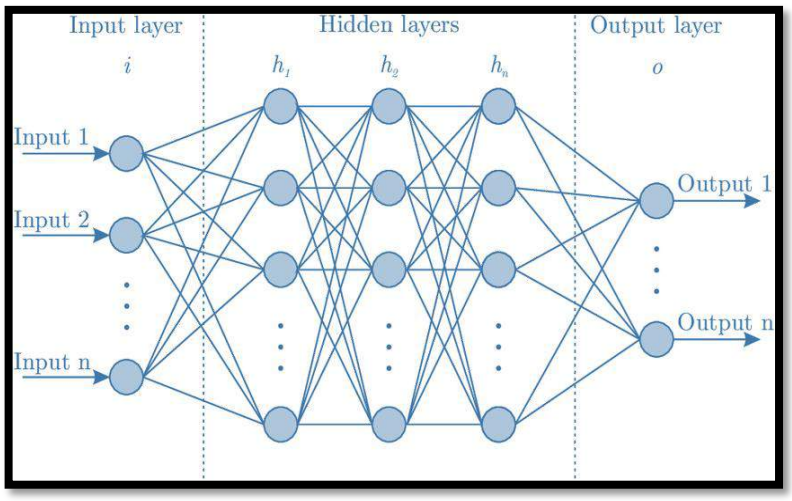
UNIT - V

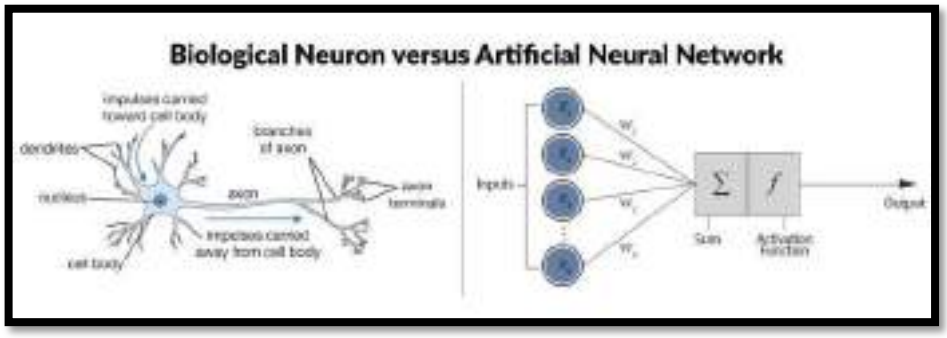
## Introduction to Neural Networks:

- A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain.
- A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.
- It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain.

### Artificial Neural Networks:




- Artificial Neural Network is an algorithm that works in a similar fashion as the human brain processes and analyzes data. It is made up of connected nodes called neurons stacked in different layers that perform complex computations to solve problems just like humans would do
- *The Artificial Neural Network (ANN) is an attempt at modelling the information processing capabilities of the biological nervous system*. The human body is made up of trillions of cells, and the nervous system cells – called neurons – are specialized to carry "messages" through an electrochemical process. The nodes in ANN are equivalent to those of our neurons, whose nodes are connected to each other by *Synaptic Weights (or simply weights)* – equivalent to the synaptic connections between axons and dendrites of the biological neuron.




- Let's think of a scenario where you're teaching a toddler how to identify different kinds of animals. You know that they can't simply identify any animal using basic characteristics like a color range and a

pattern: just because an animal is within a range of colors and has black vertical stripes and a slightly elliptical shape doesn't automatically make it a tiger.

- Instead, you should show them many different pictures, and then teach the toddler to identify those features in the picture on their own, hopefully without much of a conscious effort. This specific ability of the human brain to identify features and memorize associations is what inspired the emergence of ANNs.

- ANNs have been successfully applied in wide range of domains such as:
    - Classification of data – Is this flower a rose or tulip?
    - Anomaly detection – Is the particular user activity on the website a potential fraudulent behavior?
    - Speech recognition – Hey Siri! Can you tell me a joke?
    - Time series analysis – Is it good time to start investing in stock market?

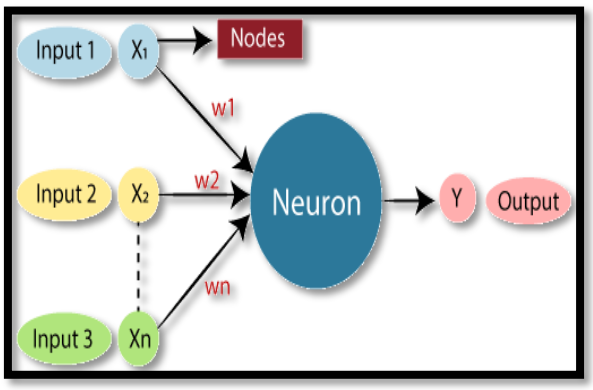
- A neural network has many layers and each layer performs a specific function, and as the complexity of the model increases, the number of layers also increases that why it is known as the multi-layer perceptron.

- Neural networks are computational algorithms or models that understand the data and process information. As these artificial neural networks are designed as per the structure of the human brain, the role of neurons in the brain is played by the perceptron in a neural network.

## Perceptron:

A perceptron is the smallest element of a neural network. Perceptron is a single-layer neural network linear or a Machine Learning algorithm used for supervised learning of various binary classifiers. It works as an artificial neuron to perform computations by learning elements and processing them for detecting the business intelligence and capabilities of the input data. A perceptron network is a group of simple logical statements that come together to create an array of complex logical statements, known as the neural network.

Neural Network with a single neuron -Perceptron




Each perceptron comprises four different parts:

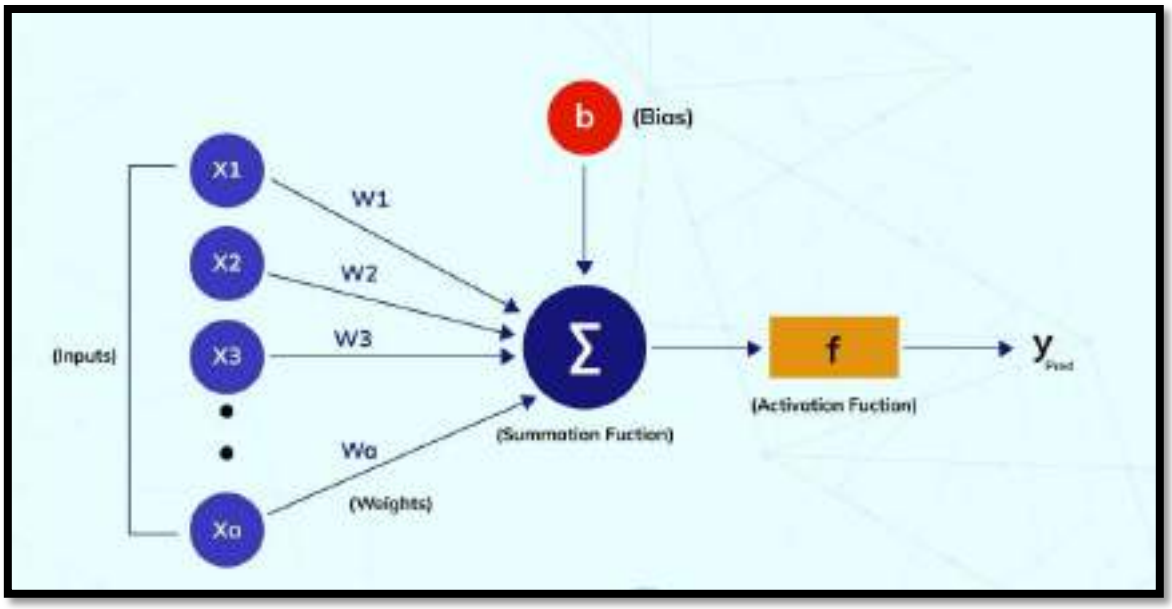
**Input Values**: A set of values or a dataset for predicting the output value. They are also described as a dataset's features and dataset.

**Weights:** The real value of each feature is known as weight. It tells the importance of that feature in predicting the final value. Weight parameters represent the strength of the connection between units. Higher is the weight, stronger is the influence of the associated input neuron to decide the output. Bias plays the same as the intercept in a linear equation.

**Bias:** The activation function is shifted towards the left or right using bias. You may understand it simply as the y-intercept in the line equation.

**Activation Function:** The activation function determines whether the neuron will fire or not. At its simplest, the activation function is a step function, but based on the scenario, different activation functions can be used.

## Working of a Perceptron



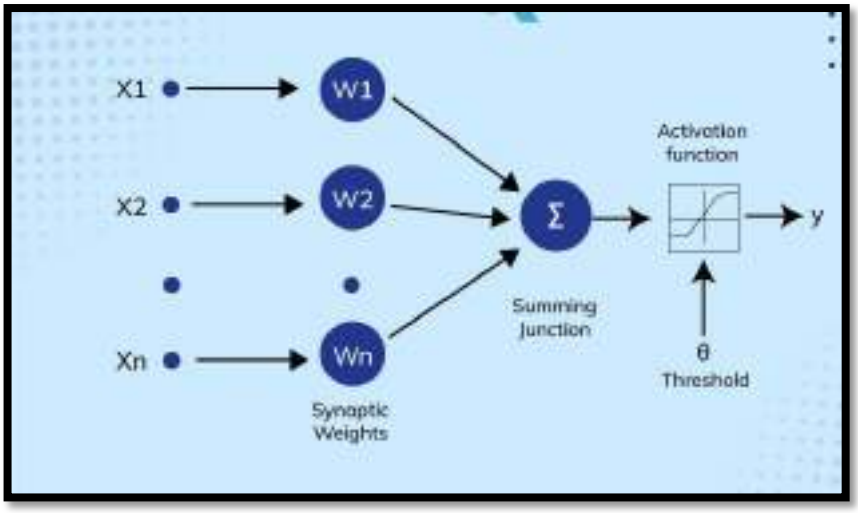

In the first step, all the input values are multiplied with their respective weights and added together. The result obtained is called weighted sum ∑wi*xi, or stated differently, $x1*w1 + x2*w2 + \ldots w_n*x_n$. This sum gives an appropriate representation of the inputs based on their importance. Additionally, a bias term *b* is added to this sum ∑wi*xi + *b*. Bias serves as another model parameter (in addition to weights) that can be tuned to improve the model's performance.

In the second step, an activation function *f* is applied over the above sum ∑wi*xi + b to obtain output Y = *f*(∑wi*xi + b). Depending upon the scenario and the activation function used, the Output is either binary {1, 0} or a continuous value.

A biological neuron only fires when a certain threshold is exceeded. Similarly, the artificial neuron will also only fire when the sum of the inputs (weighted sum) exceeds a certain threshold value.

*Need for Weight and Bias?*

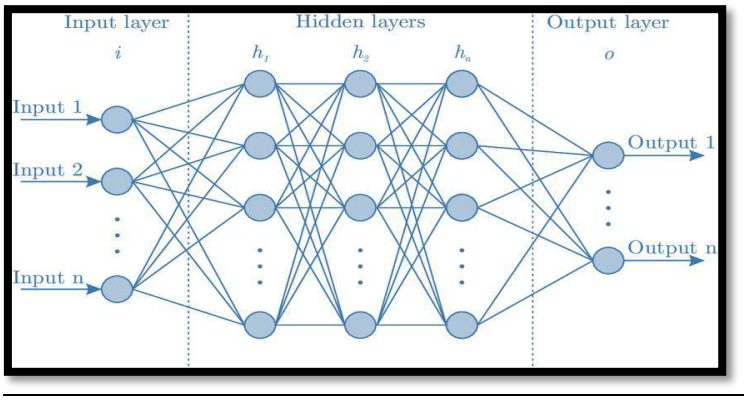
Weight and bias are two important aspects of the perceptron model. These are learnable parameters and as the network gets trained it adjusts both parameters to achieve the desired values and the correct output.




Weights are used to measure the importance of each feature in predicting output value. Features with values close to zero are said to have lesser weight or significance. These have less importance in the prediction process compared to the features with values further from zero known as weights with a larger value. Besides high-weighted features having greater predictive power than low-weighting ones, the weight can also be positive or negative. If the weight of a feature is positive then it has a direct relation with the target value, and if it is negative then it has an inverse relationship with the target value.

In contrast to weight in a neural network that increases the speed of triggering an activation function, bias delays the trigger of the activation function. It acts like an intercept in a linear equation. Simply stated, Bias is a constant used to adjust the output and help the model to provide the best fit output for the given data.

## Architecture of Neural Network:

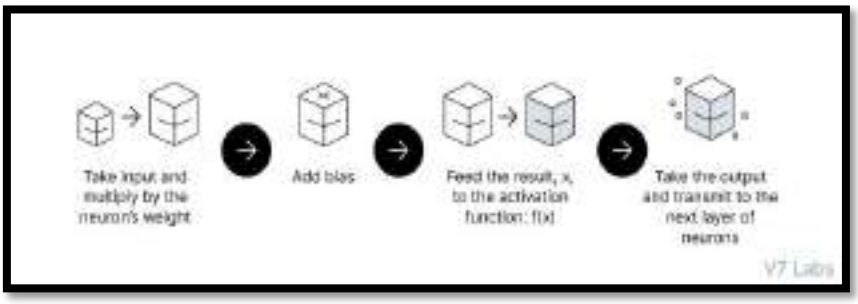



It consists of the input value and output value. Each input value is associated with its weight, which passes on to next level, each perceptron will have an activation function. The weights and input value forms a single perception.

### Forward Propagation :

Forward propagation is where input data is fed through a network, in a forward direction, to generate an output. The data is accepted by hidden layers and processed, as per the activation function, and moves to the successive layer.

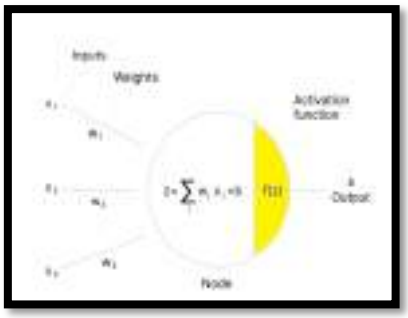
## Feed Forward Propagation




What is an Activation Function ?

An Activation Function decides whether a neuron should be activated or not.This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.

The role of the Activation Function is to derive output from a set of input values fed to a node (or a layer).




The following are the most popular activation functions in neural networks.

- Binary Step Function
- Sigmoid Function
- Tanh Function

- **Binary Step Function**
  - Binary step function depends on a threshold value that decides whether a neuron should be activated or not.
  - The input fed to the activation function is compared to a certain threshold; if the input is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.

*Binary step*

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

- **Sigmoid Function**
  - This function takes any real value as input and outputs values in the range of 0 to 1.
  - The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below

*Sigmoid / Logistic*

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

- **Tanh Function**
  - Tanh function is very similar to the sigmoid with the difference in output range of -1 to 1.
  - In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

*Tanh*

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

## Cost Function:

The cost function also called the loss function , computes the difference or distance between actual output and predicted output.

It determines the performance of a Machine Learning Model using a single real number, known as cost value/model error. This value depicts the average error between the actual and predicted outputs.

Error factor = Predicted – Actual.

Mean Squared Error (MSE) is one of the most commonly used Cost function methods. It calculates the square of the difference between the actual value and predicted value. Because of the square of the difference, it avoids any possibility of negative error. The formula for calculating MSE is :

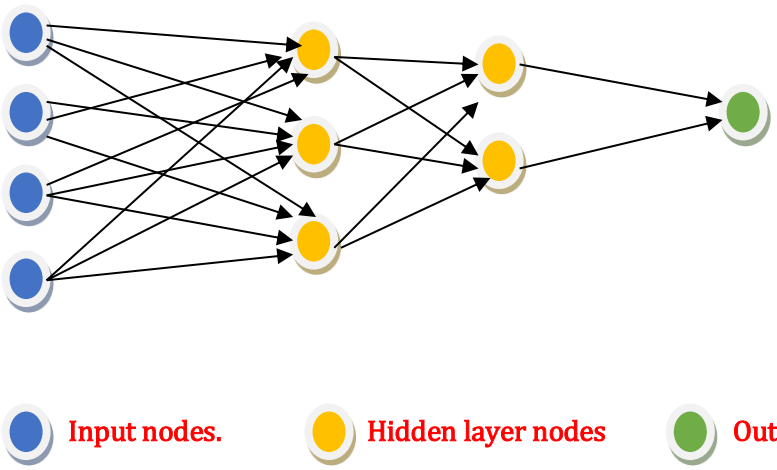
$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Exercise:

Draw an ANN with inputs ,hidden and output layers denoted as [4-3-2-1]

[ Note:

In the above notation,the first and last rows represent as input and output nodes respectively..and in between rows represent hidden layer nodes.]




## Back Propagation Algorithm in Neural Network

In an artificial neural network, the values of weights and biases are randomly initialized. Due to random initialization, the neural network probably has errors in giving the correct output. We need to reduce error values as much as possible. So, for reducing these error values, we need a mechanism that can compare the desired output of the neural network with the network's output that consists of errors and adjusts its weights and biases such that it gets closer to the desired output after each iteration. For this, we train the network such that it back propagates and updates the weights and biases. This is the concept of the back propagation algorithm.

- It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows to reduce error rates and make the model more reliable by increasing its generalization.
- It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.
- Definition:
  Backpropagation is the central mechanism by which neural networks learn. It is the messenger telling the network whether or not the network made a mistake when it made a prediction. ... Forward propagation is when a data instance sends its signal through a network's parameters toward the prediction at the end. Back-propagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

## Back Propagation - Set Up and Initialization:

Below are the steps that an artificial neural network follows to gain maximum accuracy and minimize error values:

- Parameter Initialization
- Feedforward Propagation
- Backpropagation

Parameter Initialization :
Weights and biases, associated with an artificial neuron are randomly initialized. After receiving the input, the network feeds forwards the input and it makes associations with weights and biases to give the output. The output associated with those random values is most probably not correct.

Feedforward propagation:

After initialization, when the input is given to the input layer, it propagates the input into hidden units at each layer. The nodes here do their job without being aware of whether the results produced are accurate or not (i.e., they don't re-adjust according to the results produced). Then, finally, the output is produced at the output layer. This is called feedforward propagation.

Back propagation :

The principle behind the back propagation algorithm is to reduce the error values in randomly allocated weights and biases such that it produces the correct output. The system is trained in the supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state. We need to update the weights so that we get the global loss minimum. This is how back propagation in neural networks works.

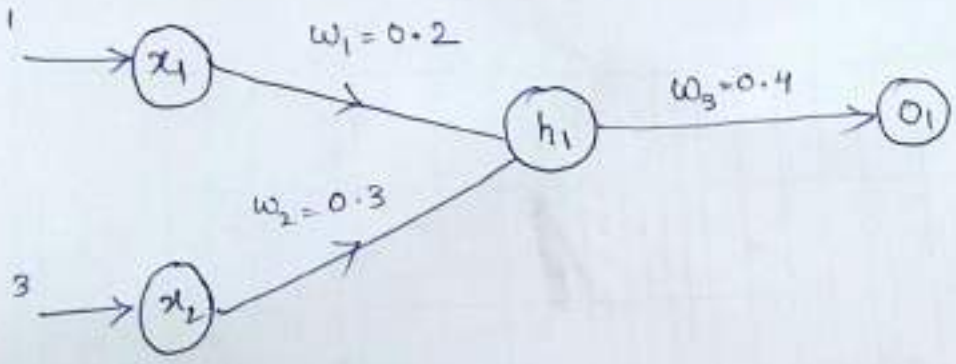
Working of Back Propagation Algorithm:

The goal of the back propagation algorithm is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs. Here, we will understand the complete scenario of back propagation in neural networks with the help of a single training set.

The inputs are $x_1 : 1$ and the target value $t_1$ is 0.3

$x_2 : 3$

Input Layer  Hidden layer  Output Layer




Input to Hidden Layer

1) $w_1 x_1 + w_2 x_2 = Zh_1$

$0.2(1) + 0.3(3) = 0.2 + 0.9 = 1.1$

$h_1 = a(Zh_1)$

$= a(1.1)$

$= 0.7503$

Hidden layer to output

$w_3 (h_1) = ZO_1$

$0.4(0.75) = ZO_1$

$ZO_1 = 0.3001$

$O_1 = a(ZO_1)$

$= a(0.3) = 0.5745$

Calulation of error value using Mean Square Error (MSE).

$$E = \frac{1}{2}(O_1 - t_1)^2$$

$$E = \frac{1}{2}(0.5745 - 0.3)^2 = \frac{(0.2745)^2}{2} = 0.03765$$

$$\frac{dE}{dO_1} = O_1 - t_1 = 0.27$$

Derivatives.

Calculation of Error with weights

(1) $\frac{dE}{dw_3} = \frac{dE}{dO_1} \cdot \frac{dO_1}{dZO_1} \cdot \frac{dZO_1}{dw_3}$

(2) $\frac{dE}{dw_2} = \frac{dE}{dh_1} \cdot \frac{dh_1}{dZh_1} \cdot \frac{dZh_1}{dw_2}$

(3) $\frac{dE}{dw_1} = \frac{dE}{dh_1} \cdot \frac{dh_1}{dZh_1} \cdot \frac{dZh_1}{dw_1}$

Now let us do the working for the above 3 derivatives.

(1) $\frac{dE}{dw_3} = \frac{dE}{dO_1} \cdot \frac{dO_1}{dZO_1} \cdot \frac{dZO_1}{dw_3}$

where $\frac{dE}{dO_1} = O_1 - t_1 = 0.27$

$$\frac{dO_1}{dZO_1} = \frac{d}{dZO_1}\left(1/1+e^{-ZO_1}\right) = O_1(1-O_1) = 0.57(1-0.57)$$

$$= 0.57(0.43)$$

$$= 0.2451$$

$$\frac{dz_{o_1}}{dw_3} = \frac{d(w_3 h_1)}{dw_3} = h_1 = 0.75.$$

$$\therefore \quad \frac{dE}{dw_3} = (0.27)(0.24)(0.75) = 0.048$$

(2) $$\frac{dE}{dw_2} = \frac{dE}{dh_1} \cdot \frac{dh_1}{dz_{h_1}} \cdot \frac{dz_{h_1}}{dw_2}$$

where $$\frac{dE}{dh_1} = \frac{dE}{do_1} \cdot \frac{do_1}{dz_{o_1}} \cdot \frac{dz_{o_1}}{dh_1}$$ ( but $\frac{dz_{o_1}}{dh_1} = \frac{d((w_3)(h_1))}{dh_1} = w_3$ )

$$= (0.27)(0.24)(w_3)$$

$$= (0.27)(0.24)(0.4)$$

$$= 0.025$$

$$\therefore \quad \frac{dE}{dw_2} = \frac{dE}{dh_1} \cdot \frac{dh_1}{dz_{h_1}} \cdot \frac{dz_{h_1}}{dw_2}$$

$$= (0.025)\left[(z_{h_1})(1 - z_{h_1})\right] \cdot (x_2)$$

$$= (0.025)\left[(1.1)(1 - 1.1)\right] \cdot (3)$$

$$= (0.025)(1.1)(-0.1)(3)$$

$$= -0.008$$

(3) $\frac{dE}{dw_1} = \frac{dE}{dh_1} \cdot \frac{dh_1}{dzh_1} \cdot \frac{dzh_1}{dw_1}$

$= (0.025)\left[(1.1)(1-1.1)\right](x_1)$

$= (0.025)(1.1)(-0.1)(1)$

$= -0.002$

Applying Gradient Descent Algorithm, we update the old weights. (Let $\alpha = 0.1$, the learning rate)

$w_1^* = w_1 - \alpha \frac{dE}{dw_1}$

$= 0.2 - \left[(0.1)(-0.002)\right] =$

$= 0.2002.$

$w_2^* = w_2 - \alpha \frac{dE}{dw_2}$

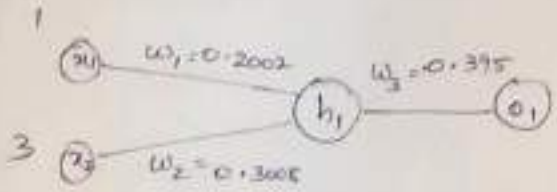
$= 0.3 - \left[(0.1)(-0.008)\right] = 0.3008.$

$w_3^* = w_3 - \frac{\alpha dE}{dw_3}$

$= 0.4 - \left[(0.1)(0.048)\right] = 0.395.$

Now applying the new weights to the network.

ANN Updated with new weights




Now again calculating feedforward & error

I/p to Hidden layer

(1) $w_1(x_1) + w_2(x_2) = zh_1$

$0.2002(1) + 0.3008(3) = 0.2002 + 0.9024$

$= 1.1026$

$h_1 = \alpha(zh_1)$

$= \alpha(1.1026) = 0.7507$

Hidden to O/p layer

$w_3(h_1) = zo_1$

$0.395(0.7507) = 0.296$

$\therefore o_1 = \alpha(zo_1) = \alpha(0.296) = 0.5735$

$E = \frac{1}{2}(o_1 - t_1)^2 = \frac{1}{2}(0.2735)^2 = \frac{0.0748}{2} = 0.0374$

In the 1st epoch the error was E=0.03765

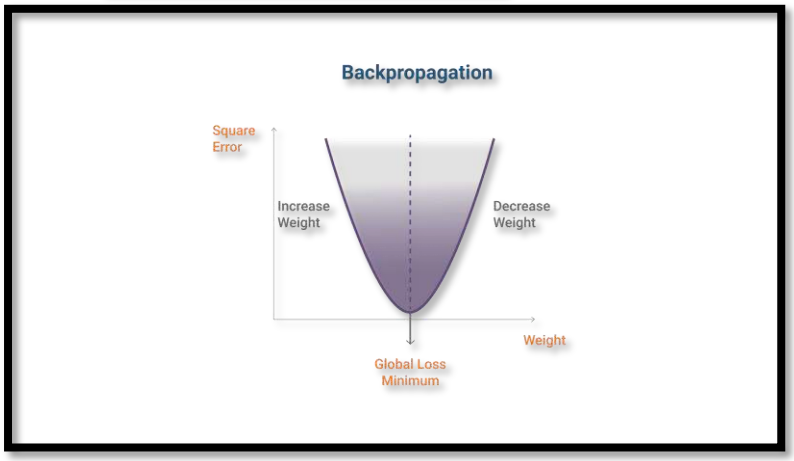
And now in the 2nd epoch the error got reduced ..and is now E=0.0374

In this way we repeat over and over many times until the error goes down and the parameter estimates stabilize or converge to some values using Back Propagation Algorithm.

## Gradient Descent:

- Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea is to tweak parameters iteratively in order to minimize the cost function
- An important parameter of Gradient Descent (GD) is the size of the steps, determined by the learning rate hyper parameters. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time, and if it is too high we may jump the optimal value.
- We calculate the gradient, ∂c/∂ω which is a partial derivative of cost with respect to weight.
  α is the learning rate, helps to adjust the weights with respect to gradient descent.

$$w := w - \alpha \frac{\partial c}{\partial \omega}$$




When the gradient is negative, an increase in weight decreases the error.
When the gradient is positive, the decrease in weight decreases the error.

Understanding Gradient Descent

- Gradient descent is by far the most popular optimization strategy used in Machine Learning and Deep Learning at the moment. It is used while training our model, can be combined with every algorithm, and is easy to understand and implement.

- Gradient measures how much the output of a function changes if we change the inputs a little.

- We can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope, and the faster the model learns.

$$b = a - \gamma \nabla f(a)$$

where,

b = next value

a = current value

'–' refers to the minimization part of the gradient descent.

$\gamma$ in the middle is the learning rate, and the gradient term $\nabla f(a)$ is simply the direction of the steepest descent.

- This formula basically tells us the next position where we need to go, which is the direction of the steepest descent.
- Gradient descent can be thought of as climbing down to the bottom of a valley instead of up a hill. This is because it is a minimization algorithm that minimizes a given function.

## Exploding of Gradient Descent:

The issue of exploding gradients can occur if the weights and inputs are large, causing the gradients to become increasingly large during backpropagation. If the gradients grow exponentially, they can lead to unstable updates of the weights and hinder the learning process.

Let's illustrate the exploding gradient problem with a simple example:

Consider a neural network with a single hidden layer consisting of two neurons (h1 and h2) and an output layer with a single neuron. We'll focus on the weights connecting the hidden layer to the output layer.

During backpropagation, the weight updates are calculated using the gradients of the loss with respect to the weights. Suppose the gradients for the weights are as follows:

Gradient for weight w1: ∇w1 = 100

Gradient for weight w2: ∇w2 = 200

Now, let's assume the current values of weights w1 and w2 are both initialized as 1. During weight update, we use a learning rate of 0.1.

The weight update equation is: new_weight = old_weight - learning_rate * gradient

Using this equation, let's calculate the updated weights:

Updated weight w1: new_w1 = 1 - 0.1 * 100 = 1 - 10 = -9

Updated weight w2: new_w2 = 1 - 0.1 * 200 = 1 - 20 = -19

As you can see, the weight updates are extremely large in magnitude, resulting in a significant change in the weights. This sudden change in weights can propagate through the network during subsequent iterations, causing the gradients to become even larger, and the weights to diverge further from the optimal values.

The exploding gradient problem can lead to several issues, such as unstable training dynamics, difficulty in finding the optimal solution, and slow convergence. It can also cause the network to fail to learn meaningful representations and result in poor performance on the task at hand.

Some of the suggested solutions to tackle the exploding gradient problem are given below:

- Use batch normalization
- Use less number of layers
- Carefully initialize weights
- Use gradient clipping:

## Vanishing gradient problem

The vanishing gradient problem occurs during training in neural networks when the gradients become extremely small, making the weight updates during backpropagation negligible. This can lead to slow convergence, difficulty in training deep networks, and an inability to effectively learn meaningful representations.

Let's illustrate the vanishing gradient problem with a simple example:

Consider a deep neural network with multiple hidden layers. For simplicity, let's focus on a network with three hidden layers (h1, h2, h3) and an output layer.

During backpropagation, the gradients for the weights are calculated based on the chain rule, starting from the output layer and propagating backward to the earlier layers. Suppose the gradients for the weights in the first hidden layer are as follows:

Gradient for weight w1: $\nabla w1 = 0.1$

Gradient for weight w2: $\nabla w2 = 0.2$

Now, let's assume the initial values of weights w1 and w2 are both initialized as 1. During weight update, we use a learning rate of 0.1.

The weight update equation is: new_weight = old_weight - learning_rate * gradient

Using this equation, let's calculate the updated weights:

Updated weight w1: new_w1 = 1 - 0.1 * 0.1 = 1 - 0.01 = 0.99

Updated weight w2: new_w2 = 1 - 0.1 * 0.2 = 1 - 0.02 = 0.98

As you can see, the weight updates are relatively small, reflecting the small magnitudes of the gradients. If the gradients in the subsequent layers are even smaller, the weight updates would be further reduced, potentially becoming negligible.

Here are some methods that are proposed to overcome the vanishing gradient problem:

1.Rectified linear unit (ReLU)
2. Multi-level hierarchy
3. Long Short Term Memory
4.Batch normalization
5.Weight initialization
6.Gradient Clipping

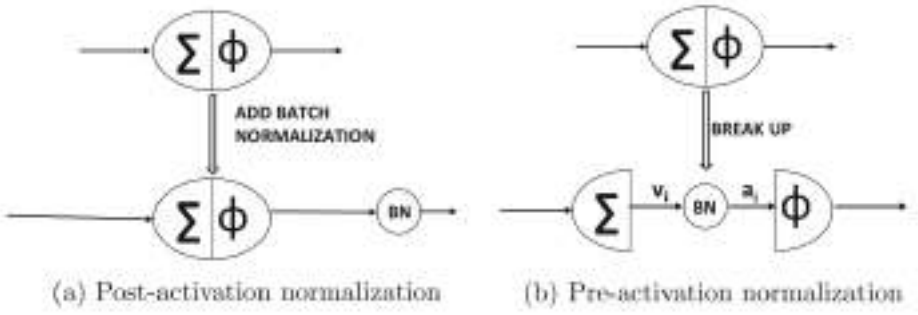
## Gradient Descent Strategies - Batch Normalization

### Batch Normalization

Batch normalization is a recent method to address the vanishing and exploding gradient problems, which cause activation gradients in successive layers to either reduce or increase in magnitude.Another important problem in training deep networks is that of internal covariate shift. The problem is that the parameters change during training, and therefore the hidden variable activations change as well. In other words, the hidden inputs from early layers to later layers keep changing. Changing inputs from early layers to later layers causes slower convergence during training because the training data for later layers is not stable.

Batch normalization helps address the internal covariate shift problem by normalizing the inputs of each neuron. By normalizing the inputs, batch normalization allows the network to focus on learning the important features of the data, which leads to faster convergence during training.

In batch normalization, the idea is to add additional "normalization layers" between hidden layers that resist this type of behaviour by creating features with somewhat similar variance.Here, we mention that there are two choices for where the normalization layer can be connected:




(a) Post-activation normalization (b) Pre-activation normalization

1. The normalization can be performed just after applying the activation function to the linearly transformed inputs. This solution is shown in Figure (a). Therefore, the normalization is performed on post-activation values.
2. The normalization can be performed after the linear transformation of the inputs, but before applying the activation function. This situation is shown in Figure (b). Therefore, the normalization is performed on pre-activation values.

The second choice has more advantages

In batch normalization, several key steps are performed during the training phase. Here's an overview of what happens in batch normalization:

Training Phase:

During the training phase, batch normalization operates on mini-batches of data.

a. Forward Pass:

- ✓ Input Calculation: For each neuron in the current layer, calculate the weighted sum of its inputs.
- ✓ Mean and Variance Calculation: Compute the mean and variance of the inputs for each neuron in the mini-batch.
- ✓ Normalization: Normalize the inputs of each neuron by subtracting the mean and dividing by the standard deviation (computed from the variance).
- ✓ Scale and Shift: Scale and shift the normalized inputs using learnable parameters called scale and shift parameters. This allows the network to learn the optimal scaling and shifting for the normalized inputs.

- ✓ Activation Calculation: Compute the activation of each neuron using the scaled and shifted normalized inputs.
- ✓ Pass the activations to the next layer.

b. Backward Pass:

- ✓ Gradient Calculation: During backpropagation, calculate the gradients of the loss function with respect to the activations of the current layer.
- ✓ Update Scale and Shift: Update the scale and shift parameters of batch normalization using the gradients computed in the previous step.

## Practical Tricks for Acceleration and Compression

Neural network learning algorithms can be extremely expensive, both in terms of the number of parameters in the model and the amount of data that needs to be processed. There are several strategies that are used to accelerate and compress the underlying implementations. Some of the common strategies are as follows:

1. GPU-acceleration: Graphics Processor Units (GPUs) have historically been used for rendering video games with intensive graphics because of their efficiency in settings where repeated matrix operations are required. Such repetitive operations are also used in neural networks, in which matrix operations are extensively used. Even the use of a single GPU can significantly speed up implementation because of its high memory bandwidth and multithreading within its multicore architecture.

2. Parallel implementations: One can parallelize the implementations of neural networks by using multiple GPUs or CPUs. Either the neural network model or the data can be partitioned across different processors. These implementations are referred to as model-parallel and data-parallel implementations.

3. Algorithmic tricks for model compression during deployment: A key point about the practical use of neural networks is that they have different computational requirements during training and deployment. While it is acceptable to train a model for a week with a large amount of memory, the final deployment might be performed on a mobile phone, which is highly constrained both in terms of memory and computational power.