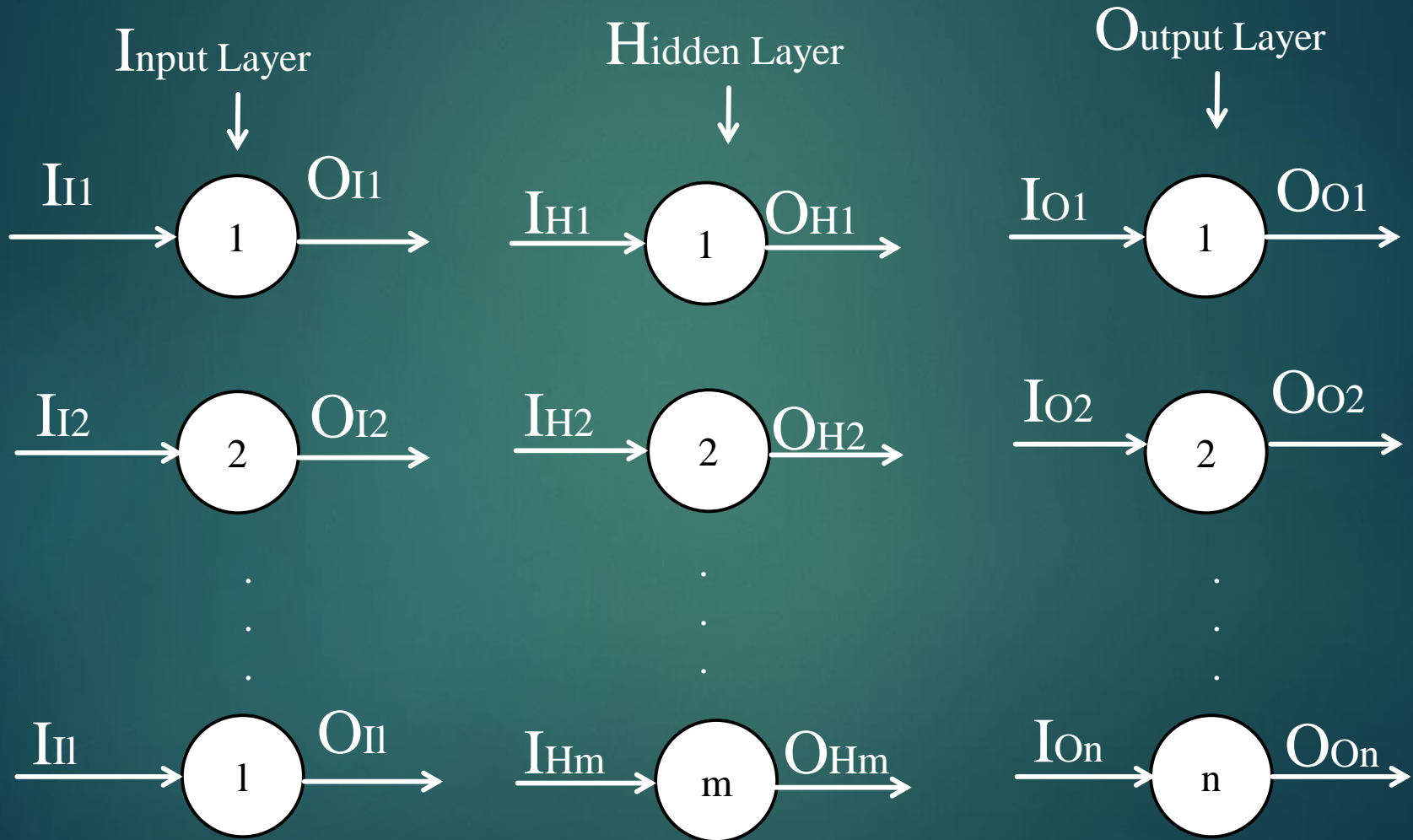


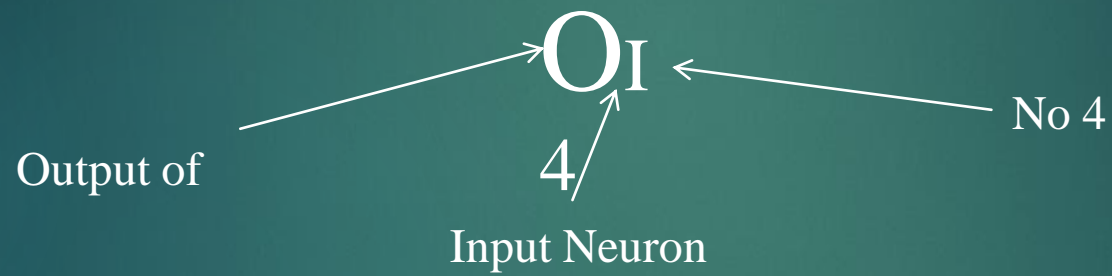
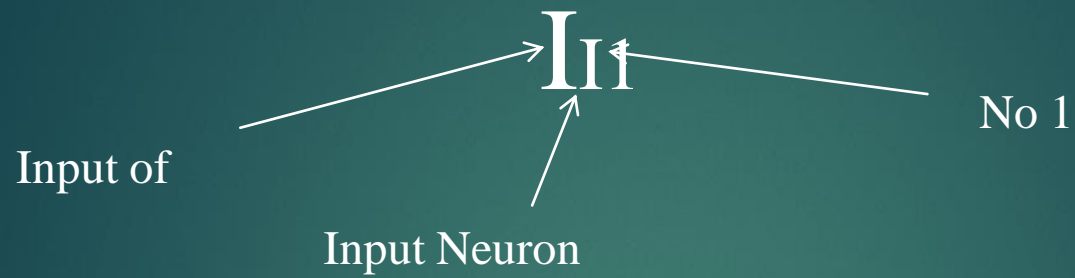
- ▶ inputs x , arrive through the preconnected path
- ▶ input is modeled using real weights W . The weights are usually randomly selected.
- ▶ calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- ▶ calculate the error in the outputs
$$\text{error} = \text{actual output} - \text{desired output}$$
- ▶ travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
- ▶ keep repeating the process until the desired output is achieved

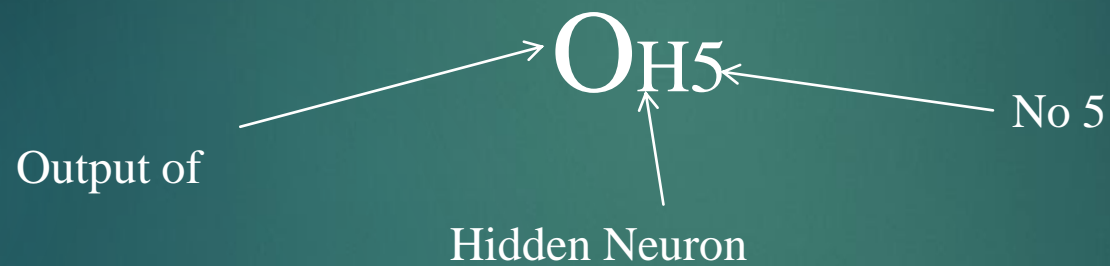
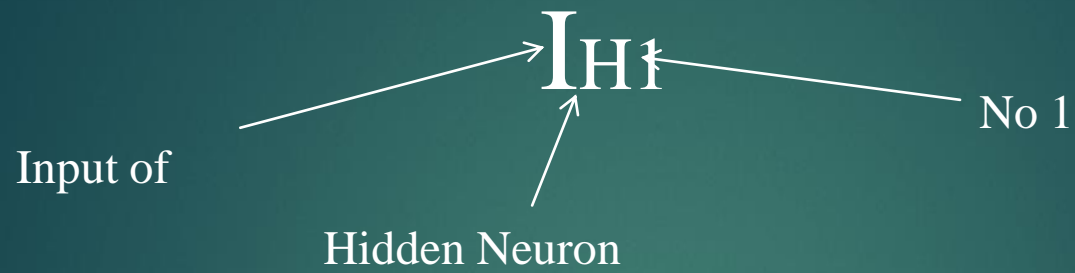
Why We Need Back-propagation?

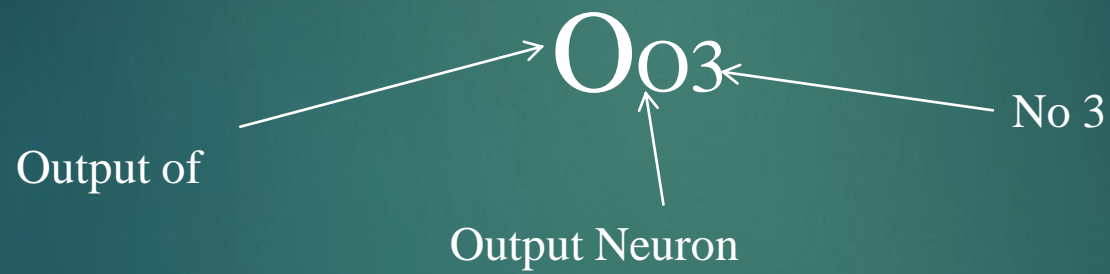
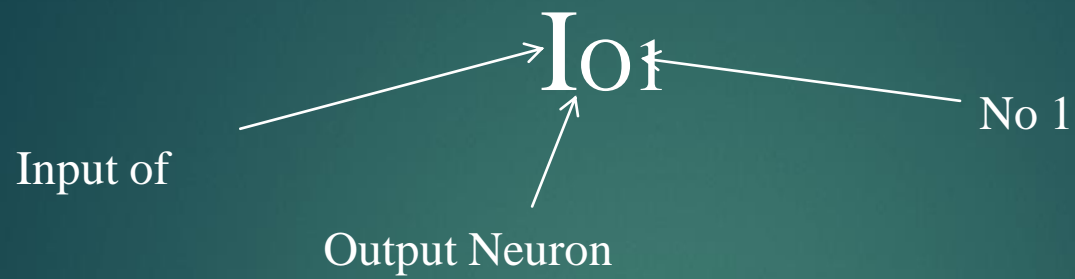
- ▶ It is fast, simple and easy to program
- ▶ It has no parameters to tune apart from the numbers of input
- ▶ It is a flexible method as it does not require prior knowledge about the network
- ▶ It is a standard method that generally works well
- ▶ It does not need any special mention of the features of the function to be learned.
- ▶ It is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.
- ▶ It takes advantage of the chain and power rules allows it to function with any number of outputs.

MULTI LAYER FEED FORWARD NETWORK (BPN)

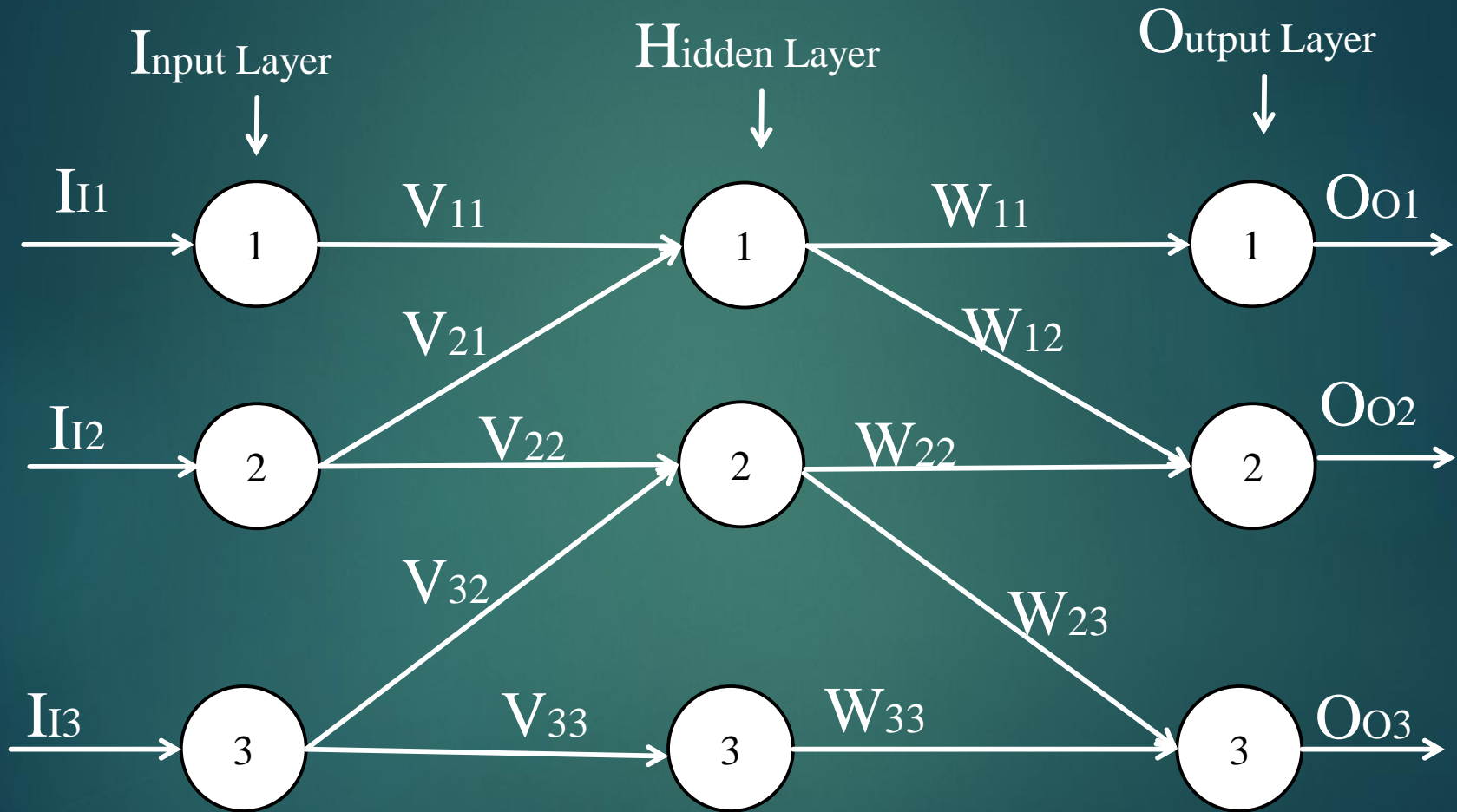




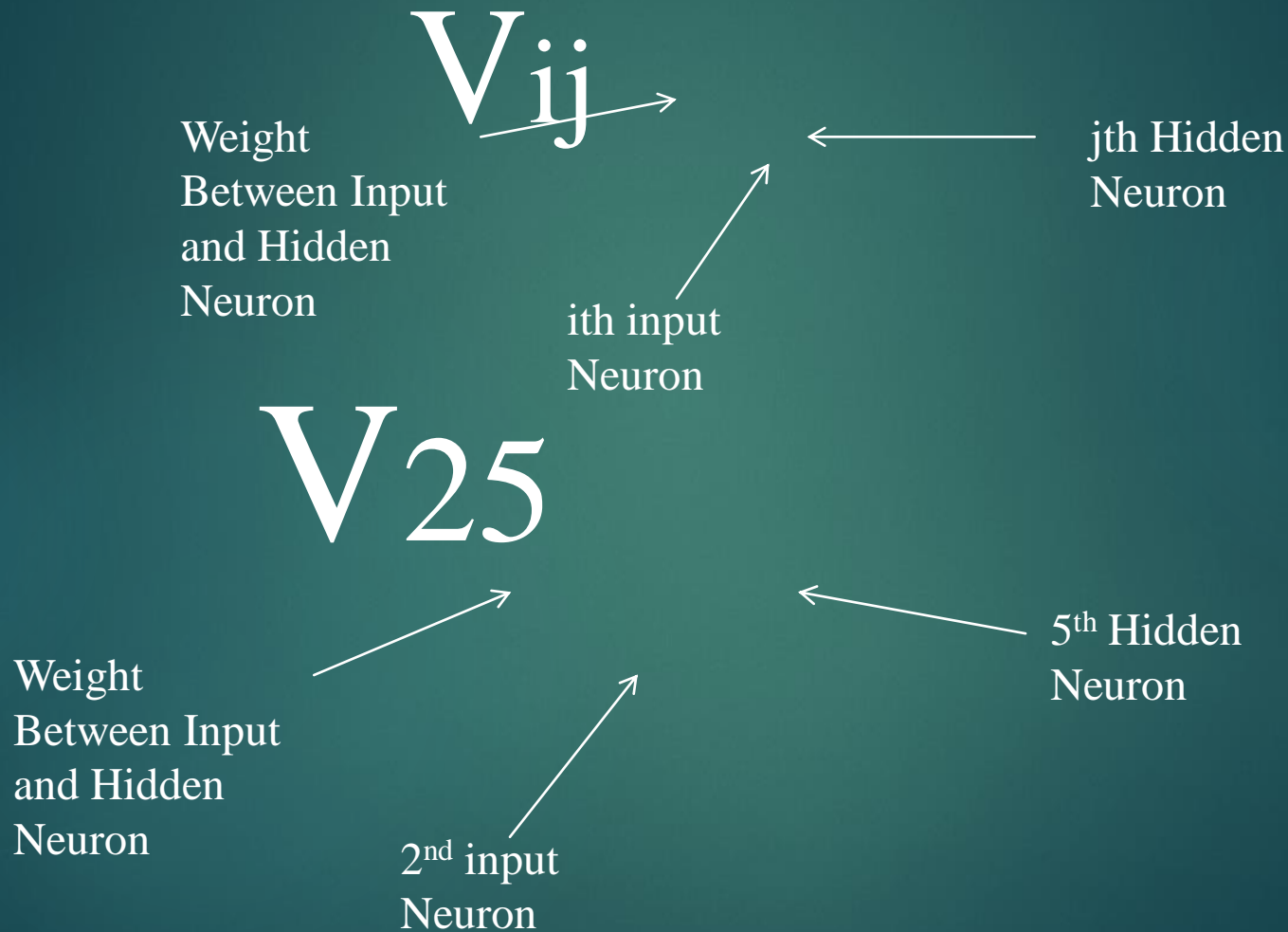




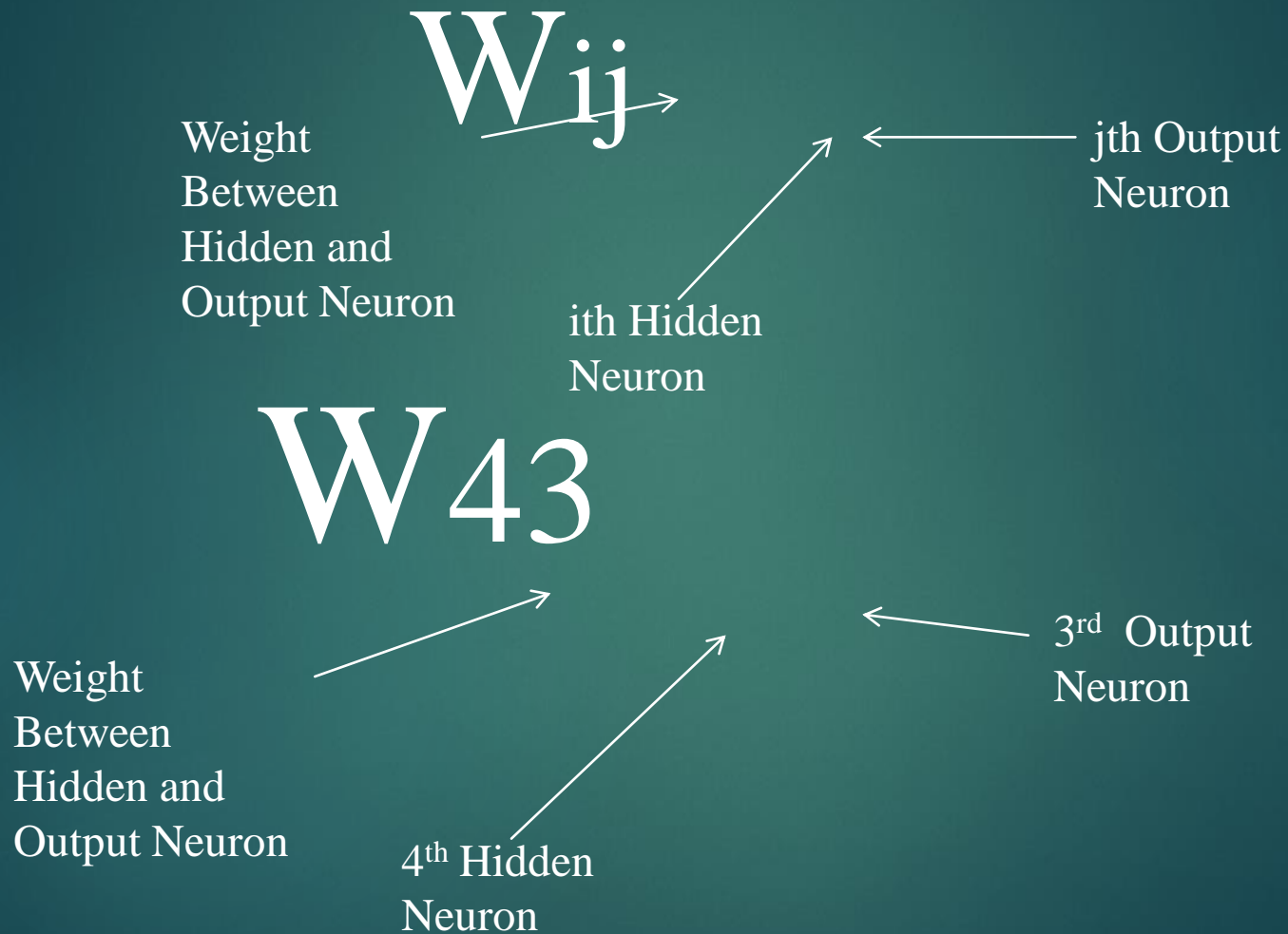
MULTI LAYER FEED FORWARD NETWORK (BPN)



V is the weight on the link from i th input neuron to j th hidden neuron.



W is the weight on the link from i th Hidden neuron to j th Output neuron.



INTRODUCTION



▶ THREE PHASES:

▶ FEED FORWARD (Forward Pass):

- ▶ Purpose is to propagate our inputs through the network by applying a series of dot products and activations until we reach the output layer of the network.
- ▶ Also known as propagation phase

▶ BACK PROPAGATION OF ERROR:

- ▶ The backward pass where we compute the gradient of the loss function at the final layer of the network.

▶ WEIGHT UPDATE:

- ▶ Use the gradient (error) calculated in backward pass to recursively apply the chain rule to update the weights in our network.



STEP 1:

Initialize weights and learning rate to a small **random** values.

STEP 2:

Perform steps 3-17 until stopping condition is false.

STEP 3:

Perform steps 4-16 for each training pair.



PHASE – 1

FEED-FORWARD

PHASE-1: FEED-FORWARD PHASE

STEP 4: OUTPUT OF INPUT LAYER

Present each input pair to input layer. Apply **linear activation function** to it.

i.e. $\phi(I) = x$ for all x

So, output of input layer will be same as it's input.

i.e. $\{O\} = \{I\}$ for all x

PHASE-1: FEED-FORWARD PHASE

STEP 5: INPUT FOR HIDDEN LAYER

Compute the **net Input** for each hidden neuron.

$$I_{Hp} = \sum_{i=1}^l V_{ij} O_{I_i}$$

j=p and p = 1, 2, ..., m

(Here Input and Output is same for input layer, so we can take either of two i.e. input or output for net input computation)

Add Bias in net Input.

$$I_{Hp} = I_{Hp} + b_{Hp}$$

Note: Bias can be same for all inputs or different for each hidden layer.

PHASE-1: FEED-FORWARD PHASE

STEP 6: OUTPUT OF HIDDEN LAYER

Output of hidden layer can be computed by applying **binary sigmoidal function** on it's input.

$$O_{Hp} = \frac{1}{1 + e^{-I_{Hp}}}$$
$$p = 1, 2, \dots, m$$

We shall have 'm' such outputs which will be passed to the output layer.

PHASE-1: FEED-FORWARD PHASE

STEP 7: INPUT FOR OUTPUT LAYER

Compute the **net Input** for each output neuron.

$$I_{O_q} = \sum_{i=1}^n W_{ij} O_{H_i}$$

$j = q \text{ and } q = 1, 2, \dots, n$

Add Bias in net Input.

$$I_{O_q} = I_{O_q} + b_{O_q}$$

Note: Bias can be same for all inputs or different for each hidden layer.

PHASE-1: FEED-FORWARD PHASE

STEP 8: OUTPUT OF OUTPUT LAYER

Output of output layer can be computed by applying **binary sigmoidal function** on it's input.

$$O_{o_q} = \frac{1}{1 + e^{-I_{o_q}}}$$

We shall have 'n' such outputs which will be passed to the output layer.

(In case of logic gates $n = 1$)



PHASE – 2

ERROR BACK-PROPAGATION

PHASE-2: ERROR BACK-PROPAGATION PHASE

STEP 9: ERROR CALCULATION

For given training pair, calculate the difference between desired output and actual output.

$$e_r = (O_{target_r} - O_{actual_r})$$

$$r = 1, 2, \dots, n$$

PHASE-2: ERROR BACK-PROPAGATION PHASE

STEP 10: CALCULATION OF DERIVATIVE OF OUTPUT

Calculate the derivative of the output of output layer. In order to apply the backpropagation algorithm, our activation function must be differentiable so that we can compute the partial derivative of the error with respect to a given weight

$$d_{O_r} = (1 - O_{O_r})(O_{O_r})$$

$$r = 1, 2, \dots, n$$

Note: This is a derivative of binary sigmoidal function.

We shall have 'n' such derivatives.

PHASE-2: ERROR BACK-PROPAGATION PHASE

STEP 11: CALCULATION OF ERROR CORRECTION TERM

Calculate the error correction term.

$$\begin{aligned} E_{O_r} &= e_r d_{O_r} \\ &= (O_{target_r} - O_{actual_r})(1 - O_{O_r})(O_{O_r}) \end{aligned}$$

$$r = 1, 2, \dots, n$$

We shall have 'n' such error correction terms.

PHASE-2: ERROR BACK-PROPAGATION PHASE

STEP 12: CHANGE IN WEIGHTS AND BIAS

Calculate change in weights and bias between hidden and output layer.

$$\Delta W_{jk} = \alpha * E_{O_k} * O_{H_j}$$

And

$$\Delta b_{O_j} = \alpha * E_{O_j}$$

Where,

α = Learning rate

PHASE-2: ERROR BACK-PROPAGATION PHASE

STEP 13: DELTA INPUTS TO HIDDEN LAYER

Compute the delta inputs to every hidden layer.

$$\delta_{H_j} = E_{O_k} * W_{jk} * d_{H_j}$$

Where,

$$d_{H_j} = (1 - O_{H_j})O_{H_j}$$

We shall have 'm' such a delta inputs which will be passed to hidden layer.

PHASE-2: ERROR BACK-PROPAGATION PHASE

STEP 14: CHANGE IN WEIGHTS AND BIAS

Calculate change in weights and bias between input and hidden layer.

$$\Delta V_{jk} = \alpha * \delta_{H_k} * O_{I_j}$$

And

$$\Delta b_{H_j} = \alpha * \delta_{H_j}$$

Where,

α = Learning rate



PHASE – 3

WEIGHT AND BIAS UPDATE

PHASE-3: WEIGHT UPDATION PHASE

STEP 15: UPDATE WEIGHTS AND BIAS

Update the weights and bias between hidden and output layer.

$$W_{jk(new)} = W_{jk(old)} + \Delta W_{jk}$$

And

$$b_{o_j(new)} = b_{o_j(old)} + \Delta b_{o_j}$$

PHASE-3: WEIGHT UPDATION PHASE

STEP 16: UPDATE WEIGHTS AND BIAS

Update the weights and bias between input and hidden layer.

$$V_{jk(new)} = V_{jk(old)} + \Delta V_{jk}$$

And

$$b_{H_j(new)} = b_{H_j(old)} + \Delta b_{H_j}$$

PHASE-3: WEIGHT UPDATION PHASE

STEP 17:

Check for the stopping condition.

A stopping condition may be certain number of epochs (iterations) or when actual output matches with the targets output.

STOPPING CONDITION:

Compute the Error Norm as follows:

$$E_{Nk} = 1/2 (e_k^2)$$

For every training pair,

$$E_{N_1}^n = \frac{(\sum_1^n e_n^2)}{n} \quad n = \text{total outputs}$$

Also called as '**Mean-squared Error**'

If E is acceptably low than tolerance, then stop.

STOPPING CONDITION:

The Euclidean norm of Error:

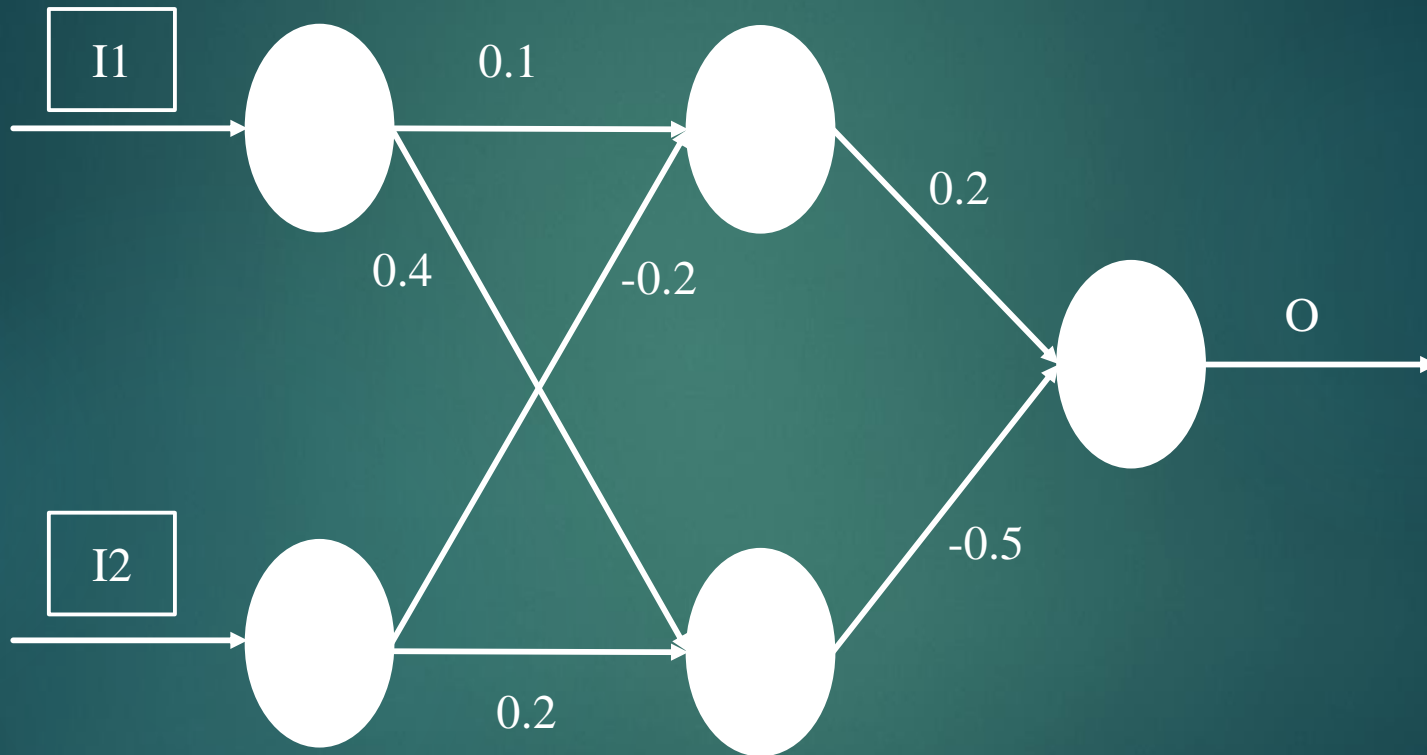
$$E_{N_1}^n = \frac{\sqrt{(\sum_1^n e_n^2)}}{n}$$

n = total outputs

SOLVED EXAMPLE:

SR. NO.	INPUTS		OUTPUT
	I1	I2	O
1	0.4	-0.7	0.1
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12

SOLVED EXAMPLE:





Given:

$$I_1 = 0.4, \quad I_2 = -0.7$$

$$V_{11} = 0.1, \quad V_{12} = 0.4, \quad V_{21} = -0.2, \quad V_{22} = 0.2$$

$$W_{11} = 0.2, \quad W_{21} = -0.5$$

$$a = 1$$

1. Output of Input Layer will be same as its Input because of Linear Activation Function.

$$I_{I1} = 0.4, \quad I_{I2} = -0.7$$

$$O_{I1} = 0.4 \quad O_{I2} = -0.7$$

2. Input to Hidden Layer

$$\begin{aligned} I_{H1} &= V_{11} * I_1 + V_{21} * I_2 \\ &= (0.1 * 0.4) + (-0.2 * -0.7) \\ &= 0.04 + 0.14 \end{aligned}$$

$$I_{H1} = \mathbf{0.18}$$

$$\begin{aligned} I_{H2} &= V_{12} * I_1 + V_{22} * I_2 \\ &= (0.4 * 0.4) + (0.2 * -0.7) \\ &= 0.16 - 0.14 \end{aligned}$$

$$I_{H2} = \mathbf{0.02}$$

3. Output of Hidden Layer

$$\begin{aligned} O_{H1} &= 1 / (1 + e^{-0.18}) \\ &= 0.5448 \end{aligned}$$

$$\begin{aligned} O_{H2} &= 1 / (1 + e^{-0.02}) \\ &= 0.505 \end{aligned}$$

4. Input to Output Layer

$$\begin{aligned} I_{O1} &= W_{11} * O_{H1} + W_{21} * O_{H2} \\ &= (0.2 * 0.5448) + (-0.5 * 0.505) \\ &= 0.10896 - 0.2525 \end{aligned}$$

$$I_{O1} = \mathbf{-0.14354}$$

5. Output of Output Layer

$$O_{O1} = 1 / (1 + e^{0.14354})$$

$$O_{O1} = \mathbf{0.4642}$$

6. Error

$$e = (0.1 - 0.4642)$$

$$e = \mathbf{-0.3642}$$

7. Derivative of Output of Output layer

$$d = (1 - O_{o1}) * O_{o1}$$

$$= (1 - 0.4642) * 0.4642$$

$$= 0.5358 * 0.4642$$

$$\mathbf{d = 0.2487}$$

8. Error Correction Term

$$E = e * d$$

$$= -0.3642 * 0.2487$$

$$\mathbf{E = -0.09058}$$

9. Change in weights of H-O layer

$$\Delta W_{11} = a * E * O_{H1}$$

$$= 1 * (-0.09258) * 0.5448$$

$$\Delta W_{11} = \mathbf{-0.0493}$$

$$\Delta W_{21} = a * E * O_{H2}$$

$$= 1 * (-0.09258) * 0.505$$

$$\Delta W_{21} = \mathbf{-0.0457}$$

10. Derivative of output of Hidden Layer

$$\begin{aligned}d_{H1} &= (1 - O_{H1}) * O_{H1} \\&= (1-0.5448) * 0.5448 \\&= 0.4552 * 0.5448\end{aligned}$$

$$\mathbf{d_{H1} = 0.2479}$$

$$\begin{aligned}d_{H2} &= (1 - O_{H2}) * O_{H2} \\&= (1-0.505) * 0.505 \\&= 0.495 * 0.505\end{aligned}$$

$$\mathbf{d_{H2} = 0.2499}$$

10. Delta inputs to Hidden Layer

$$\begin{aligned}d_1 &= d_{H1} * W_{11} * E \\&= 0.2479 * 0.2 * -0.09058\end{aligned}$$

$$\mathbf{d_1 = -0.00449}$$

$$\begin{aligned}d_2 &= d_{H2} * W_{21} * E \\&= 0.2499 * -0.5 * -0.09058\end{aligned}$$

$$\mathbf{d_2 = 0.01131}$$


11. Change in weights of I-H layer

$$\begin{aligned}\Delta V_{11} &= a * d_1 * O_{11} \\ &= 1 * (-0.00449) * 0.4\end{aligned}$$

$$\Delta V_{11} = \mathbf{-0.001796}$$

$$\begin{aligned}\Delta V_{21} &= a * d_1 * O_{12} \\ &= 1 * (-0.00449) * (-0.7)\end{aligned}$$

$$\Delta V_{12} = \mathbf{0.003143}$$


$$\Delta V_{12} = a * d_2 * O_{11}$$
$$= 1 * (0.01131) * 0.4$$

$$\Delta V_{21} = \mathbf{0.004524}$$

$$\Delta V_{22} = a * d_2 * O_{12}$$
$$= 1 * (0.01131) * (-0.7)$$

$$\Delta V_{22} = \mathbf{-0.007917}$$

12. Weight Update (H-O layer)

$$\begin{aligned} W_{11} &= W_{11} + \Delta W_{11} \\ &= 0.2 - 0.0493 \end{aligned}$$

$$W_{11} = \mathbf{0.1507}$$

$$\begin{aligned} W_{21} &= W_{21} + \Delta W_{21} \\ &= -0.5 - 0.0457 \end{aligned}$$

$$W_{21} = \mathbf{-0.5457}$$

12. Weight Update (I-H layer)

$$\begin{aligned}V_{11} &= V_{11} + \Delta V_{11} \\ &= 0.1 - 0.001796\end{aligned}$$

$$\mathbf{V_{11} = 0.098204}$$

$$\begin{aligned}V_{12} &= V_{12} + \Delta V_{12} \\ &= 0.4 + 0.004524\end{aligned}$$

$$\mathbf{V_{12} = 0.404524}$$

12. Weight Update (I-H layer)

$$\begin{aligned}V_{21} &= V_{21} + \Delta V_{21} \\ &= -0.2 + 0.003143\end{aligned}$$

$$\mathbf{V_{21} = -0.196857}$$

$$\begin{aligned}V_{22} &= V_{22} + \Delta V_{22} \\ &= 0.2 - 0.007917\end{aligned}$$

$$\mathbf{V_{22} = 0.192083}$$

STOPPING CONDITION:

$$\begin{aligned} E_N &= \frac{1}{2} * e * e \\ &= \frac{1}{2} * (-0.3642) * (-0.3642) \\ &= \frac{1}{2} * 0.13264 \end{aligned}$$

$$E_N = \mathbf{0.06632}$$

ADDING MOMENTUM IN WEIGHTS

To improve the rate of convergence, we can add momentum term in weights.

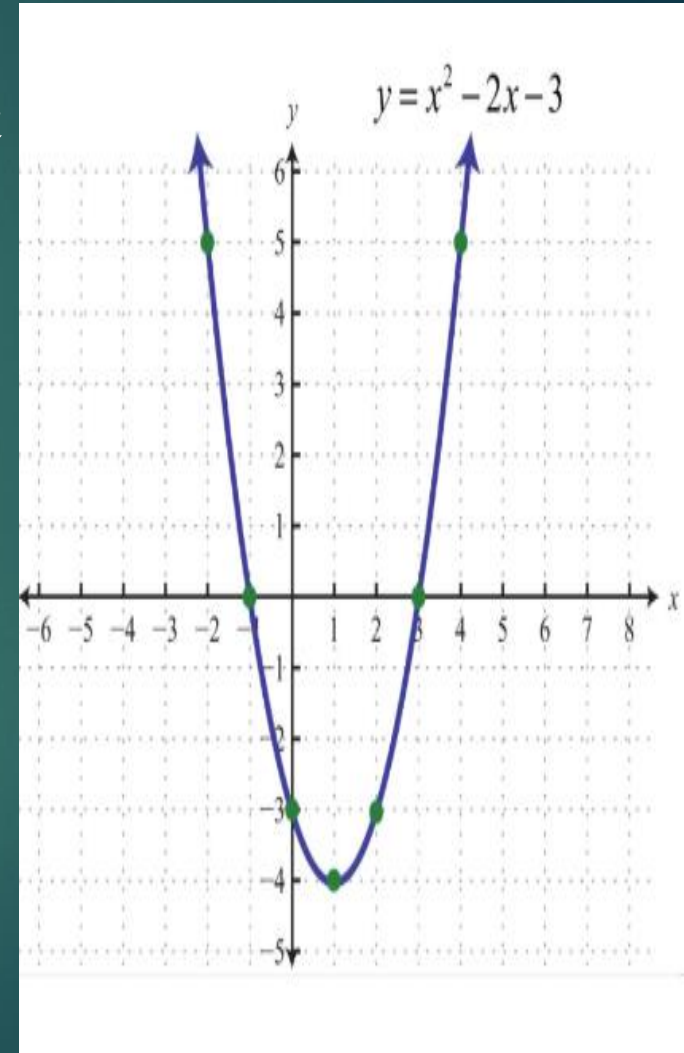
$$\Delta W_{jk(new)} = m * \Delta W_{jk(old)} + \alpha(E_{O_k} * O_{H_j})$$

And

$$\Delta V_{jk(new)} = m * \Delta V_{jk(old)} + \alpha(\delta_{H_k} * O_{I_j})$$

Gradient Descent

- ▶ Gradient, in plain terms means slope or slant of a surface. So gradient descent literally means descending a slope to reach the lowest point on that surface. Let us imagine a two dimensional graph, such as a parabola in the figure.
- ▶ In the graph, the lowest point on the parabola occurs at $x = 1$.
- ▶ The objective of gradient descent algorithm is to find the value of “x” such that “y” is minimum.
- ▶ “y” here is termed as the objective function that the gradient descent algorithm operates upon, to descend to the lowest point.



Gradient Descent

- ▶ Stochastic Gradient Descent: Completing one pass of forward and back propagation using a single example at a time from training set.
- ▶ EPOCH: Using entire dataset once in performing forward and back propagation
- ▶ Batch Gradient Descent: inputting the entire dataset in our forward pass and updating weights and biases using back propagation.
- ▶ Mini Batch Gradient Descent: mini batches of entire dataset are given as input to network.