Artificial Neural Networks

Outline

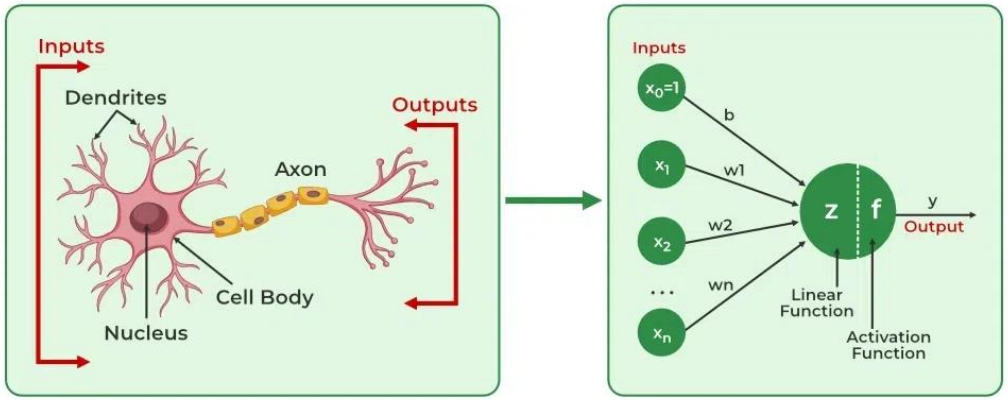
- Why study ANN?
- Basic Neural Network Unit
- Variants

Why neural networks?

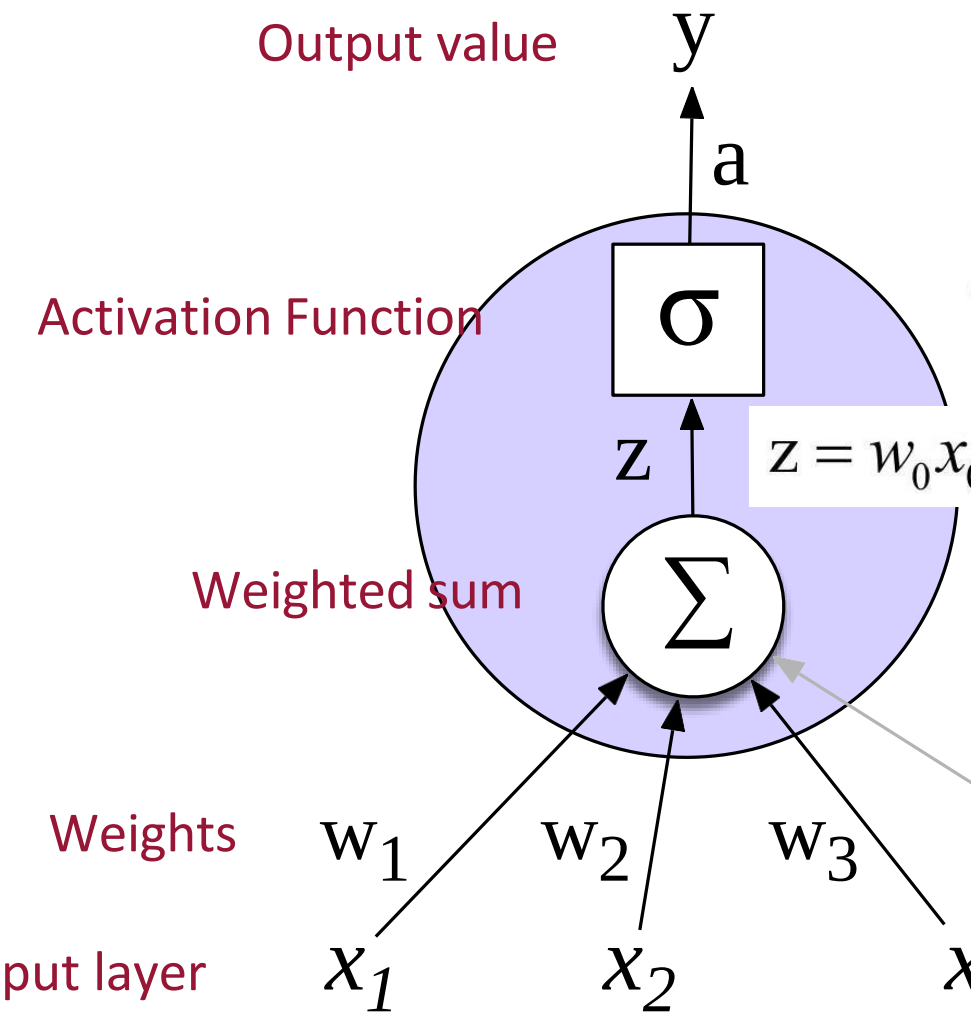
- For certain types of problems, such as learning to interpret complex real-world data, artificial neural networks are among the most effective learning methods currently known.
  - Ex: ANNhas proven surprisingly successful in many practical problems such as
    - handwritten character recognition
    - Speech recognition
    - Face recognition
    - Self-driving car trajectory prediction
    - Email spam filtering
    - Medical diagnosis
    - Applications in computational biology- 3D Protein structure prediction- Alpha fold

Artificial Neural Networks

- Artificial Neural Networks (ANN) are machine learning models that tries to mimic the complex functions of the human brain.
- It consists of interconnected nodes (neurons) organized into layers.
- Information flows through these nodes, and the network adjusts the connection strengths (weights) during training to learn from data, enabling it to recognize patterns, make predictions, and solve various tasks in machine learning and artificial intelligence.



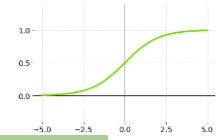
Basic Neural Network Unit: Neuron



Step

$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$



Activation Functions

Sigmoid (0-1)	$s(z) = \frac{1}{1+e^{-z}}$
Softmax (0-1)	$s(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
ReLU(0,∞)	S(z)=max(0,z)
Linear (-∞,+∞)	S(z)= z

Example

Suppose a unit has:

$w = [0.2, 0.3, 0.9]$

$b = 0.5$

What happens with input x:

$x = [0.5, 0.6, 0.1]$

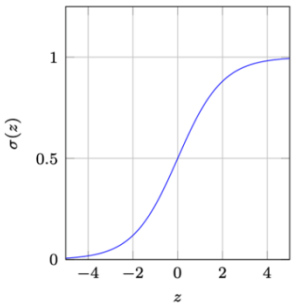
$Z = w_1x_1 + w_2x_2 + w_3x_3 + b$

$= 0.2 \times 0.5 + 0.3 \times 0.6 + 0.9 \times 0.1 + 0.5$

$= 0.87$

Output:
Using Step function
Φ(z)=1
Output= 1

If Sigmoid is taken with 0.5 as threshold
S(z) = 0.70
S(z)>=0.5 → 1



(a) Sigmoid activation function.

- **Input Layer:** The input layer consists of one or more input neurons, which receive input signals from the external world or from other layers of the neural network.
- **Weights:** Each input neuron is associated with a weight, which represents the strength of the connection between the input neuron and the output neuron.
- **Bias:** A bias term is added to the input layer to provide the perceptron with additional flexibility in modeling complex patterns in the input data.
- **Activation Function:** The activation function determines the output of the neuron based on the weighted sum of the inputs and the bias term. Common activation functions used include the step function, sigmoid function, and ReLU function.

1. The neuron computes the weighted sum (z) of the inputs (x)

$$z = \sum (x_i \cdot w_i) + b$$

2. The weighted sum (z) is then passed through an activation function. This introduces non-linearity into the network, enabling it to learn complex patterns.

Common activation functions include:

- **Sigmoid:** Outputs a value between 0 and 1.

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

- **ReLU (Rectified Linear Unit):** Outputs the input if positive, otherwise 0.

$$f(z) = \max(0, z)$$

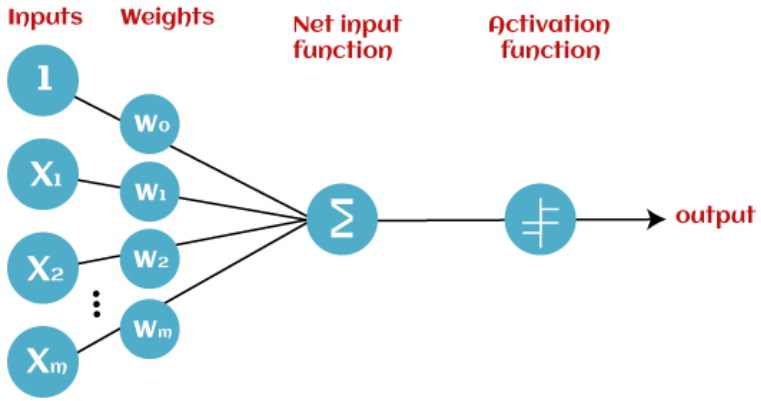
- **Tanh:** Outputs values between -1 and 1.

$$f(z) = \tanh(z)$$

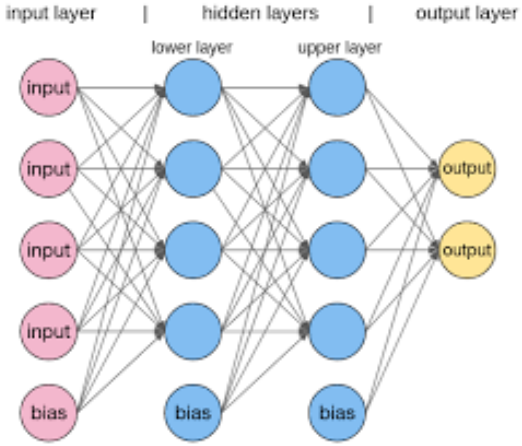
3. The output of the activation function becomes the neuron's output, which can be passed to neurons in the next layer. The process continues through the network until the final output is produced. The final output indicates the class or category to which the input data belongs.

4. Training Algorithm: During training, the weights and biases are adjusted to minimize the error between the predicted output and the true output for a given set of training examples.

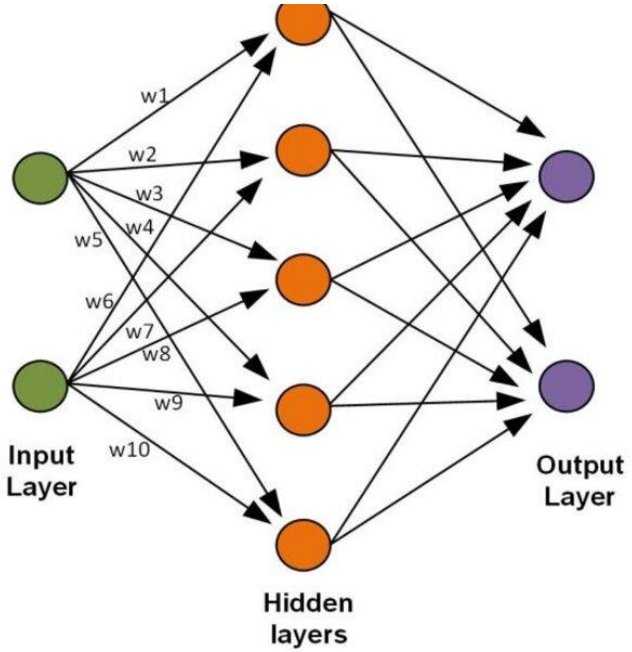
Perceptron



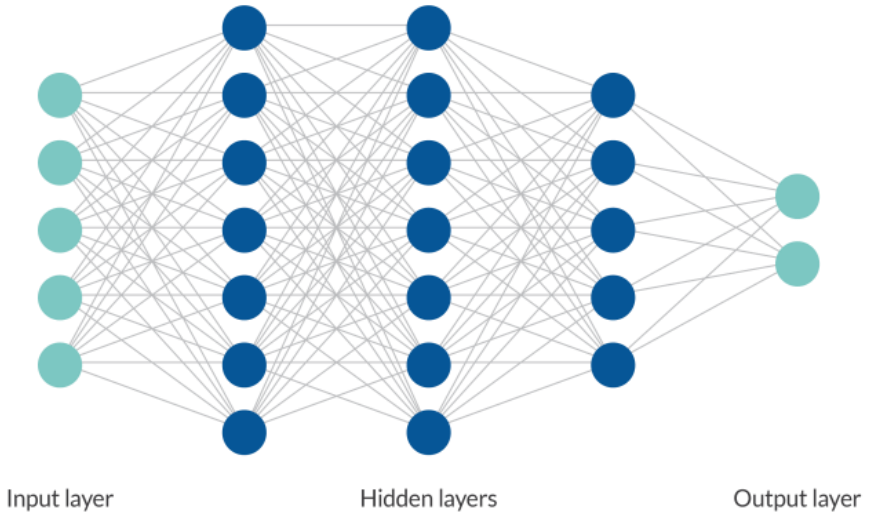
Multi Layer Perceptron



Multi Layer Perceptron



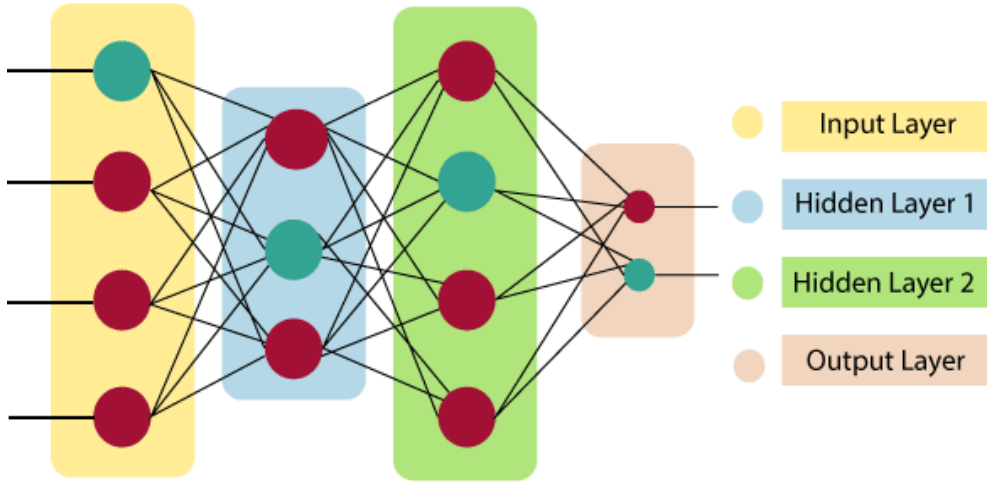
Deep Neural Network (MLP with multiple hidden layers)



- How much do you remember?
  - Z
  - Activation function
  - Deep neural network

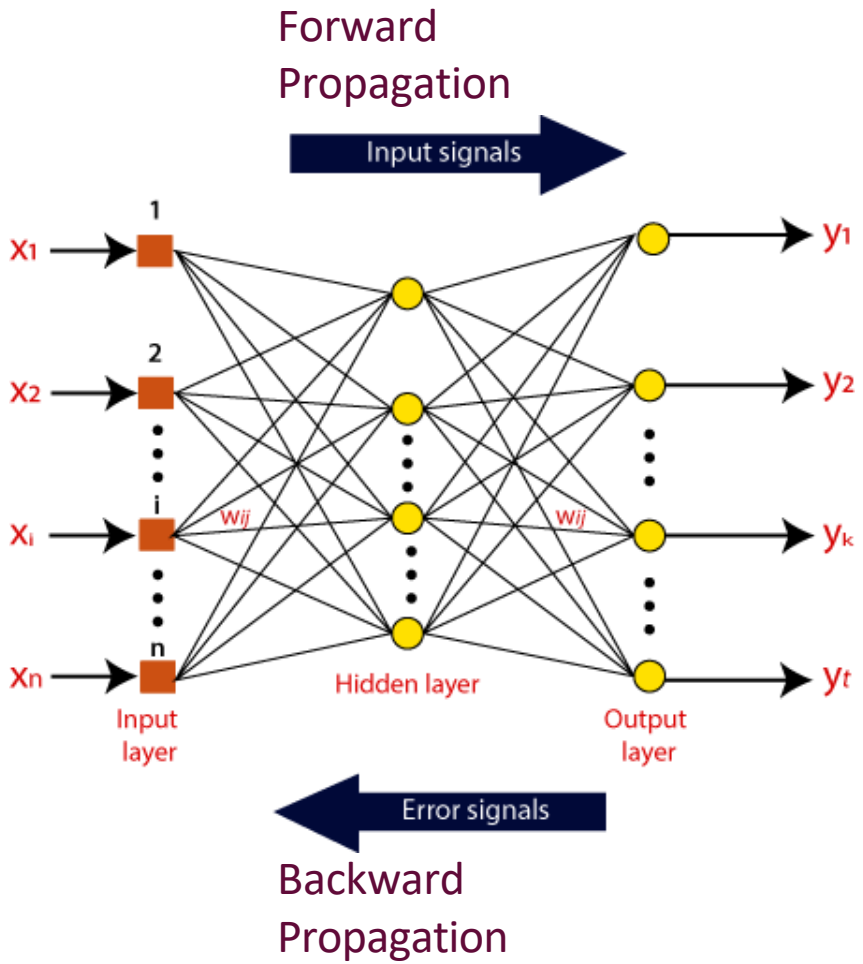
Components / Architecture of Neural Network

- A simple neural network consists of three components :
  - **Input Layer:**
    - As the name suggests, it accepts inputs in several different formats provided by the programmer.
  - **Hidden Layer(s):**
    - The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.
    - Extracts some of the most relevant patterns from the inputs and sends them on to the next layer for further analysis
  - **Output Layer:**
    - The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.



How do artificial neural networks learn?

- ANNs learn through a process called **training**, where the network adjusts its weights and biases to minimize the error between predicted and actual outputs.
- **This involves two phases:**
- Forward propagation and
- Backpropagation



Forward Propagation

- First step of training a neural network is Forward Propagation
- The input data is fed in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function and passes to the successive layer.
- At each neuron in a hidden or output layer, the processing happens in two steps:

  1. **Pre-activation:** computes the *weighted sum of inputs* i.e. the *linear transformation of weights* w.r.t to inputs.

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

  2. **Activation:** the calculated weighted sum of inputs is passed to the activation function (eg. Sigmoid).

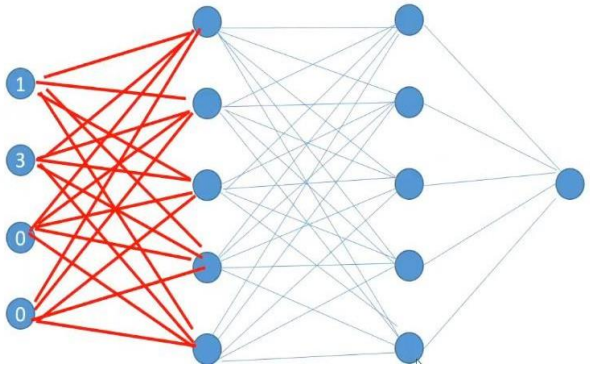
$$s(z) = \frac{1}{1 + e^{-z}}$$

Example

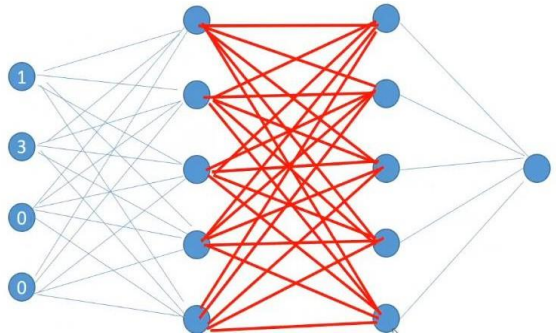
Gender	Age	Hypertension	Does_smoke	Stroke
1	3	0	0	0
1	58	1	1	1
0	8	0	0	0
0	70	0	1	1
1	14	0	0	0

Forward Propagation

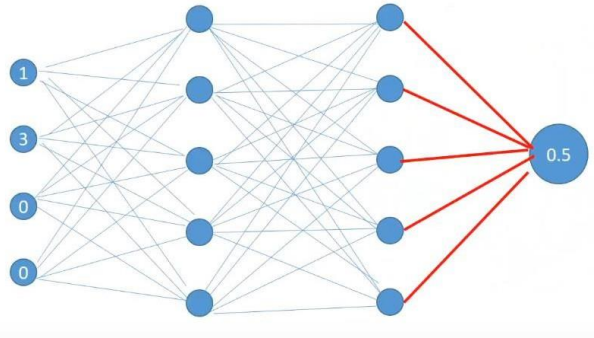
Actual	Predicted
0	0.5



first hidden layer activation



second hidden layer activation



Final output calculation

This series of calculations which takes us from the input to output is called Forward Propagation.

- After forward propagation the predicted value of the first row is 0.5 against the actual value of 0.

Actual	Predicted
0	0.5

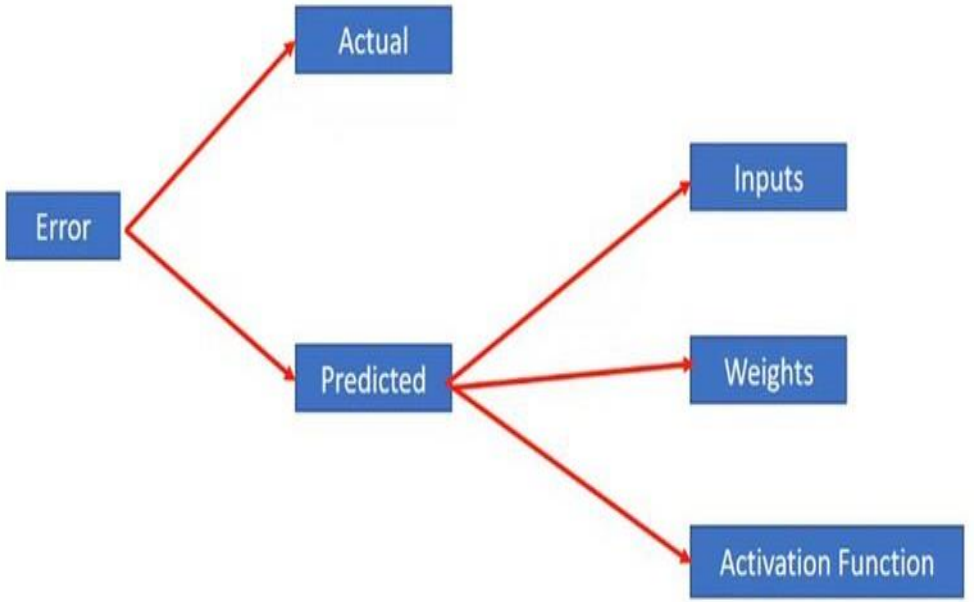
- **How does the computer know there is error?**
  - Use **cost/ loss/ error function** to find the difference between actual o/p and predicted o/p.
  - An example of cost function is Mean Square Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Common tasks have "standard" loss functions:
- mean squared error for regression
- binary cross-entropy for two-class classification
- categorical cross-entropy for multi-class classification
- etc.

What Caused the Error?

- The predicted values are dependent upon 3 things-
  - Inputs to the output layer i.e the hidden activation of the second layer
  - Weights between the second hidden layer and the output
  - And the activation function present at the output layer
- Inputs and activation can’t be changed.
- Hence wrong weights caused the error. To reduce error we need to adjust the weights
- But we cannot change the weights in arbitrary amounts to check out what weights produce the lowest error as this would amount to an infinite amount of time to check what sort of weights produce the lowest error.



How to reduce Errors?

Adjust the weights and reduce the errors.

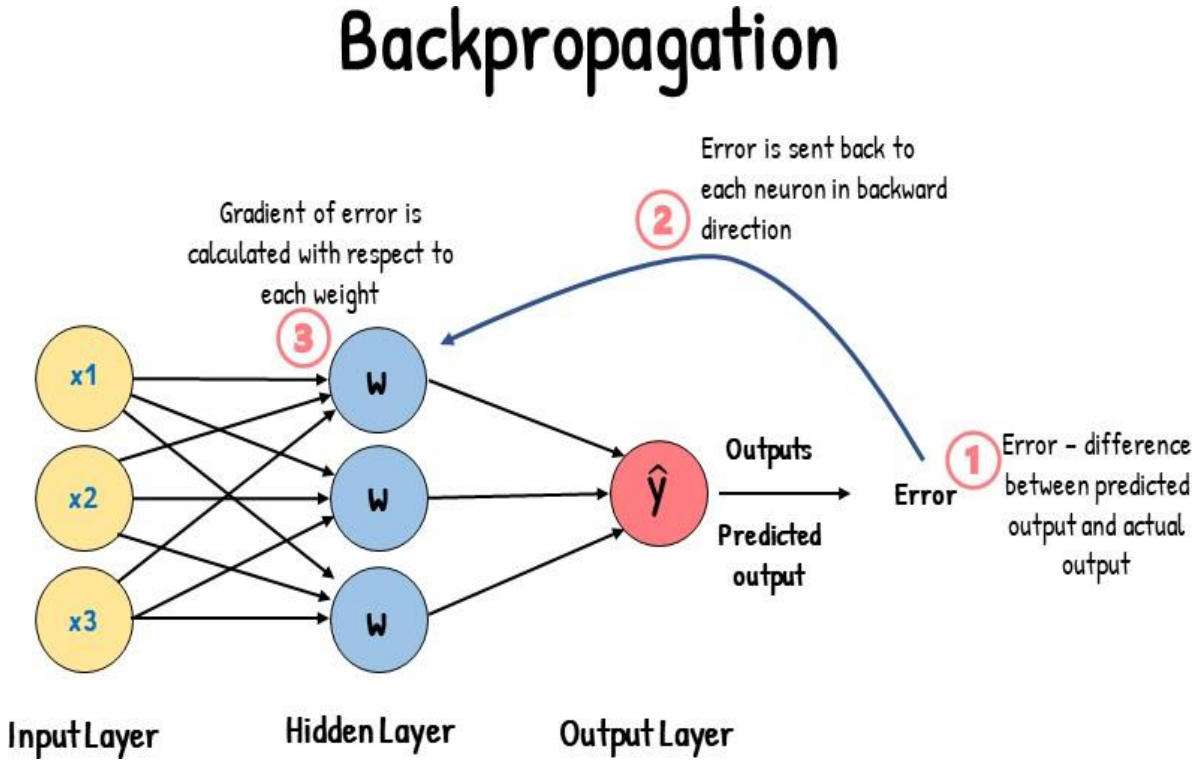
First – observe the amount of change in error on changing the weight by a small amount

Second – note the direction of that change.

A popular algorithm is **backpropagation**, which uses gradient descent to update weights.

Backpropagation

- The parameters (weights) of the neural network have a relationship with the error the neural net produces; and hence, when the parameters change, the error does, too.
- To adjust the weights, we use backpropagation algorithm. The error/ loss is fed backwards in such a way that we can fine-tune the weights
- **B**ackpropagation (backward propagation of errors) is the method of fine-tuning the weights of a neural network based on the error obtained in the previous epoch (i.e., iteration).
- The aim is to find optimal weights (parameters) corresponding to lowest error.
- [Backpropagation in Neural Network - GeeksforGeeks]



Epoch
1 forward propagation
&
1 Backward propagation

An Example - backpropagation

Output and error with different values of '**W**'.

Input	Desired Output
0	0
1	2
2	4

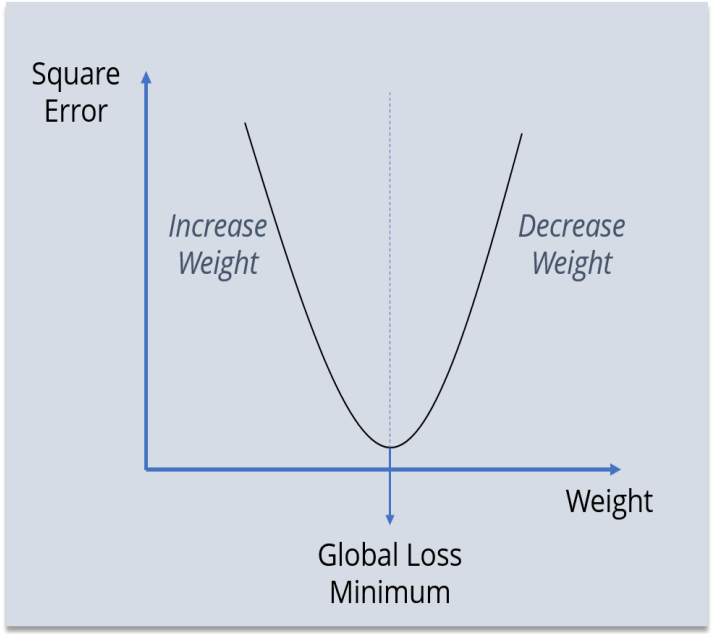
Input	Desired Output	Model output (W=3)
0	0	0
1	2	3
2	4	6

Input	Desired Output	Model output (W=3)	Absolute Error	Squared Error
0	0	0	0	0
1	2	3	1	1
2	4	6	2	4

Input	Desired Output	Model output (W=4)	Absolute Error	Square Error
0	0	0	0	0
1	2	4	2	4
2	4	8	4	16

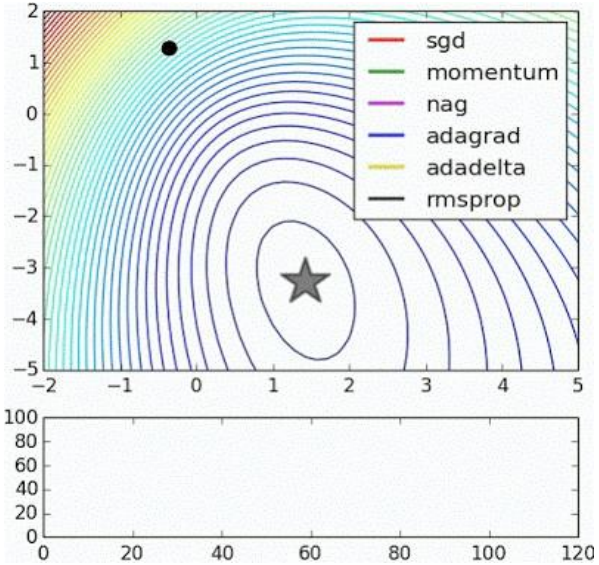
input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=2)	Square Error
0	0	0	0	0	0	0
1	2	3	2	4	3	1
2	4	6	2	4	4	0

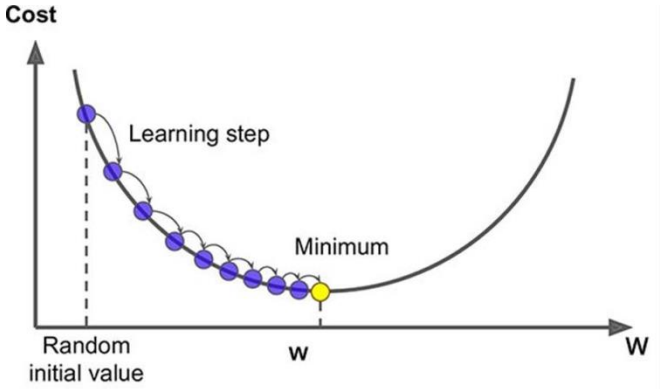
- We first initialized some random value to ‘W’ and propagated forward.
- Then, we noticed that there is some error. To reduce that error, we propagated backwards and increased the value of ‘W’.
- After that, also we noticed that the error has increased. We came to know that, we can’t increase the ‘W’ value.
- So, we again propagated backwards and we decreased ‘W’ value; Now, we noticed that the error has reduced.

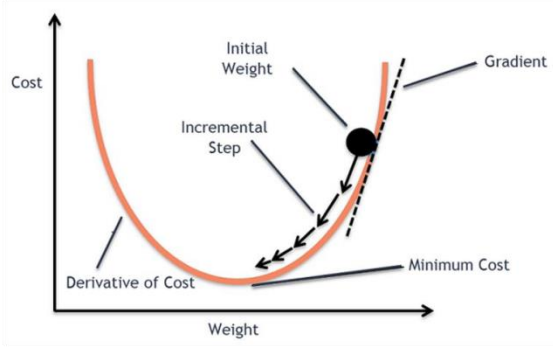


How to know the optimal weights with minimum error?

- Optimization
  - optimization algorithm is used to find the weights that minimize the cost function.
  - Minimizing the cost function means getting to the minimum point of the cost function, corresponding to lowest cost/loss
    - Ex: Gradient descent





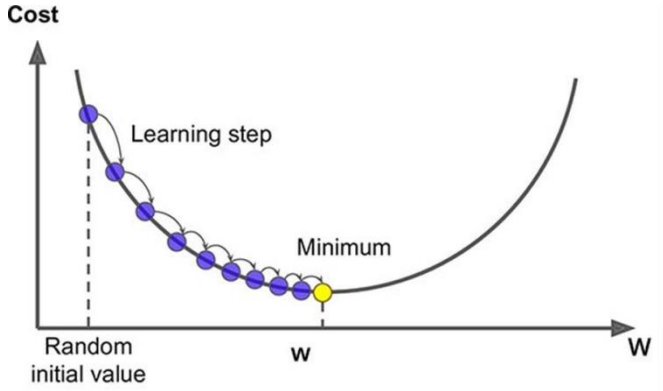


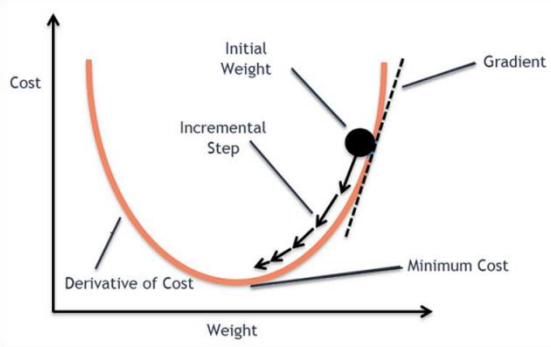
- We are seeking to minimize the error (also known as the **loss function or the objective function or cost function –** function for calculating the error)
  - **Example of cost function: Mean squared error or Cross-entropy loss (log loss)**
- The optimization function/algorithm (Gradient Descent) will help us find the weights that will — hopefully — yield a smaller loss in the next iteration.
- The key calculation involves calculating the gradients for each weight and bias in the network.
- A gradient measures how much the output of a function changes if you change the inputs a little bit**. It** tells us how much each weight/bias should be adjusted to minimize the error in the next forward pass.
- The chain rule is used iteratively to calculate this gradient efficiently.

- You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn.
- When the slope is zero, the model stops learning.
- In mathematical terms, a gradient is a partial derivative with respect to its inputs
- we can define the process of gradient descent as

$$W_i := W_i + \Delta w_i$$

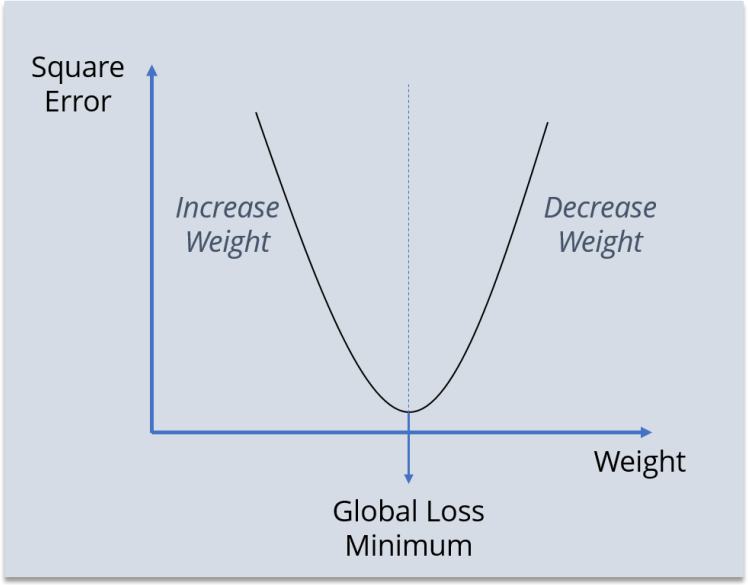
- Where $\Delta W_i = -\alpha \frac{\partial L}{\partial W_i}$, where L is the loss function.
- ΔW_i is the "step" we take walking along the gradient, and we set a learning rate, α to control the size of our steps.





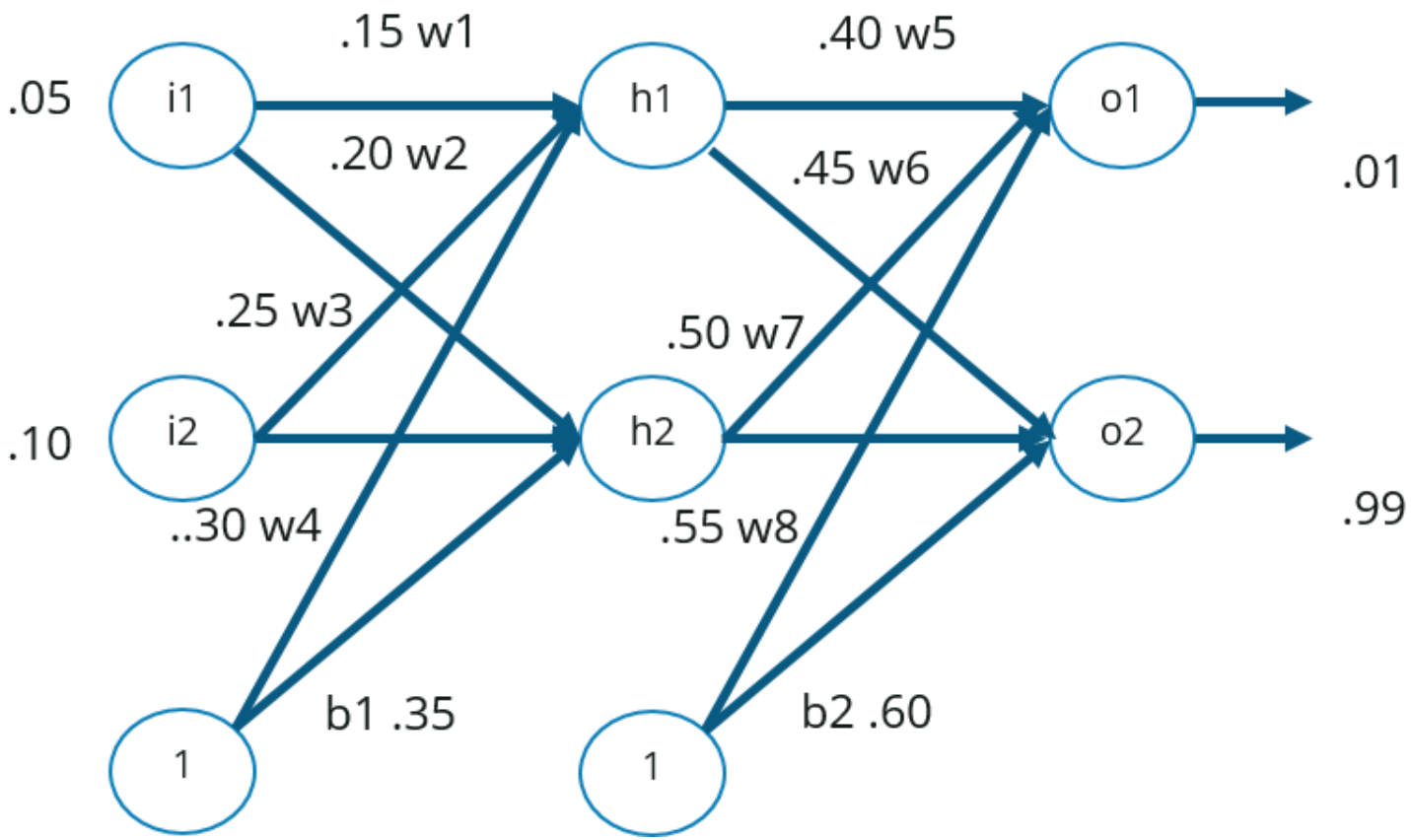
- Use this formula to update each of the weights, recompute forward propagation with the new weights, backpropagate the error, and calculate the next weight update. This process continues until we've converged on an optimal value for our parameters.
- During each iteration we perform forward propagation to compute the outputs and backward propagation to compute the errors; one complete iteration is known as an epoch.
- It is common to report evaluation metrics after each epoch so that we can watch the evolution of our neural network as it trains.

- At some point if you further update the weight, the error will increase. At that time you need to stop, and that is your final weight value



- Once the optimal weights are learned, the model can be used for classification/ regression.

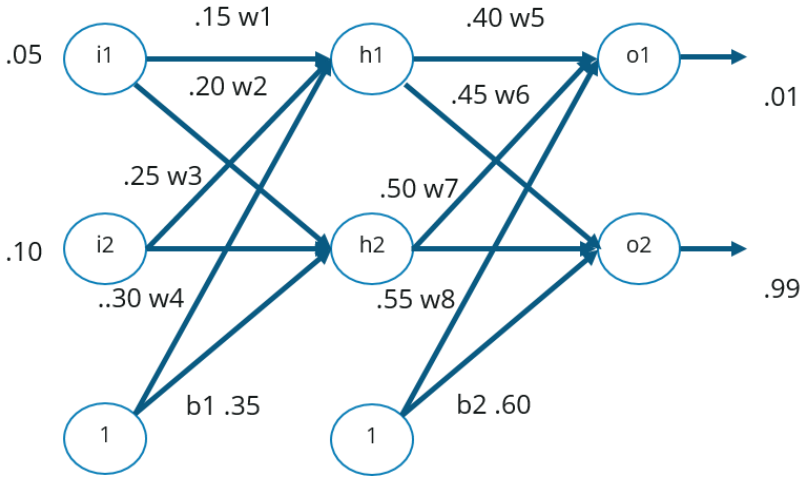
How Backpropagation Works?: Numerical Example



The steps involved in Backpropagation:

- Step – 1: Forward Propagation
- Step – 2: Backward Propagation
- Step–3: Putting all the values together and calculating the updated weight value

- **Step – 1: Forward Propagation**



Net Input For h1:

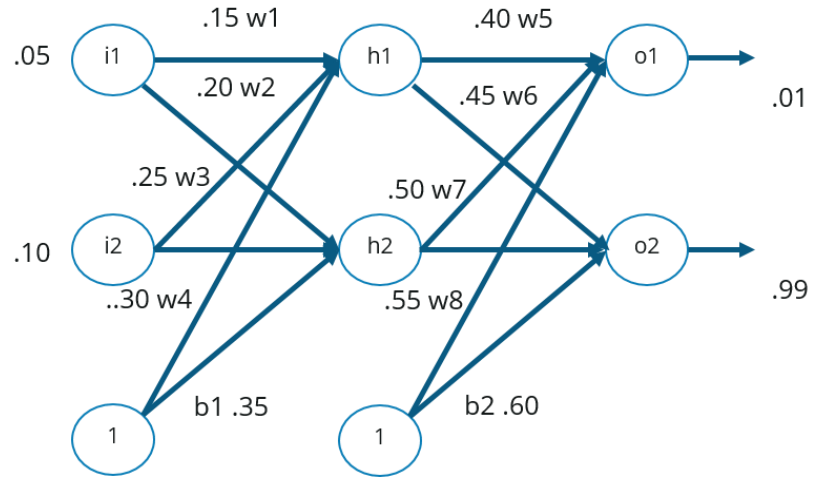
net h1 = w1*i1 + w2*i2 + b1*1 → net h1 = 0.15*0.05 + 0.2*0.1 + 0.35*1 = 0.3775

Output Of h1:

out h1 = $1/1+e^{-\text{net h1}}$ → $1/1+e^{.3775}$ = 0.593269992

Output Of h2:

out h2 = 0.596884378



Output For o1:

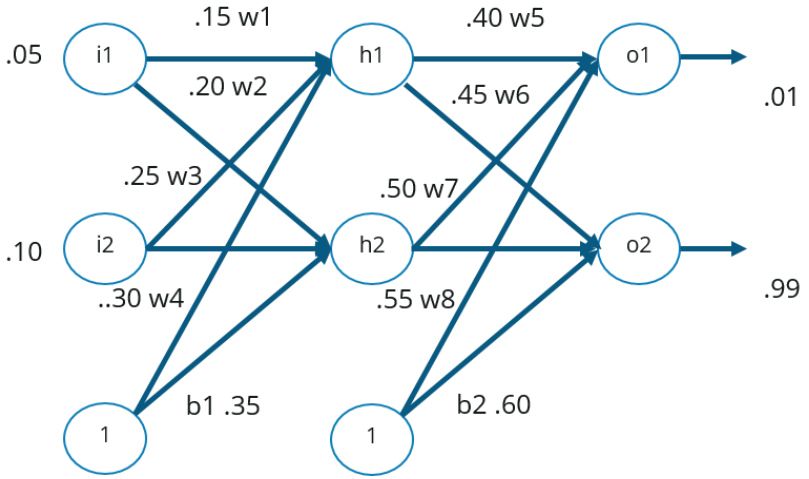
net o1 = w5*out h1 + w6*out h2 + b2*1 → 0.4*0.593269992 + 0.45*0.596884378 + 0.6*1 = 1.105905967

Out o1 = $1/1+e^{-\text{net o1}}$ → $1/1+e^{-1.105905967}$ = 0.75136507

Output For o2:

Out o2 = 0.772928465

- what is the value of the error ?



Error For o1:

$E_{o1} = \Sigma 1/2(target - output)^2$ → $\frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$

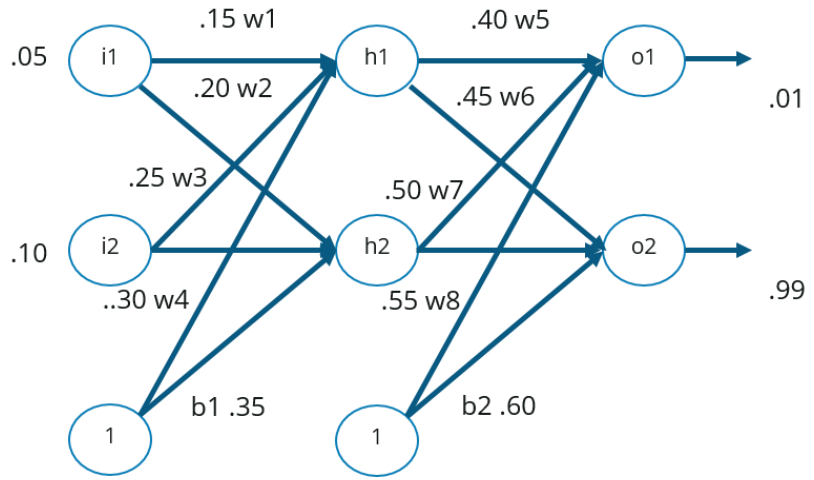
Error For o2:

$E_{o2} = 0.023560026$

Total Error:

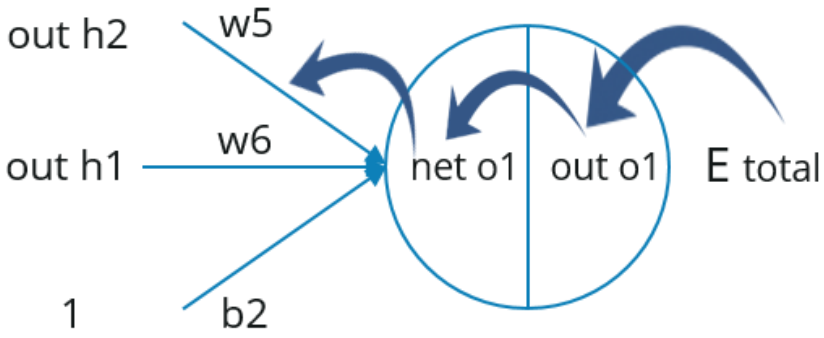
$E_{total} = E_{o1} + E_{o2}$ → $0.274811083 + 0.023560026 = 0.298371109$

- **Step – 2: Backward Propagation**



- we will try to reduce the error by changing the values of weights and biases.
- calculate the rate of change of error w.r.t change in weight W5.

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$



$$E_{total} = 1/2(\text{target o1} - \text{out o1})^2 + 1/2(\text{target o2} - \text{out o2})^2$$

$$\frac{\delta Etotal}{\delta out\ o1} = -(\text{target o1} - \text{out o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$\text{out o1} = 1/1+e^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = \text{out o1 (1 - out o1)} = 0.75136507\ (1 - 0.75136507) = 0.186815602$$

$$\text{net o1} = w5 * \text{out h1} + w6 * \text{out h2} + b2 * 1$$

$$\frac{\delta net\ o1}{\delta w5} = \text{out h1}\ w5^{(1-1)} + 0 + 0 = 0.593269992$$

- **Step – 3: Putting all the values together and calculating the updated weight value**

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$

→ 0.082167041

updated value of W5:

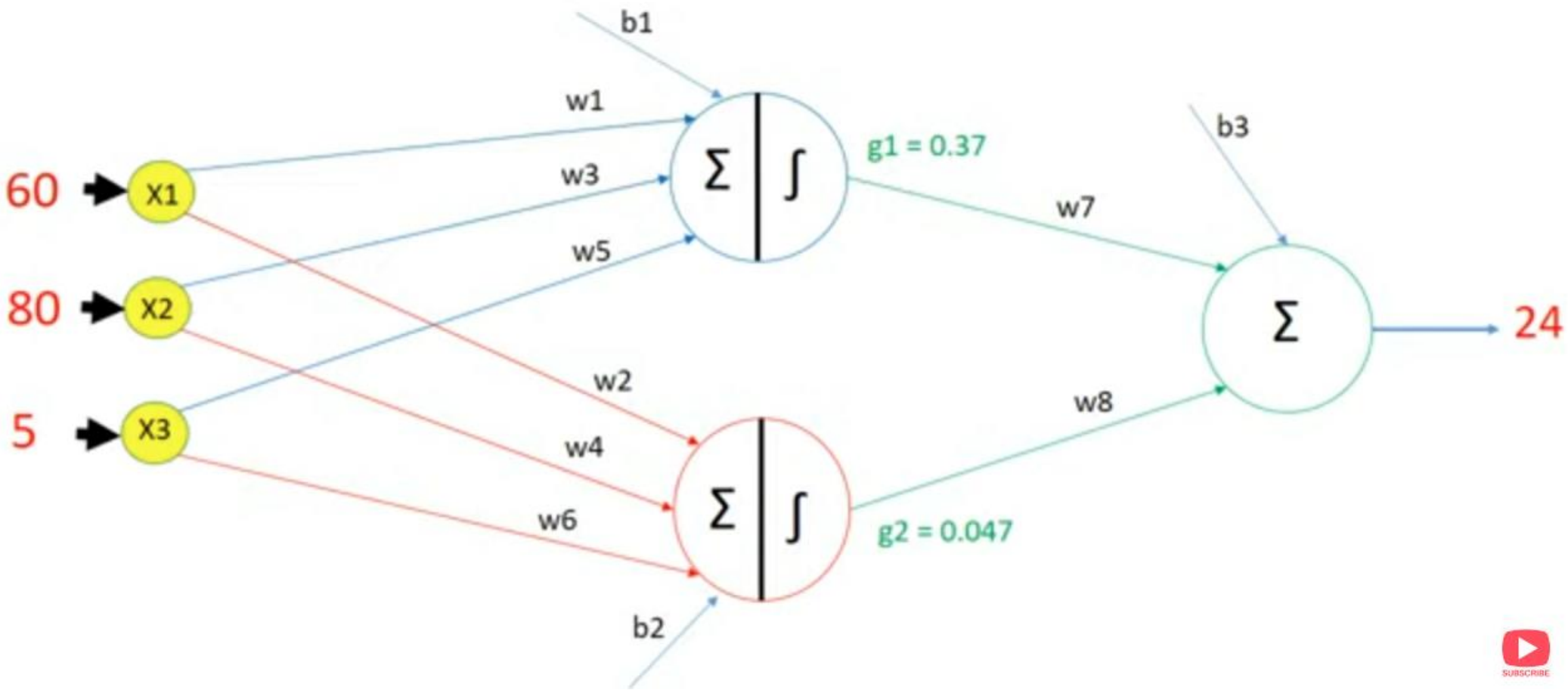
$$w5^{+} = w5 - n\frac{\delta Etotal}{\delta w5}$$

→ $w5^{+} = 0.4 - 0.5 * 0.082167041$

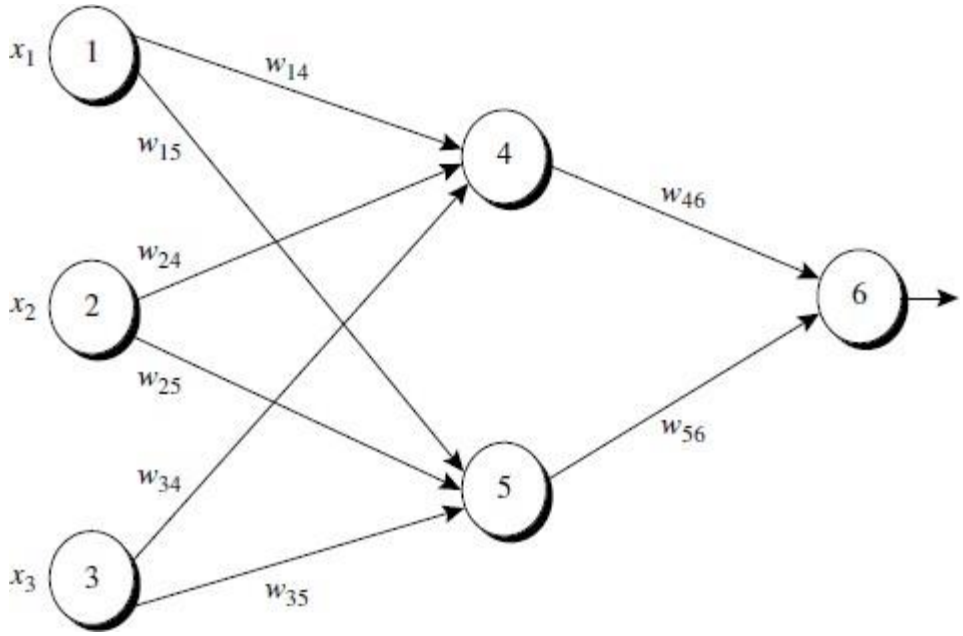
Updated w5 → 0.35891648

- Similarly, we can calculate the other weight values as well.
- After that we will again propagate forward and calculate the output. Again, we will calculate the error.
- If the error is minimum we will stop right there, else we will again propagate backwards and update the weight values.
- This process will keep on repeating until error becomes minimum

Q1


b1
w1
w3
w5
60
X1
80
X2
5
X3
Σ
∫
g1 = 0.37
b3
w7
Σ
24
w2
w4
w6
Σ
∫
g2 = 0.047
w8
b2
SUBSCRIBE

Q2



Question

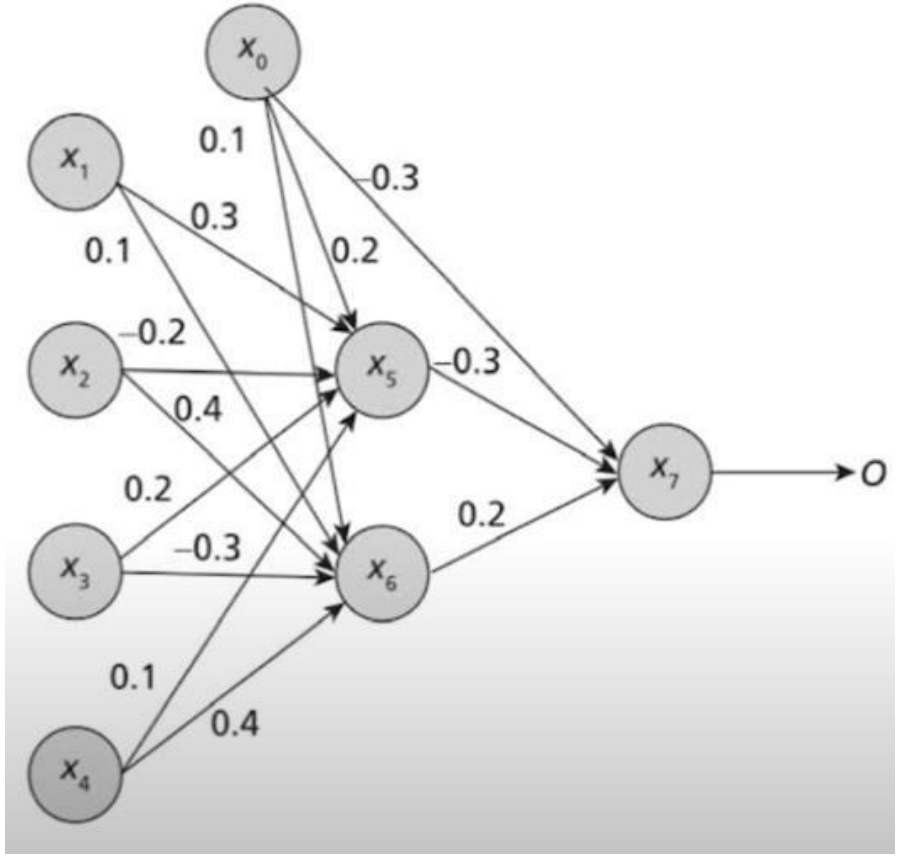
For the given multilayer feed-forward neural network, Let the learning rate be 0.9. The initial weight and bias values of the network are given in Table along with the first training tuple,

$\boldsymbol{X}$ ={ 1, 0, 1}, with a class label of 1. Find the updated weights and biases after 1 epoch

Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Question 3



Learning Rate = 0.8

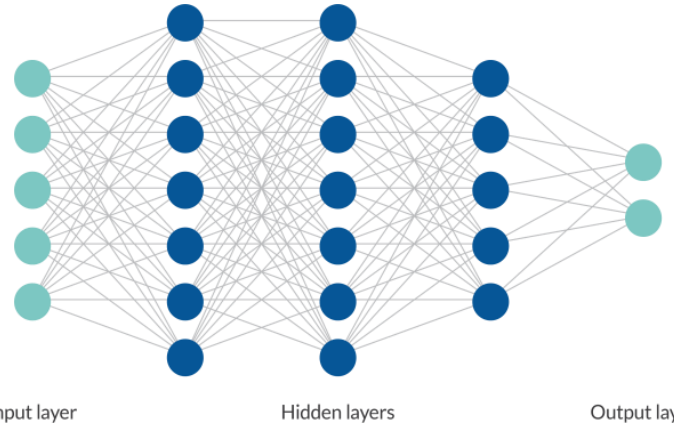
x_1	x_2	x_3	x_4	$O_{Desired}$
1	1	0	1	1

Activation function: Sigmoid
X0= Bias

x_1	x_2	x_3	x_4	w_{15}	w_{16}	w_{25}	w_{26}	w_{35}	w_{36}	w_{45}	w_{46}	w_{57}	w_{67}	θ_5	θ_6	θ_7
1	1	0	1	0.3	0.1	−0.2	0.4	0.2	−0.3	0.1	0.4	−0.3	0.2	0.2	0.1	−0.3

Feed Forward NN

- A **multilayer feed-forward** neural network consists of an *input layer*, one or more *hidden layers*, and an *output layer*.
- Each layer is made up of units.
- The inputs to the network correspond to the attributes measured for each training tuple. The inputs are fed simultaneously into the units making up the **input layer**.
- These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of "neuronlike" units, known as a **hidden layer**. The outputs of the hidden layer units can be input to another hidden layer, and so on. The number of hidden layers is arbitrary.
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction for given tuples.
- Each output unit takes, as input, a weighted sum of the outputs from units in the previous layer. It applies a nonlinear (activation) function to the weighted input.
- Multilayer feed-forward neural networks are able to model the class prediction as a nonlinear combination of the inputs. From a statistical point of view, they perform nonlinear regression.
- *Multilayer feed-forward networks, given enough hidden units and enough training samples, can closely approximate any function.*



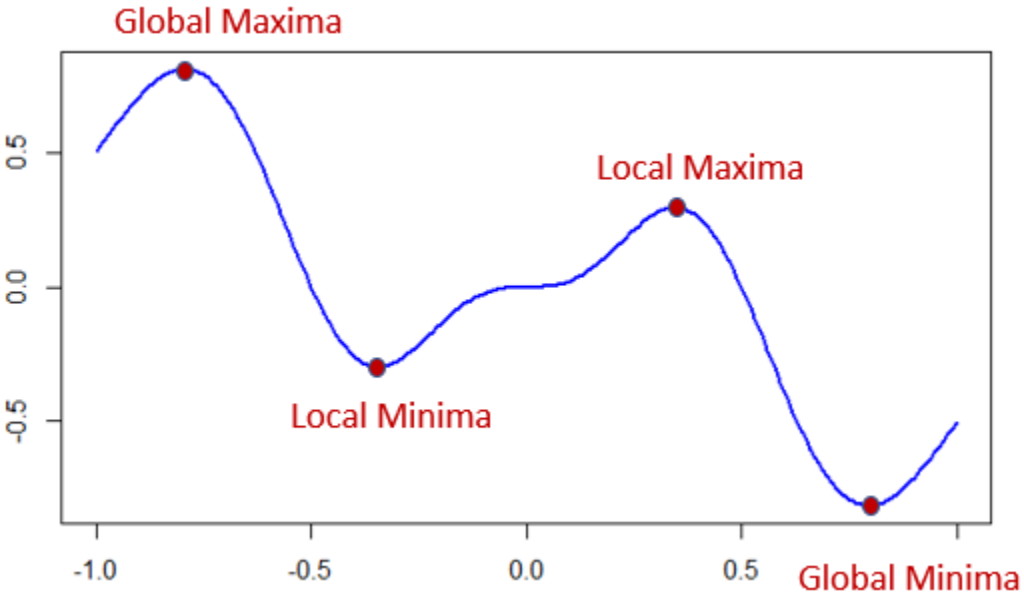
- In a feed-forward network none of the weights cycles back to an input unit or to a previous layer's output unit.
- A network is **fully connected if** each unit provides input to each unit in the next forward layer.

Appendix

Understanding Gradient Descent

An analogy for understanding gradient descent

- The basic intuition behind gradient descent can be illustrated by a hypothetical scenario. A person is stuck in the mountains and is trying to get down (i.e., trying to find the global minimum). There is heavy fog such that visibility is extremely low. Therefore, the path down the mountain is not visible, so they must use local information to find the minimum. They can use the method of gradient descent, which involves looking at the steepness of the hill at their current position, then proceeding in the direction with the steepest descent (i.e., downhill). If they were trying to find the top of the mountain (i.e., the maximum), then they would proceed in the direction of steepest ascent (i.e., uphill).
- Using this method, they would eventually find their way down the mountain or possibly get stuck in some hole (i.e., local minimum or saddle point), like a mountain lake. However, assume also that the steepness of the hill is not immediately obvious with simple observation, but rather it requires a sophisticated instrument to measure, which the person happens to have at the moment. It takes quite some time to measure the steepness of the hill with the instrument, thus they should minimize their use of the instrument if they wanted to get down the mountain before sunset. The difficulty then is choosing the frequency at which they should measure the steepness of the hill so not to go off track.



What is Chain Rule?

- The rule applied for finding the derivative of the composite function (e.g. cos 2x, log 2x, etc.) is basically known as the chain rule. It is also called the composite function rule.
- The chain rule is applicable only for composite functions.
  - Ex: $f(x) = \sin(2x^2 - 6x)$

Chain Rule Formula

The formula of chain rule for the function y = f(x), where f(x) is a composite function such that x = g(t), is given as:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

Chain Rule in Differentiation

Let f represent a real valued function which is a composition of two functions u and v such that:

f = v(u(x))

Let us assume u(x) = t

Now if the functions u and v are differentiable and dt/dx and dv/dt exist, then the composite function f(x) is also differentiable. This can be done as given below.

df/dx = (dv/dt) × (dt/dx)

- As the name suggests, chain rule means differentiating the terms one by one in a chain form, starting from the outermost function to the innermost function.
- In layman terms, to differentiate a composite function at any point in its domain, first differentiate the outer part (i.e. the function enclosing some other function) and then multiply it with the inner function's derivative function.
- This will provide us with the desired differentiation.

Chain Rule Solved Examples

Example 1:

Find the derivative of the function $f(x) = \sin(2x^2 - 6x)$.

Solution:

The given can be expressed as a composite function as given below:

$f(x) = \sin(2x^2 - 6x)$

$u(x) = 2x^2 - 6x$

$v(t) = \sin t$

Thus, $t = u(x) = 2x^2 - 6x$

$\Rightarrow f(x) = v(u(x))$

According to the chain rule,

$df(x)/dx = (dv/dt) \times (dt/dx)$

Where,

$dv/dt = d/dt (\sin t) = \cos t$

$dt/dx = d/dx [u(x)] = d/dx (2x^2 - 6x) = 4x - 6$

Therefore, $df/dx = \cos t \times (4x - 6)$

$= \cos(2x^2 - 6x) \times (4x - 6)$

$= (4x - 6) \cos(2x^2 - 6x)$

Derivative of the Sigmoid Function

- Sigmoid function is a mathematical function that has an “S”-shaped curve (sigmoid curve).
- It is widely used in various fields, including machine learning, statistics, and artificial intelligence, particularly for its smooth and bounded nature.
- The sigmoid function is often used to introduce non-linearity in models, especially in neural networks.
- The formula of the sigmoid activation function is:

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

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

Derivative is given as:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Let's derive this derivative of sigmoid function as follows:

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

Let $u = 1 + e^{-x}$. Thus, $y = 1/u$.

First, find the derivative of u with respect to x :

$du/dx = -e^{-x}$.

Then, find the derivative of y with respect to u:

$$\frac{dy}{du} = -\frac{1}{u^2}$$

Apply the chain rule:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = -\frac{1}{u^2} \cdot (-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

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

$$\text{Thus,} \sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1}{1+e^{-x}}\right)\left(\frac{e^{-x}}{1+e^{-x}}\right) = \sigma(x)\,(1-\sigma(x))$$