Unit 3.
Classification- logistic regression & Neural Network

By Suchita patil

Logistic Regression

- **Definition:**

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.

Some of the examples of classification problems are

- ❏ Email spam or not spam,
- ❏ Online transactions Fraud or not Fraud,
- ❏ Tumor Malignant or Benign.

Types of logistic regression

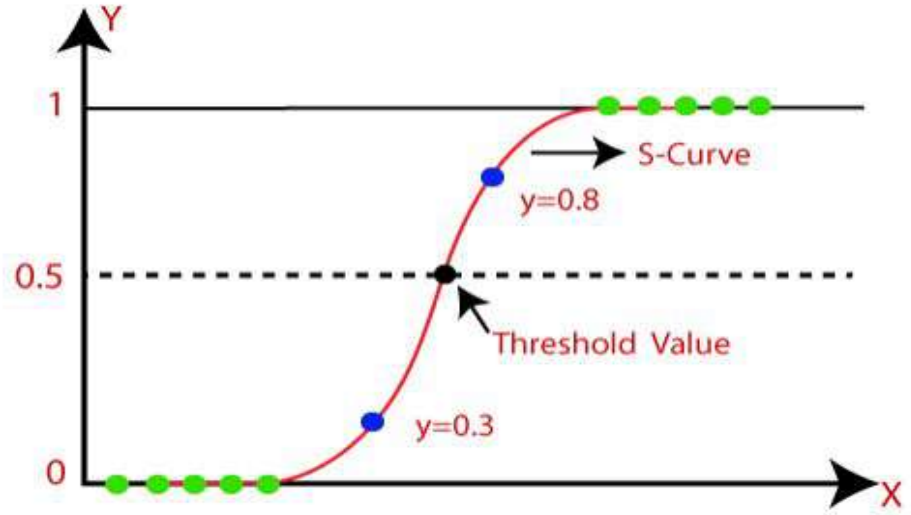
1. Binary classification (eg. Tumor Malignant or Benign)
2. Multiclass classification (eg. Cats, dogs or Sheep's)

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.

● Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

● Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Hypothesis Representation

- We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function' or also known as** the 'logistic function' instead of a linear function.
- The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression.

- We want,

$$0 \leq h_\theta(x) \leq 1$$

Logistic regression hypothesis expectation

So, hypothesis for logistic regression is,

- $h_\theta(x) = g(\theta^\top x)$,

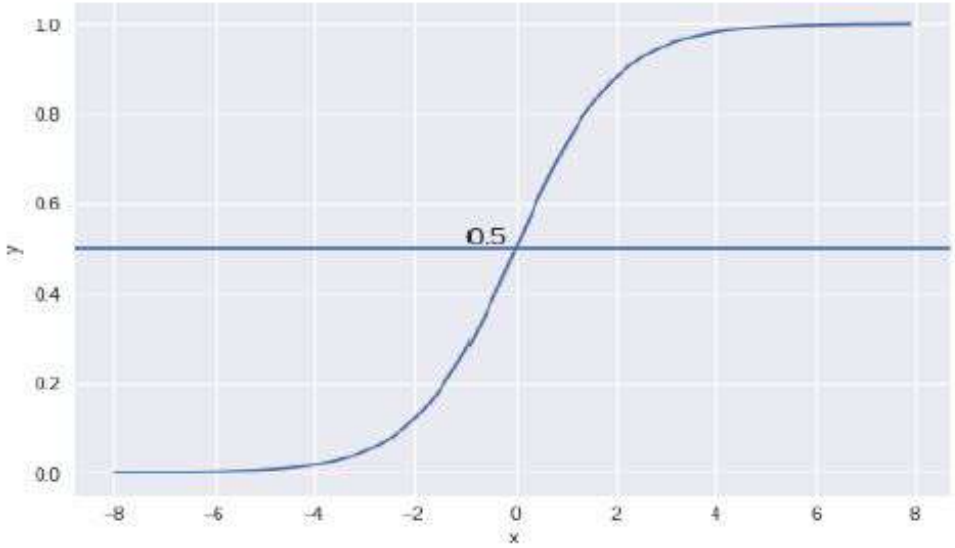
where $g(z) = \frac{1}{1+e^{-z}}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

Decision Boundary:

We expect our classifier to give us a set of outputs or classes based on probability when we pass the inputs through a prediction function and returns a probability score between 0 and 1.

For Example, We have 2 classes, let's take them like cats and dogs(1 — dog , 0 — cats). We basically decide with a threshold value above which we classify values into Class 1 and of the value goes below the threshold then we classify it in Class 2.



Example

As shown in the above graph we have chosen the threshold as 0.5, if the prediction function returned a value of 0.7 then we would classify this observation as Class 1(DOG). If our prediction returned a value of 0.2 then we would classify the observation as Class 2(CAT).

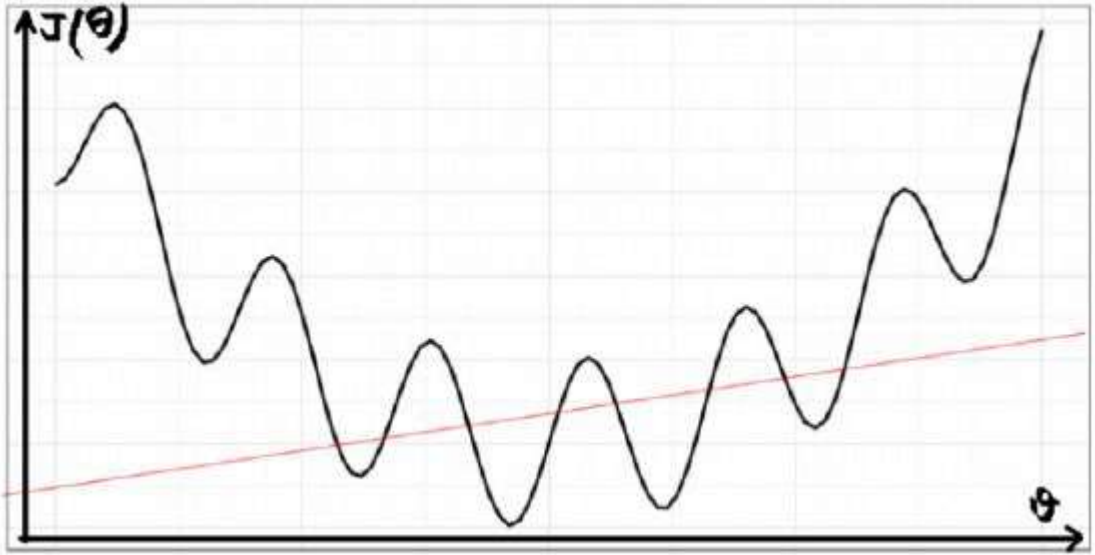
Cost Function

The cost function represents optimization objective i.e. we create a cost function and minimize it so that we can develop an accurate model with minimum error.

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

The Cost function of Linear regression

If we try to use the cost function of the linear regression in 'Logistic Regression' then it would be of no use as it would end up being a **non-convex** function with many local minimums, in which it would be very **difficult** to **minimize the cost value** and find the global minimum.



For logistic regression, the Cost function is defined as:

$-log(h\theta(x))$ *if* $y = 1$

$-log(1-h\theta(x))$ *if* $y = 0$

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Cost function of Logistic Regression

The above two functions can be compressed into a single function i.e.

$$J(\theta) = -\frac{1}{m}\sum\left[y^{(i)}\log(h\theta(x(i))) + \left(1 - y^{(i)}\right)\log(1 - h\theta(x(i)))\right]$$

Above functions compressed into one cost function

Gradient Descent for logistic regression

Now the question arises, how do we reduce the cost value. Well, this can be done by using **Gradient Descent.** The main goal of Gradient descent is to **minimize the cost value.** i.e. min J($\boldsymbol{\theta}$).

Now to minimize our cost function we need to run the gradient descent function on each parameter i.e.

$$\theta j := \theta j - \alpha \frac{\partial}{\partial \theta j} J(\theta)$$

Objective: To minimize the cost function we have to run the gradient descent function on each parameter

Want $\min_\theta J(\theta)$:

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

}

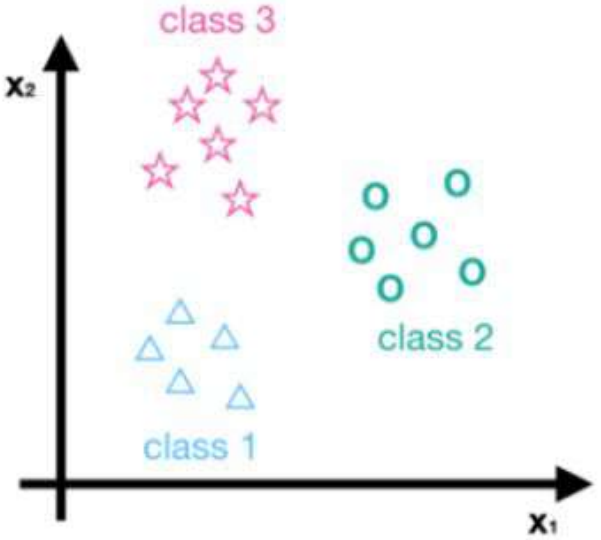
Though algorithm looks identical to linear regression, here

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

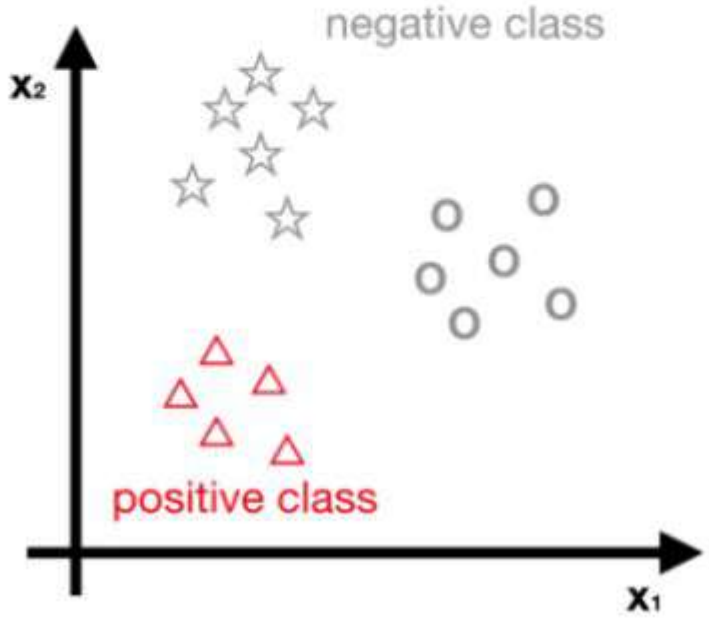
Multi-class Classification

- A multi-class classification is a classification technique that allows us to categorize data with more than two class labels.
- Trained multi-class classifiers are able to predict labels for test data based on those that are present in training data.
- One-Vs-All or One-Vs-Rest Classification is a method of multi-class classification. It can be broken down by splitting up the multi-class classification problem into multiple binary classifier models.
- For k class labels present in the dataset, k binary classifiers are needed in One-vs-All multi-class classification.

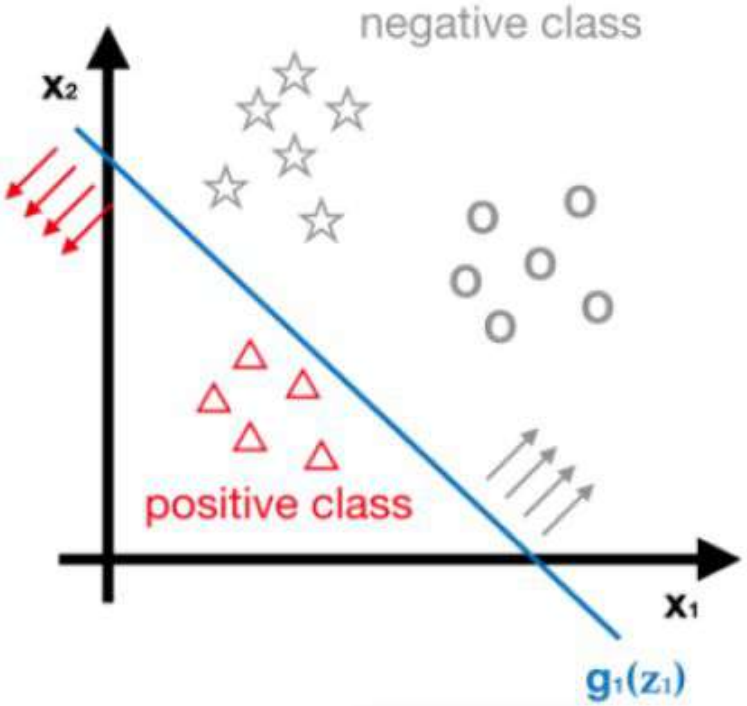
- The core idea of one-vs-all method is to classify the dataset into positive class and negative class (i.e. goal class and the other classes) at each time, and then to get the probability of being the positive class. After calculating the probabilities of being all positive classes, we choose the maximum one as the predictor's result.
- For example, suppose we have a dataset with three classes.
  - Class1 (△)
  - Class2 (◯)
  - Class3 (☆)



Step1 - Choosing the class1 as positive class, and the other classes are negative class.

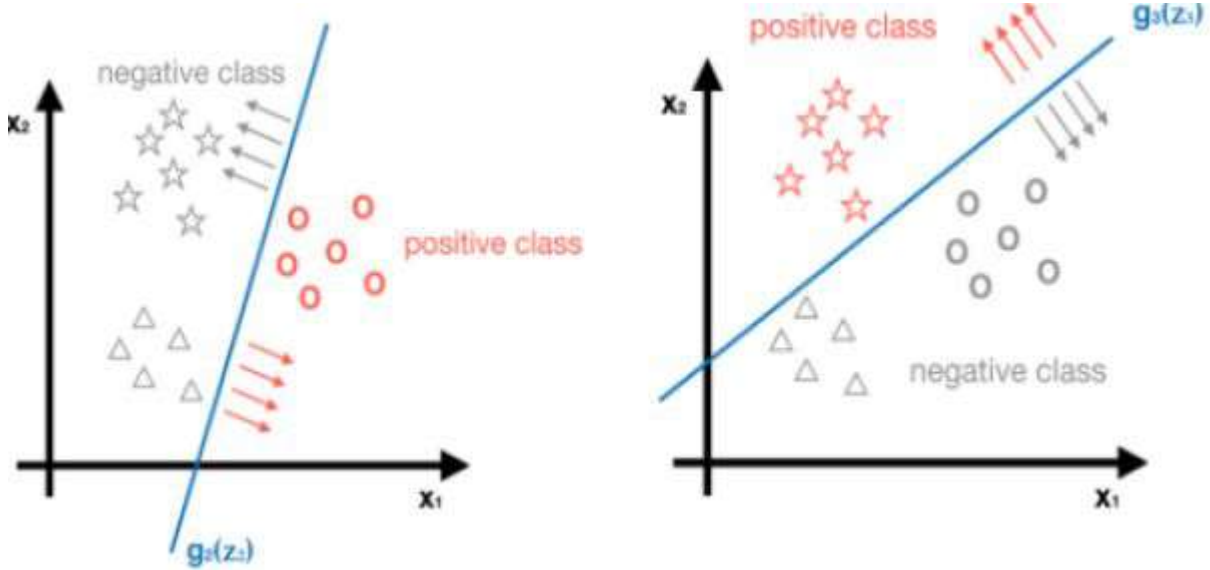


Step2 - Suppose we get a predictor g1(z1) after learning process, we use this predictor to estimate the probability of being class1 (△)

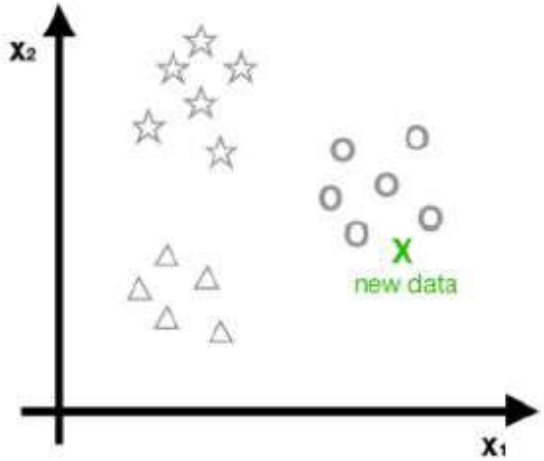


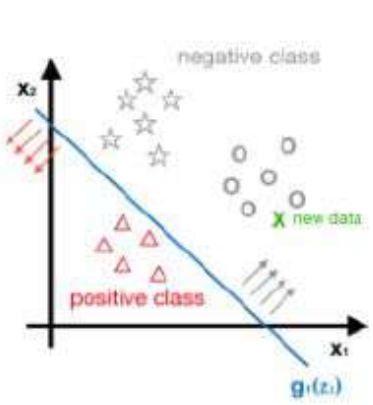
Step3 - Repeat step1 and step2, but set a different positive class.

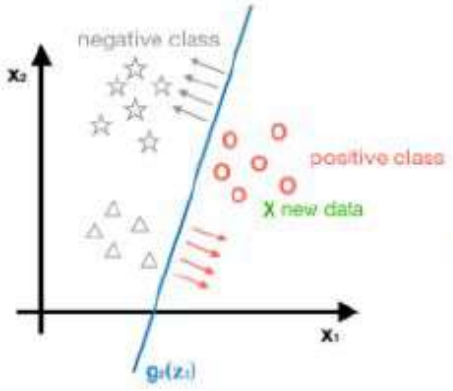
Finally, we can get g2(z2) and g3(z3), as shown in the following picture. We use predictor g2 and g3 to estimate the probability of being class2 (◯) and class3 (☆) respectively.

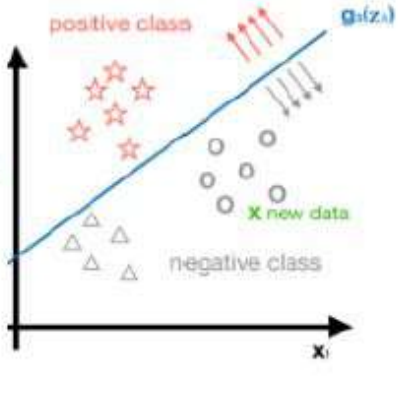


Step4 - Predicting the class of new data by using three predictors, g1, g2 and g3.

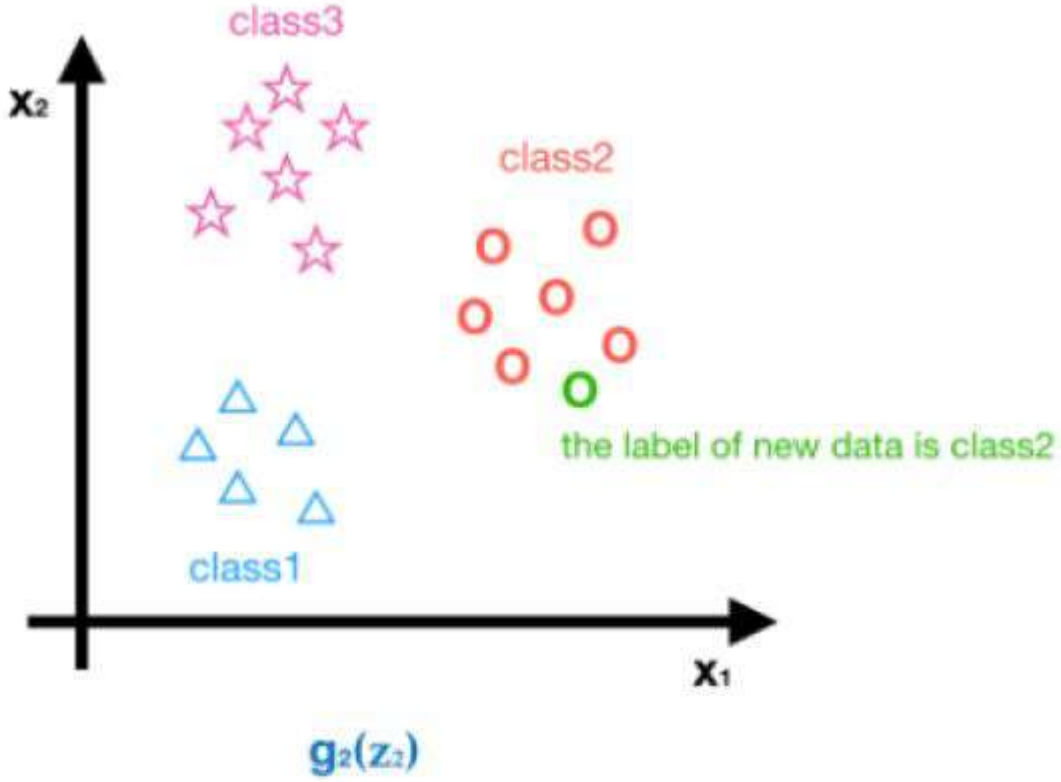








As you can see, the probabilities we get for the new data are $g1<0.5$, $g2>0.5$ and $g3<0.5$. The maxima is $g2>0.5$, Hence, we can conclude the predict label of this new data is class2 (◯).



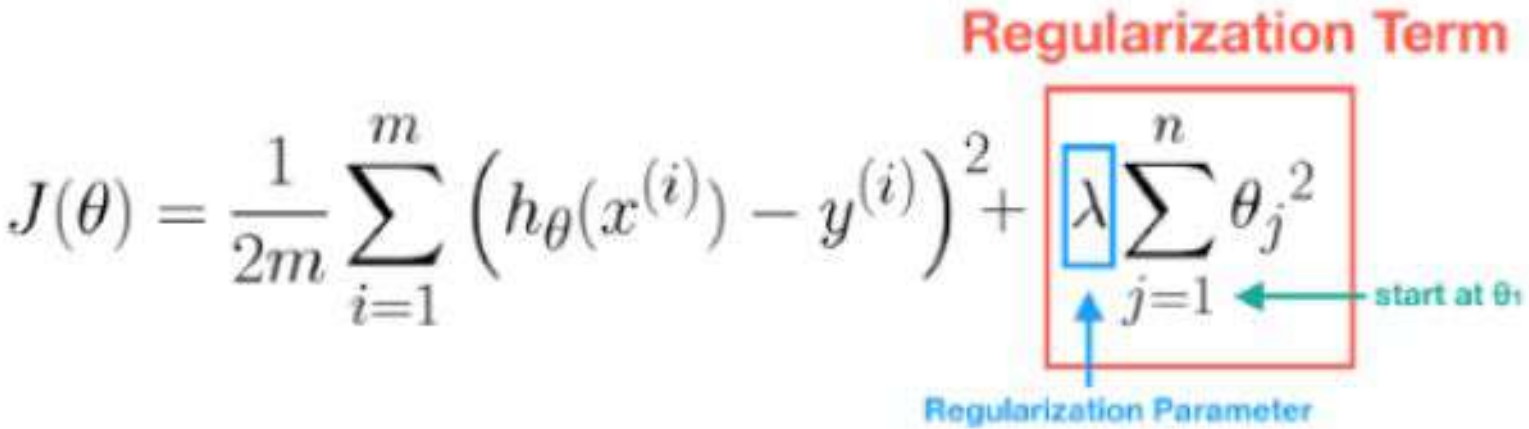
Regularization

- A 'good' model is the one with least training error.
- First, define a hypothesis function H (i.e. model).
- Second, define a Cost Function $J(\theta)$ for error measuring.
- Third, learn parameters of H by minimizing $J(\theta)$.
- In the end, we can get a learned model with least training error.
- In the extreme case, this 'good' model has zero training error and fits training data perfectly, that is, the predict label of all training data is equal to their corresponding truth label.
- However, when new data (green point) comes in, this model has terrible performance with huge prediction error.
- The problem here is called Overfitting. A truly good model must have both little training error and little prediction error.

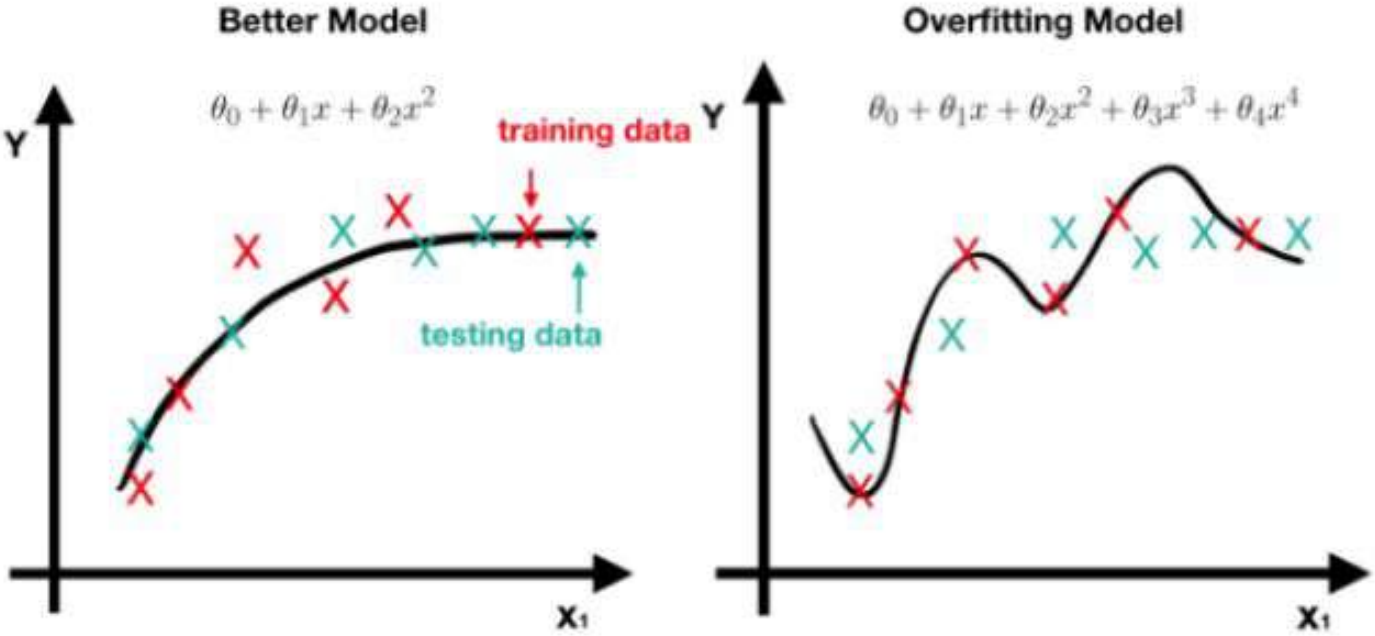
- **Overfitting**
  The learned model works well for training data but terrible for testing data (unknown data). In other words, the model has little training error but has huge perdition error.
  One way to avoid it is to apply Regularization and then we can get a better model with proper features.

- **Regularization**
  It's a technique applied to Cost Function $\boldsymbol{J(\theta)}$ *in order to avoid Overfitting.*
- The core idea in Regularization is to keep more important features and ignore unimportant ones. The importance of feature is measured by the value of its parameter $\boldsymbol{\theta j}$.

- *In linear regression, we modify its cost function by adding regularization term. The value of θj is controlled by regularization parameter λ.*



For instance, if we want to get a better model instead of the overfitting one. We don't need features X^3 and X^4 since they are unimportant. The procedure describes below.



First, we modify the Cost Function ***J(θ)*** *by adding regularization.*
Second, apply gradient descent in order to minimize ***J(θ)*** *and get the values of* ***θ3*** *and* ***θ4.***
After the minimize procedure, the values of ***θ3*** *and* ***θ4*** *must be near to zero if* ***λ=1000.***

Regularization Term

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \boxed{1000{\theta_3}^2 + 1000{\theta_4}^2}$$

$$\min_{\theta} J(\theta), \text{ getting } \theta_3 \approx 0, \theta_4 \approx 0$$

- *If λ is too large, then all the values of θ may be near to zero and this may cause Underfitting. In other words, this model has both large training error and large prediction error. (Note that the regularization term starts from $\theta 1$)*
- *If λ is zero or too small, its effect on parameters θ is little. This may cause Overfitting.*

There are two ***advantages of using regularization.***

- *The prediction error of the regularized model is lesser, that is, it works well in testing data.*
- *The regularization model is simpler since it has less features (parameters).*

Regularized linear regression

Note: Use above regularization theory along with its necessity and use below formulas of linear regression, to write about regularized linear regression.

Regularization Term

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \boxed{\lambda\sum_{j=1}^{n}\theta_j{}^2}$$

start at θ_1

Regularization Parameter

Repeat until converge {

$\frac{\partial}{\partial\theta_0}J(\theta)$

$$\theta_0 := \theta_0 - \alpha\boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0{}^{(i)}}$$

$\frac{\partial}{\partial\theta_j}J(\theta)$

$$\theta_j := \theta_j - \alpha\boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j{}^{(i)} + \boxed{\frac{\lambda}{m}\theta_j}} \qquad j = 1, 2, \ldots, n$$

regularization term

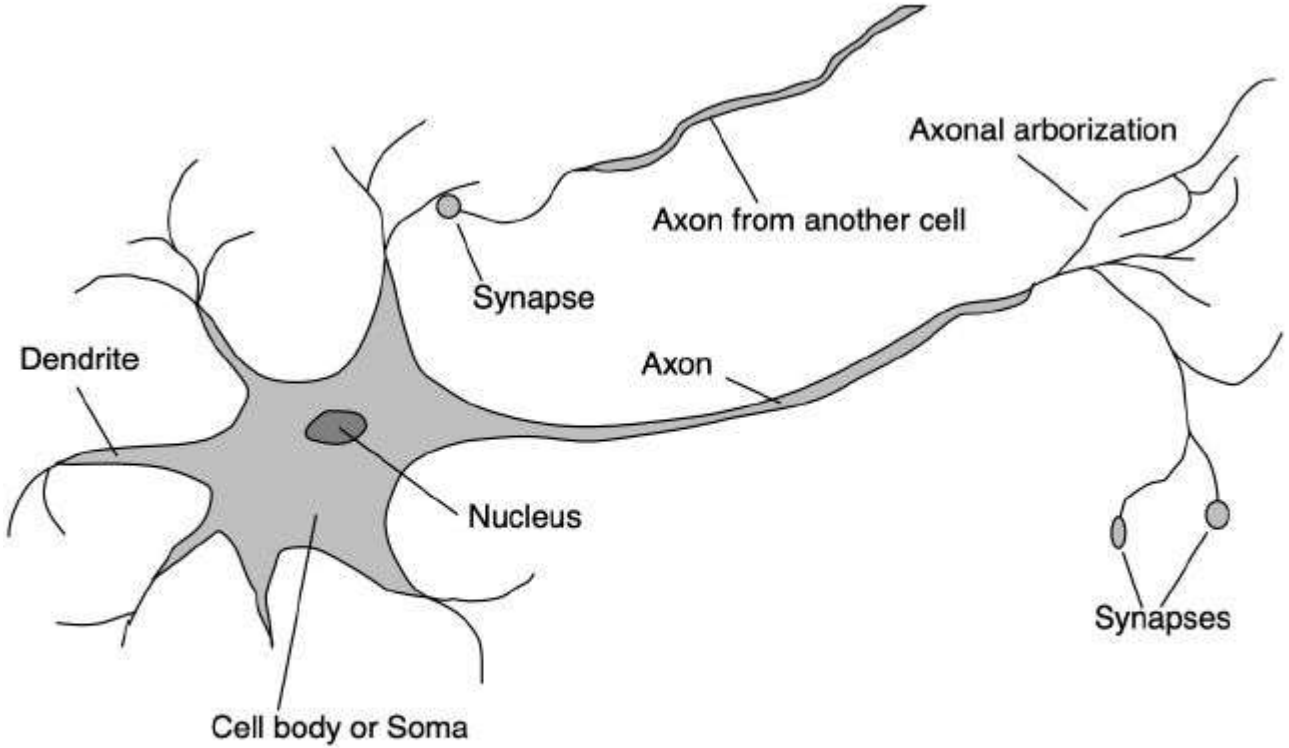
}

Neural networks

- Origins: Algorithms that try to mimic the brain.

What is this?

Neurons



10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time Signals of electrical potential

Model Representation

Artificial Neural Network is computing system inspired by biological neural network that constitute animal brain. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

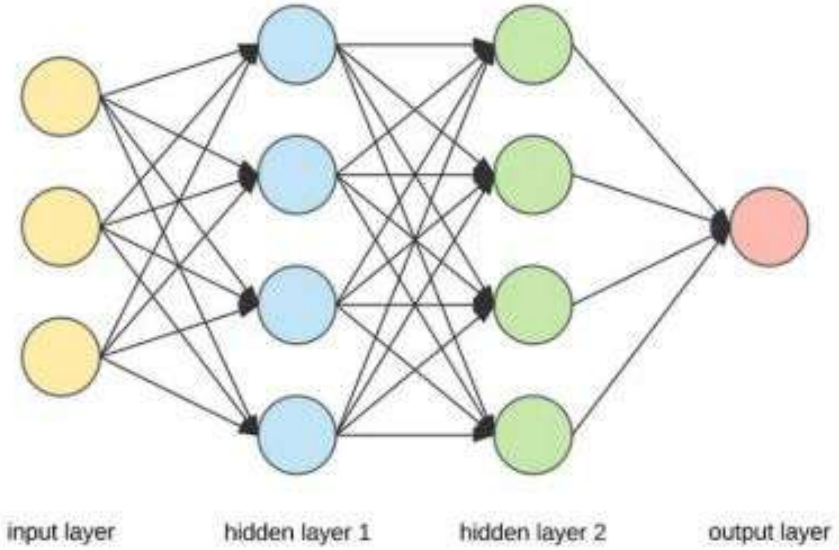


Image 1: Neural Network Architecture

The Neural Network is constructed from 3 type of layers:

1. **Input layer** — initial data for the neural network.
2. **Hidden layers** — intermediate layer between input and output layer and place where all the computation is done.
3. **Output layer** — produce the result for given inputs.

In above image there are 3 yellow circles. They represent the input layer and usually are noted as vector $\boldsymbol{X}$.

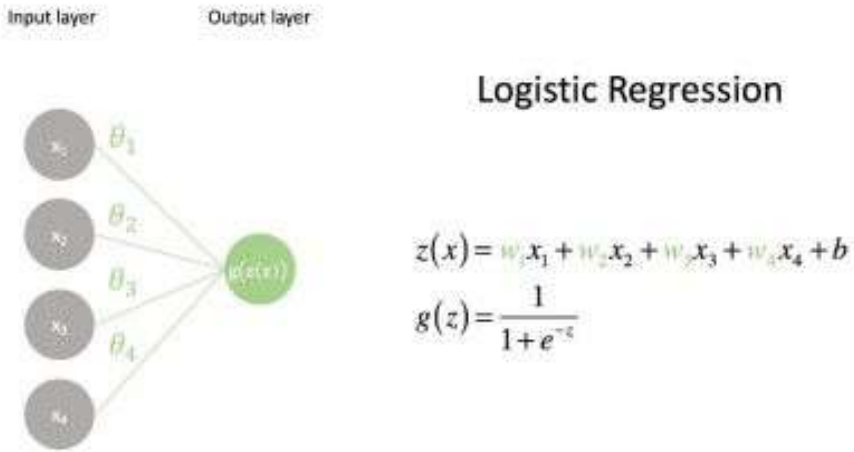
There are 4 blue and 4 green circles that represent the hidden layers. These circles represent the "activation" nodes and usually are noted as $\boldsymbol{W}$ or $\boldsymbol{\theta}$.

The red circle is the output layer or the predicted value.

A computational model of a neuron

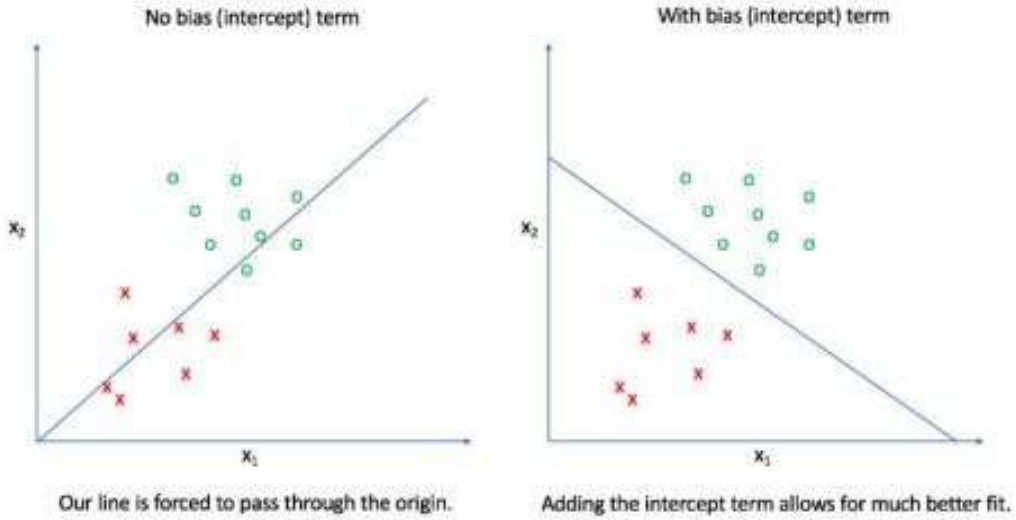
In logistic regression, we composed a linear model z(x) with the logistic function g(z) to form our predictor. This linear model was a combination of feature inputs x_i and weights w_i.

$$z(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b = w^{\mathrm{T}}x + b$$



The first layer contains a node for each value in our input feature vector. These values are scaled by their corresponding weight, w_i, and added together along with a bias term, b.

The bias term allows us to build linear models that aren't fixed at the origin. The following image provides an example of why this is important. Notice how we can provide a much better decision boundary for logistic regression when our linear model isn't fixed at the origin.



The input nodes in given network visualization are all connected to a single output node, which consists of a linear combination of all of the inputs.

Each connection between nodes contains a parameter, w, which is what we'll tune to form an optimal model.

The final output is functional composition, $g(z(x))$.

When we pass the linear combination of inputs through the logistic (also known as sigmoid) function, the neural network community refers to this as **activation**.

The sigmoid is an activation function which controls whether or not the end node "neuron" will fire.

Perceptron

Perceptron is a single layer neural network *and a multi-layer perceptron is called Neural Networks.*

Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.

The perceptron consists of 4 parts.

1. Input values or One input layer
2. Weights and Bias
3. Net sum
4. Activation Function

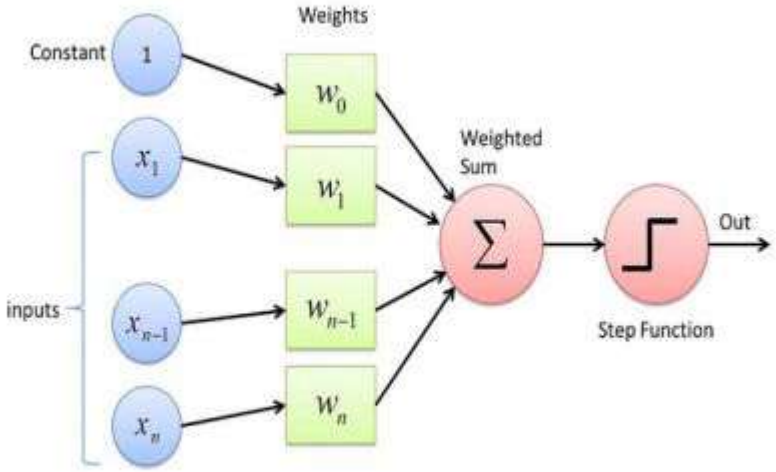


Fig : Perceptron

The additional node with value 1 is called “bias” node.

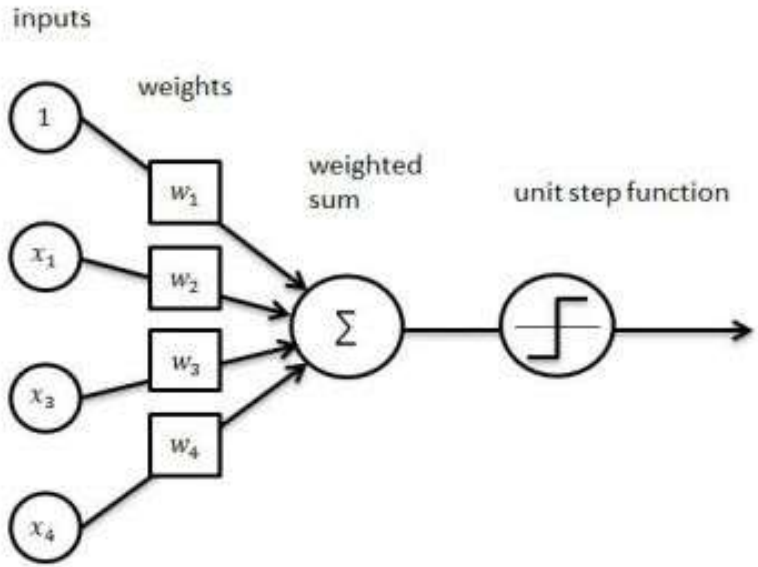


Image 2: Node from Neural Network

How does it work?

The perceptron is the simplest neural unit that we can build. It takes a series of inputs, x_i, combined with a series of weights, w_i, which are compared against a threshold value, Θ.

If the linear combination of inputs and weights is higher than the threshold, the neuron fires, and if the combination is less than the threshold it doesn't fire.

The perceptron works on these simple steps:

a. All the inputs ***x*** are multiplied with their weights ***w***. Let's call it ***k***.

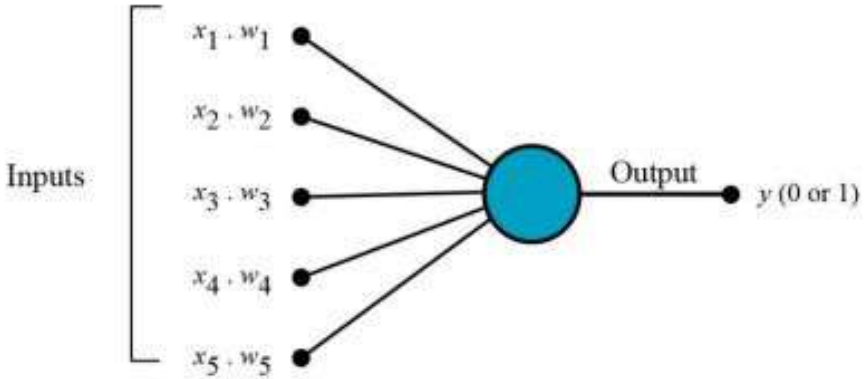


Fig: Multiplying inputs with weights for 5 inputs

b. ***Add*** all the multiplied values and call them ***Weighted Sum.***

c. ***Apply*** that weighted sum to the correct ***Activation Function***.
For Example: Unit Step Activation Function.

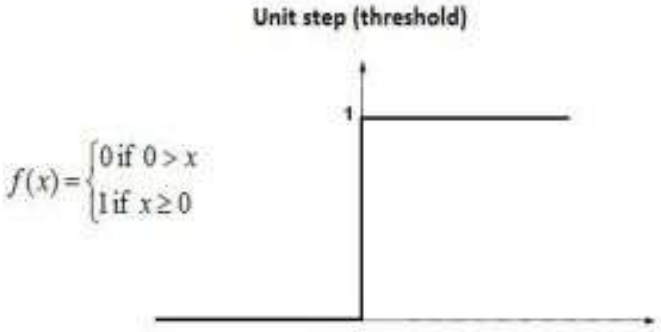


Fig: Unit Step Activation Function

The perceptron activation is a step-function from 0 (when the neuron doesn't fire) to 1 (when the neuron fires) while the logistic regression model has a smoother activation function with values ranging from 0 to 1.

Why do we need Weights and Bias?

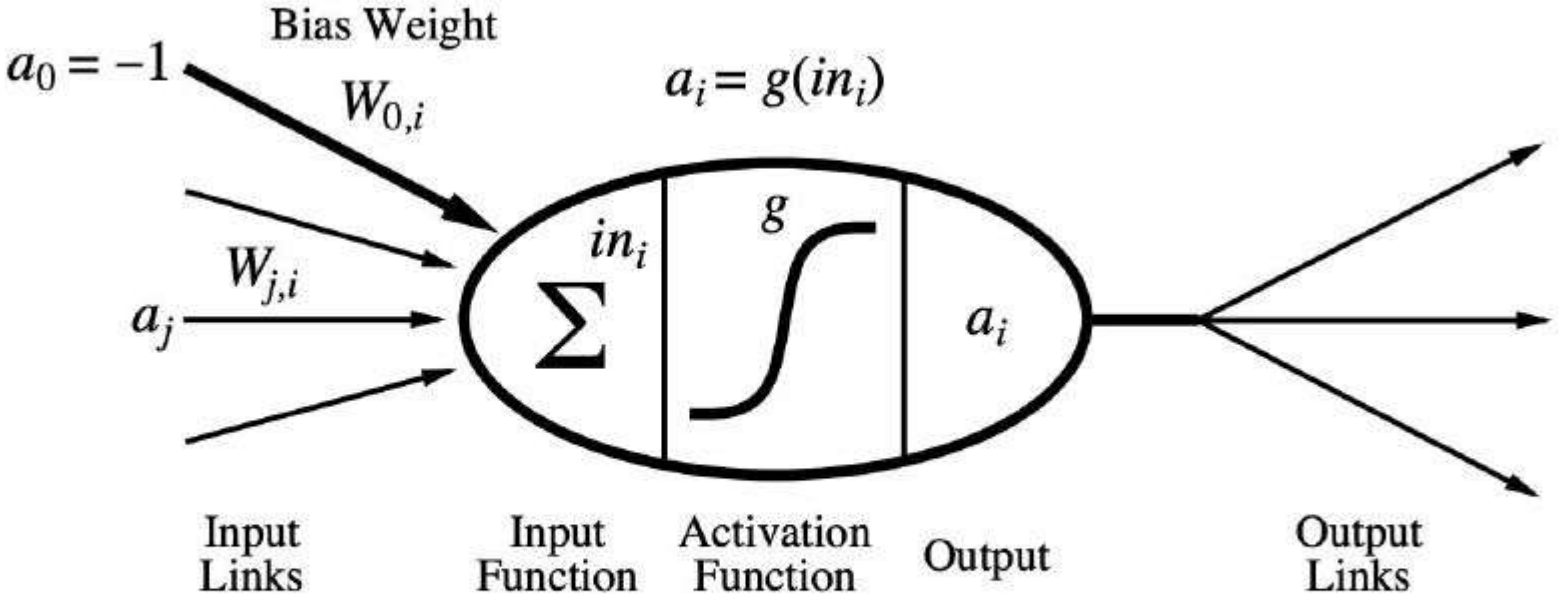
Weights shows the strength of the particular node.

A bias value allows you to shift the activation function curve up or down.

The activation functions are used to map the input between the required values like (0, 1) or (-1, 1).

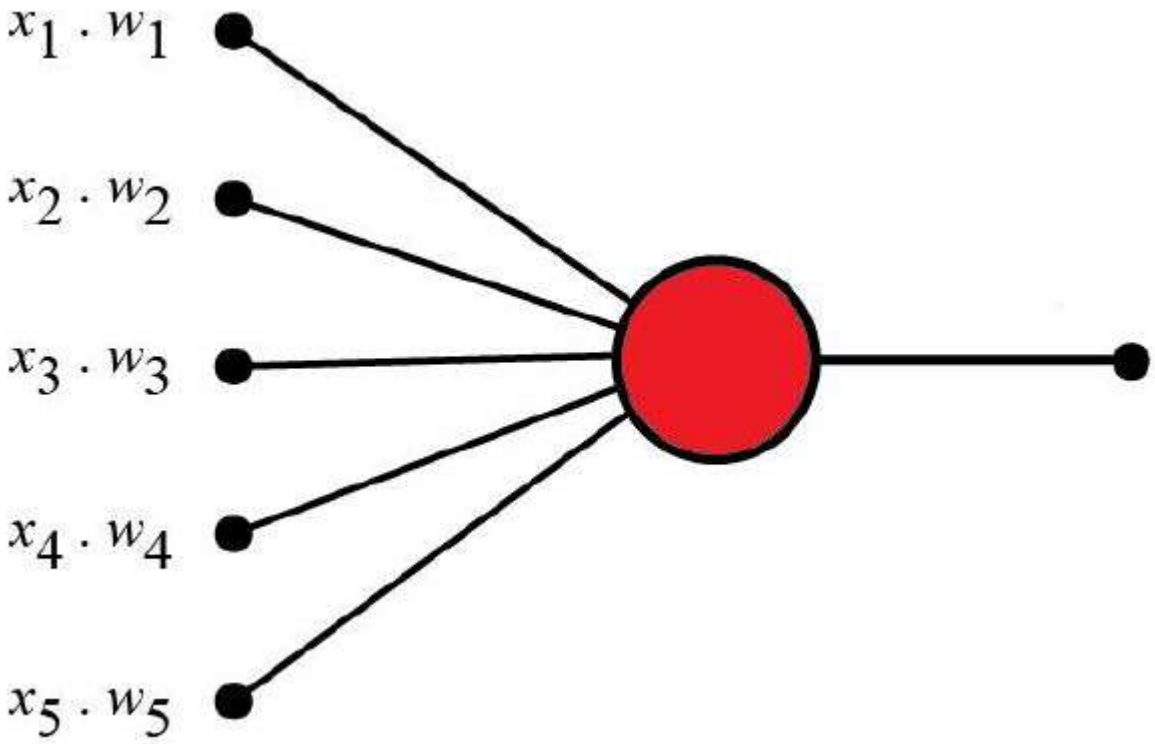
McCulloch–Pitts “unit”

Mankind’s First Mathematical Model of A Biological Neuron

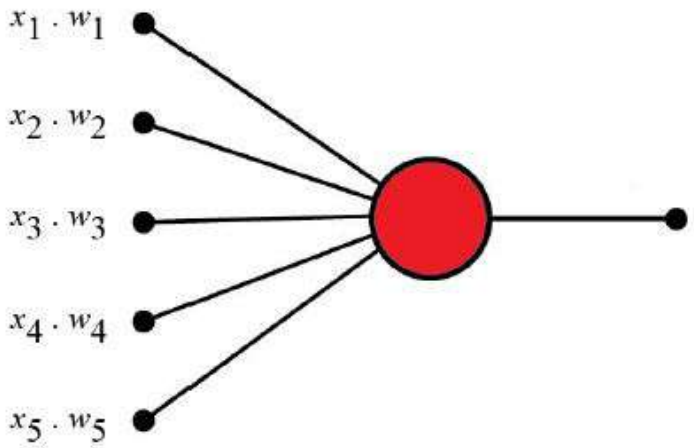


$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$

Perceptron

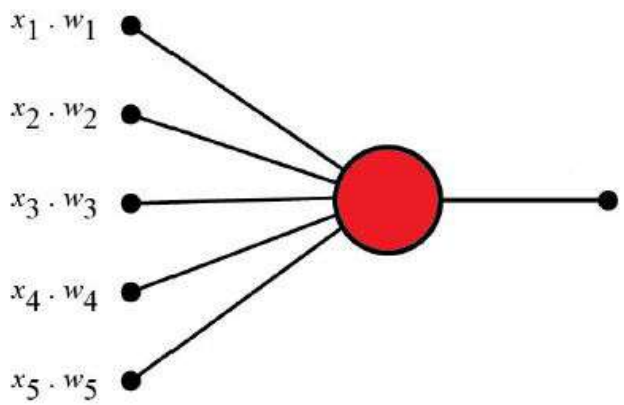


- The idea was to use different weights to represent the importance of each input, and
- Change the sum of the values into a binary value 0 or 1, using a activation function and/or a threshold value.

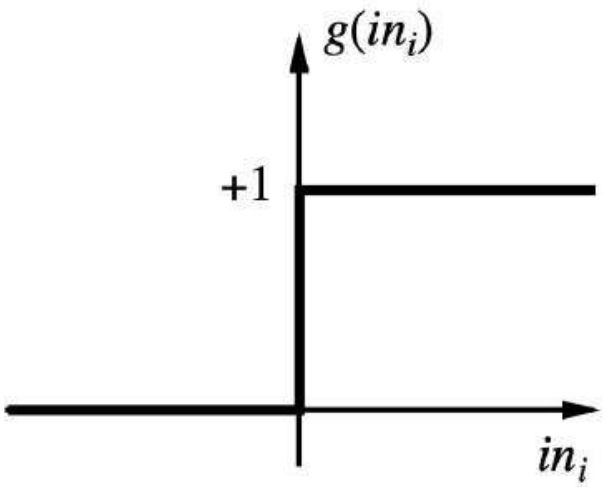


Basic perceptron algorithm:

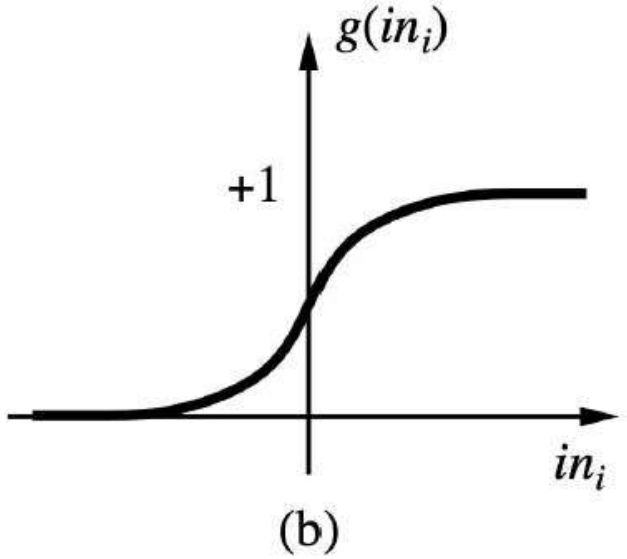
1. Decide an threshold value
2. Multiply all inputs with its weights
3. Sum all the results
4. Activate the output using the threshold value



Activation Functions



A step function or
threshold function



A sigmoid function
$1/(1 + e^{-x})$

Expressiveness of perceptrons

Network structures

Feed-forward networks:

- Single-layer perceptrons
- Multi-layer perceptrons

Feed-forward networks implement functions, have no internal state.

Recurrent networks:

- Recurrent neural networks have directed cycles with delays
  - have internal state (like flip-flops), can oscillate etc.

Feed-forward Neural Network:

Introduction:

Neural networks Feedforward is an artificial neural network that solves many problems, including image classification, natural language processing, and time series prediction. They are particularly effective for tasks involving pattern recognition.

These networks consist of interconnected "neurons" organized into layers, with the inputs passed through the first layer and the output produced by the final layer. There may also be any number of hidden layers between the input and output layers.

Each neuron has associated weights and biases that are adjusted during training to optimize the network's performance. Once trained, these networks can process new inputs and produce outputs based on their learned patterns.

What is a Feed-forward Neural Network?

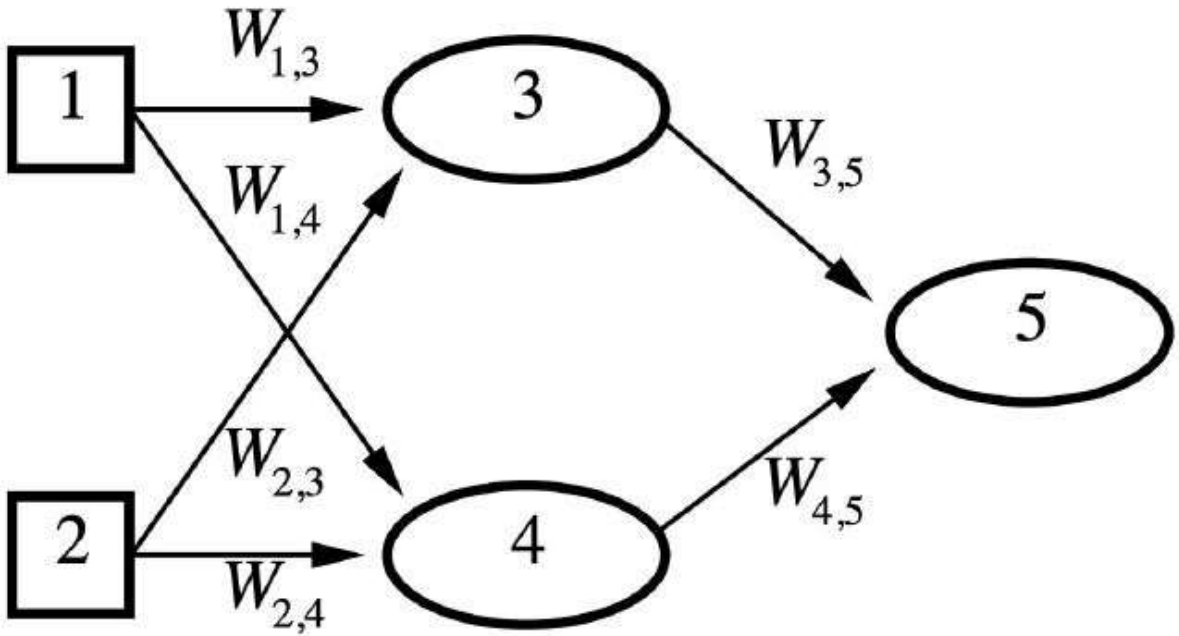
Neural networks feedforward, also known as multi-layered networks of neurons, are called "feedforwa where information flows in one direction from the input to the output layer without looping back. It is composed of three types of layers:

- Input Layer:
  The input layer accepts the input data and passes it to the next layer.

- Hidden Layers:
  One or more hidden layers that process and transform the input data. Each hidden layer has a set of neurons con to the neurons of the previous and next layers. These layers use activation functions, such as ReLU or sigmoid, t introduce non-linearity into the network, allowing it to learn and model more complex relationships between the and outputs.

- Output Layer:
  The output layer generates the final output. Depending on the type of problem, the number of neurons in the out layer may vary. For example, in a binary classification problem, it would only have one neuron. In contrast, a multi-class classification problem would have as many neurons as the number of classes.

Feed-forward - example



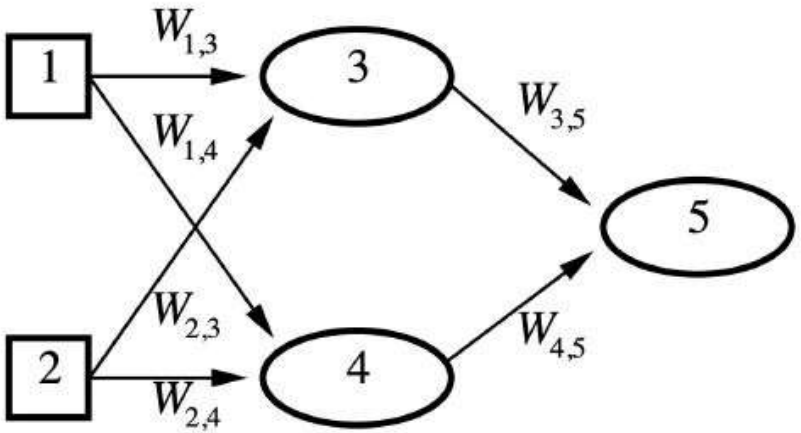
Feed-forward network = a parameterized family of nonlinear functions

Working

- The input to the network is a vector of values, x, which is passed through the network, layer by layer, and transformed into an output, y. The network's final output predicts the target function for the given input.
- The network makes this prediction using a set of parameters, θ (theta) or w, adjusted during training to minimize the error between the network's predictions and the target function.
- The training involves adjusting the θ (theta) values to minimize errors. This is done by presenting the network with a set of input-output pairs (also called training data) and computing the error between the network's prediction and the true output for each pair.

- This error is then used to compute the gradient of the error concerning the parameters, which tells us how to adjust the parameters to reduce the error. This is done using optimization techniques like gradient descent.

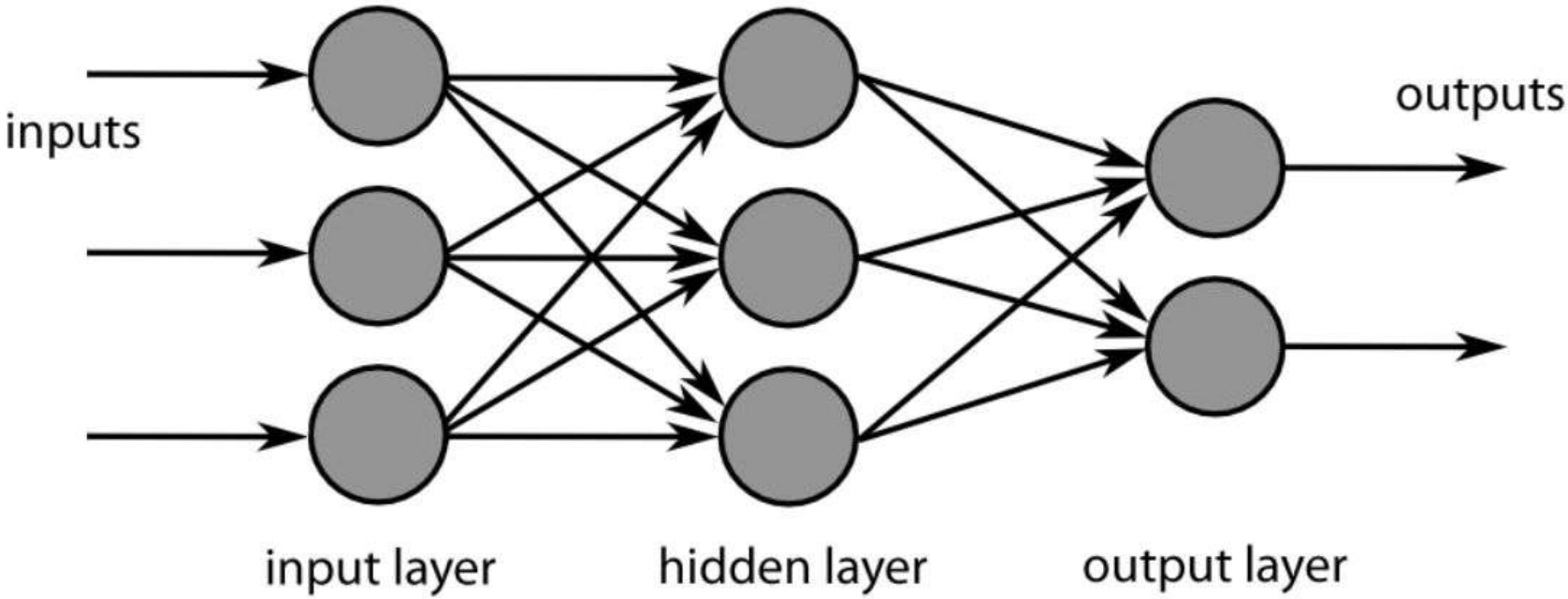
Feed-forward - example



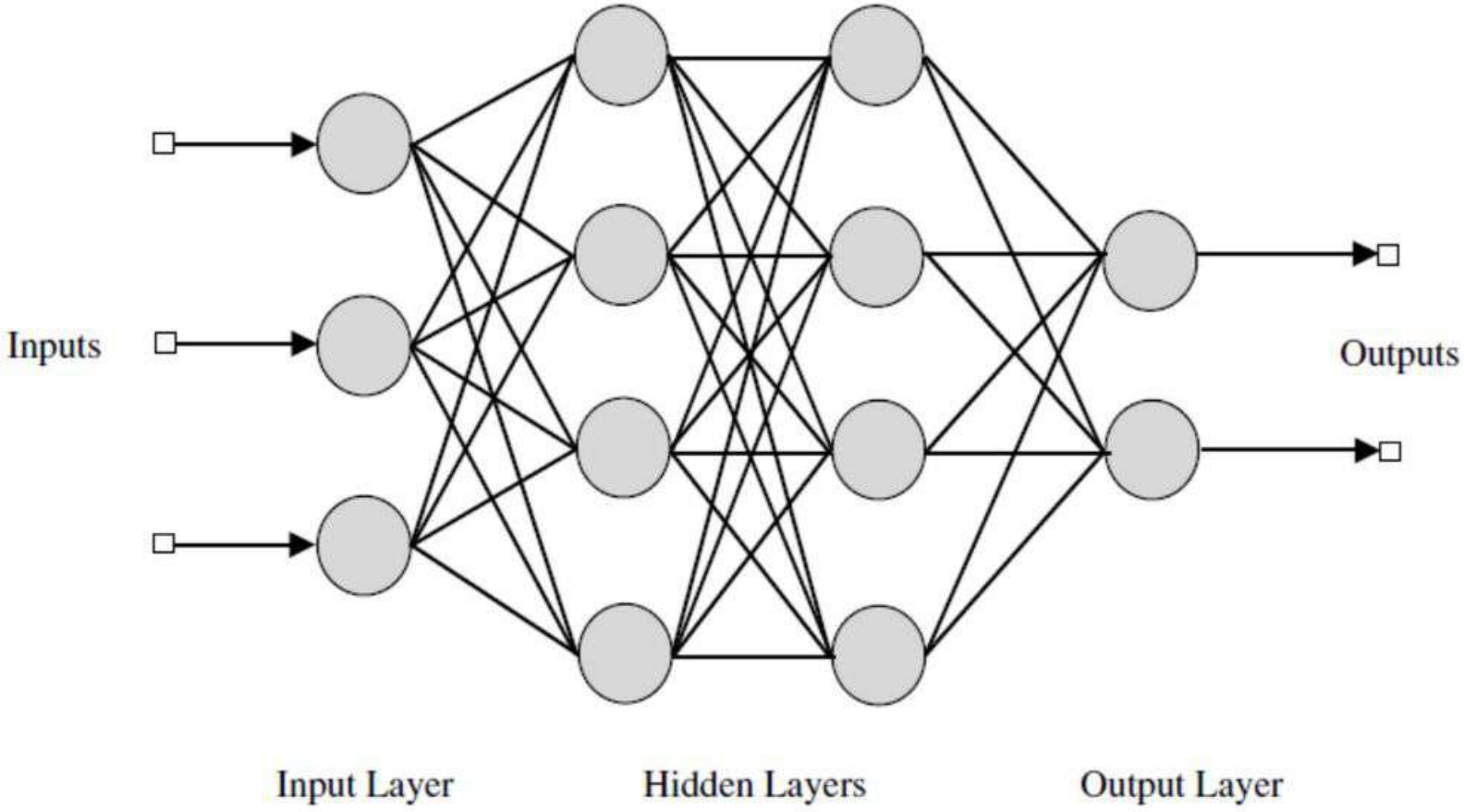
$$a5 = g(W3,5 \cdot a3 + W4,5 \cdot a4)$$
$$= g(W3,5 \cdot g(W1,3 \cdot a1 + W2,3 \cdot a2) + W4,5 \cdot g(W1,4 \cdot a1 + W2,4 \cdot a2))$$

Adjusting weights changes the function:
This is how machine LEARNING is represented!

Feed Forward Neural Network



Multilayer perceptrons

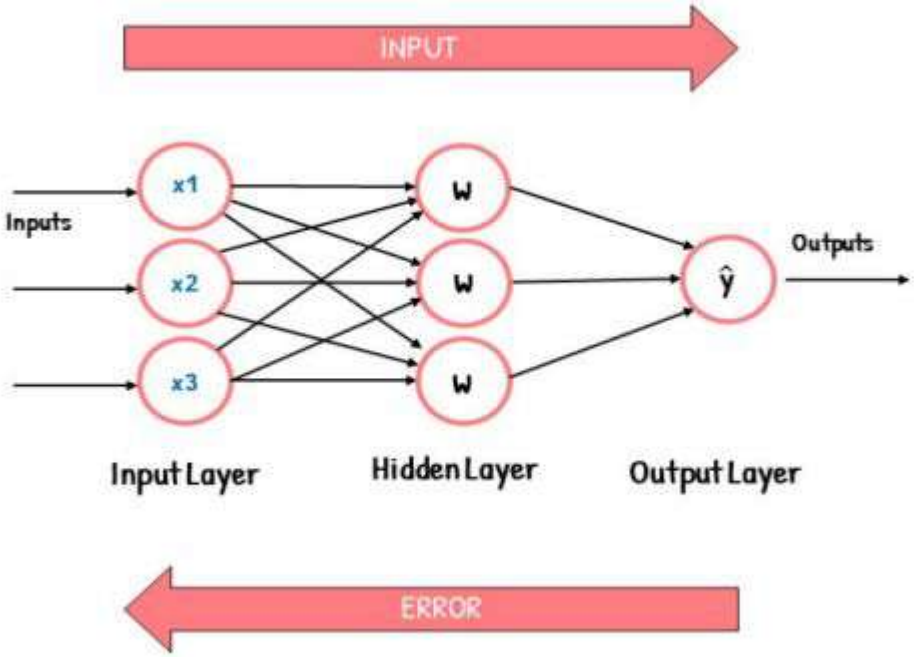


Backpropagation

- **Backpropagation** 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 make the model reliable by increasing its generalization.
- Backpropagation in neural network is a short form for “backward propagation of errors.” 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.

Backpropagation – Algorithm For Training A Neural Network

- Backpropagation is a supervised learning algorithm, for training Multi-layer Perceptrons (Artificial Neural Networks).
- It is a training algorithm used for training feedforward neural networks. It plays an important part in improving the predictions made by neural networks. This is because backpropagation is able to improve the output of the neural network iteratively.
- In a feedforward neural network, the input moves forward from the input layer to the output layer. Backpropagation helps improve the neural network's output. It does this by propagating the error backward from the output layer to the input layer.


INPUT
Inputs
x1
x2
x3
w
w
w
ŷ
Outputs
Input Layer
Hidden Layer
Output Layer
ERROR

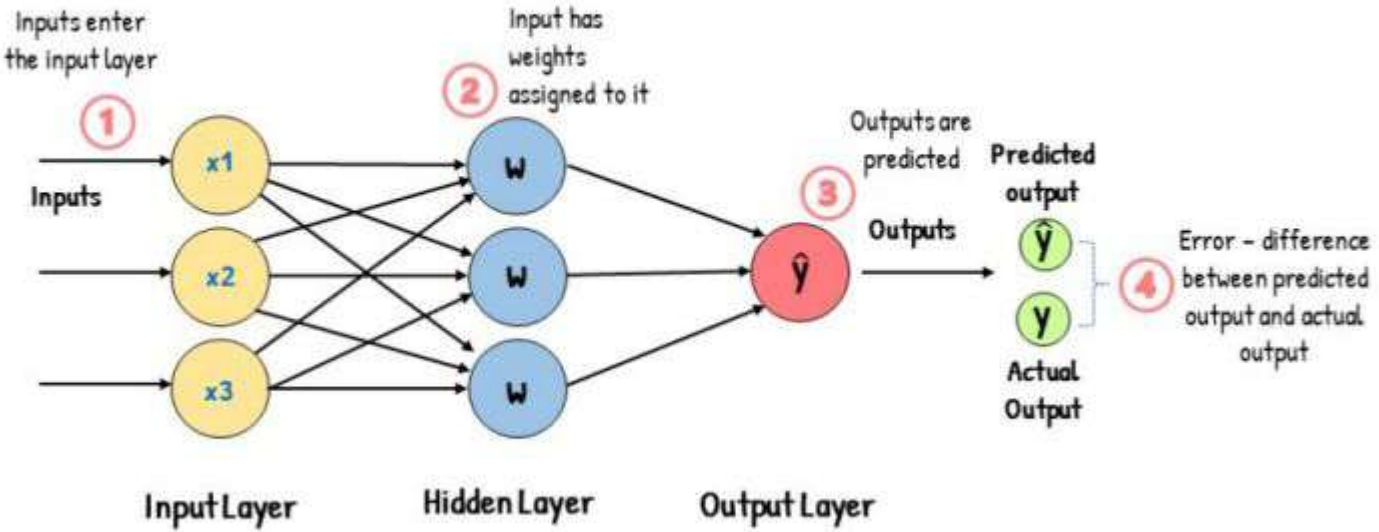
How Does Backpropagation Work?

To understand how backpropagation works, let's first understand how a feedforward network works.

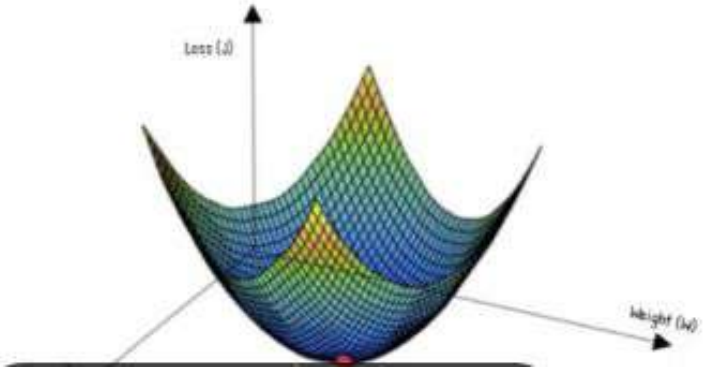
Feed Forward Networks

- A feedforward network consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the input into the neural network, and each input has a weight attached to it.
- The weights associated with each input are numerical values. These weights are an indicator of the importance of the input in predicting the final output. For example, an input associated with a large weight will have a greater influence on the output than an input associated with a small weight.
- When a neural network is first trained, it is first fed with input. Since the neural network isn't trained yet, we don't know which weights to use for each input. And so, each input is randomly assigned a weight. Since the weights are randomly assigned, the neural network will likely make the wrong predictions. It will give out the incorrect output.

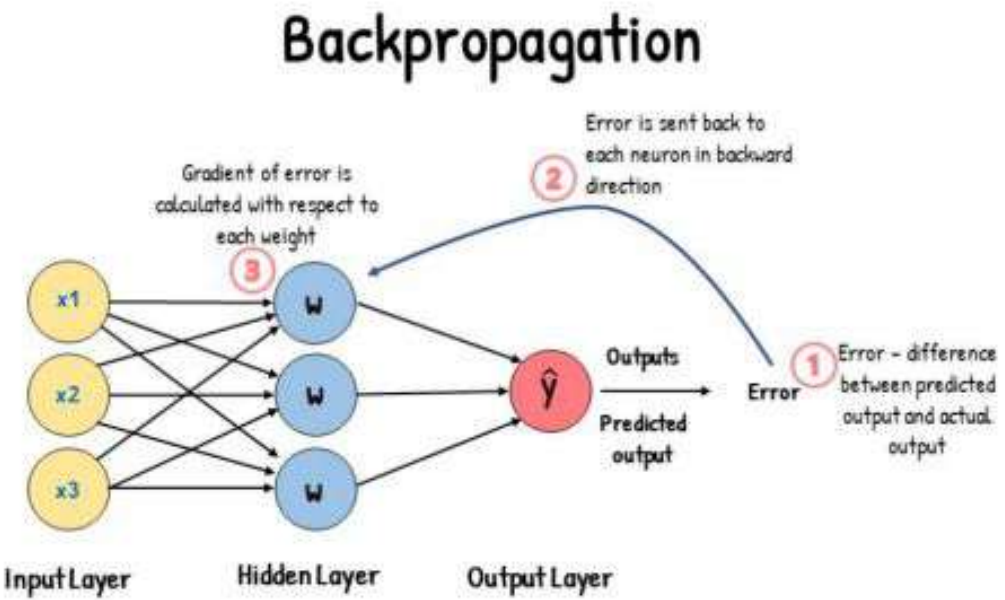
Feed-Forward Neural Network



- When the neural network gives out the incorrect output, this leads to an output error. This error is the difference between the actual and predicted outputs. A cost function measures this error.
- The cost function (J) indicates how accurately the model performs. It tells us how far-off our predicted output values are from our actual values. It is also known as the error. Because the cost function quantifies the error, we aim to minimize the cost function.
- We have to adjust the weights such that we have a combination of weights that minimizes the cost function.



Backpropagation allows us to readjust our weights to reduce output error. The error is propagated backward during backpropagation from the output to the input layer. This error is then used to calculate the gradient of the cost function with respect to each weight.



- Backpropagation aims to calculate the negative gradient of the cost function. This negative gradient is what helps in adjusting of the weights.
- Backpropagation uses the chain rule to calculate the gradient of the cost function. The chain rule involves taking the derivative. This involves calculating the partial derivative of each parameter. These derivatives are calculated by differentiating one weight and treating the other(s) as a constant. As a result of doing this, we will have a gradient.
- Since we have calculated the gradients, we will be able to adjust the weights.
- The weights are adjusted using a process called gradient descent.

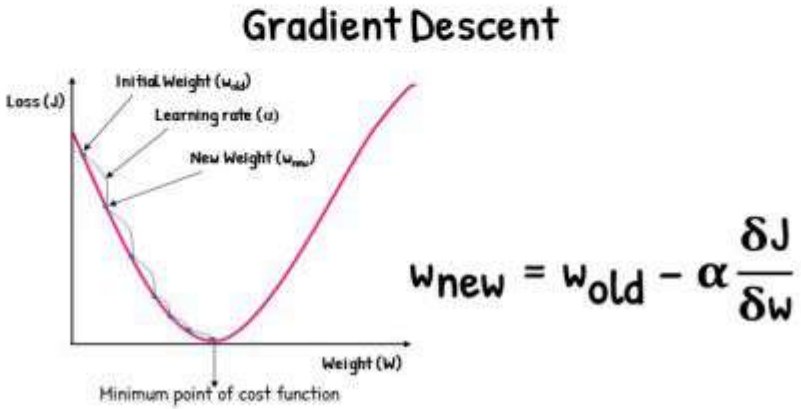
Descending the Cost Function

- Navigating the cost function consists of adjusting the weights. The weights are adjusted using the following formula:

$$w_{new} = w_{old} - \alpha \underbrace{\frac{dJ}{dw}}_{\text{Gradient}}$$

This is the formula for gradient descent.

Adjusting the weights consists of multiple iterations. We take a new step down for each iteration and calculate a new weight. Using the initial weight and the gradient and learning rate, we can determine the subsequent weights.



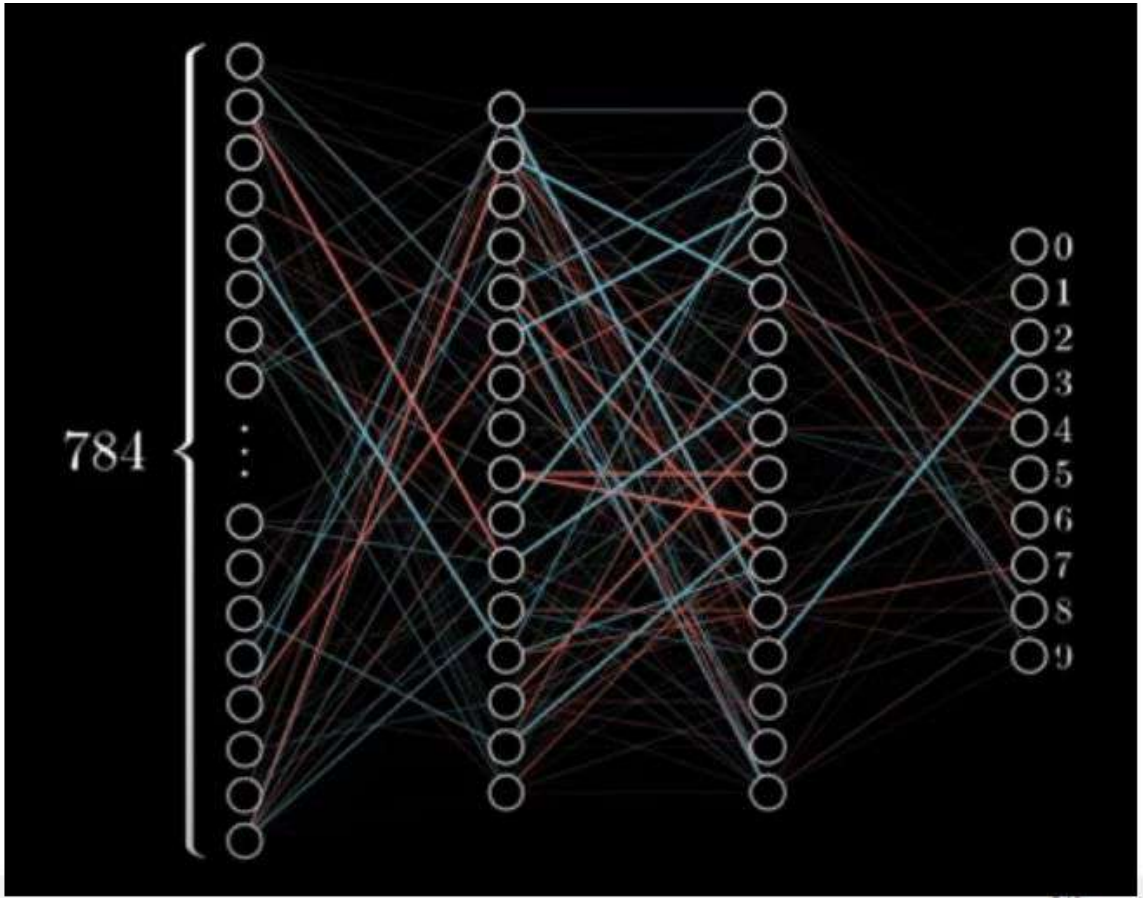
$$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$$

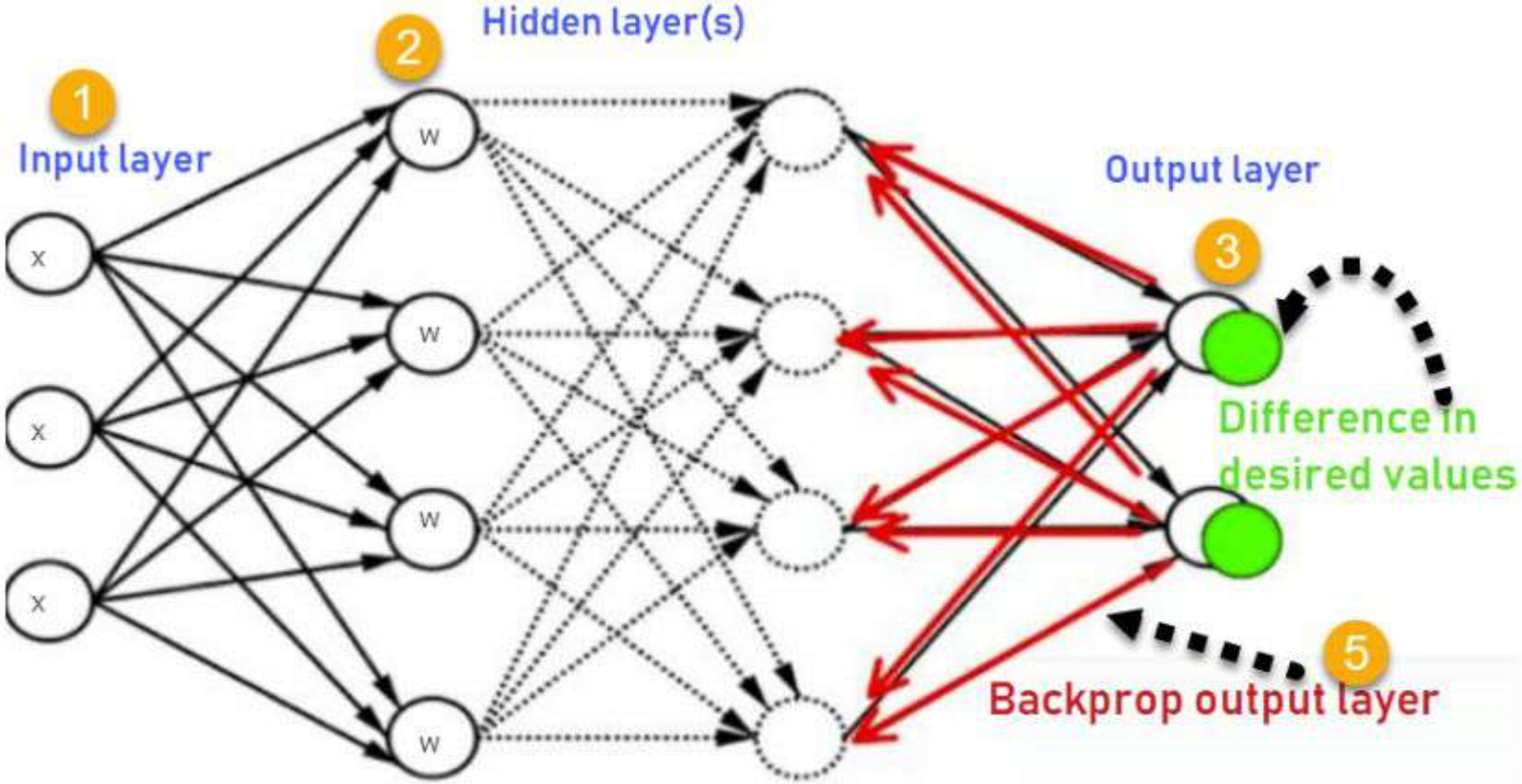
From the graph of the cost function, we can see that:

- To start descending the cost function, we first initialize a random weight.
- Then, we take a step down and obtain a new weight using the gradient and learning rate. With the gradient, we can know which direction to navigate. We can know the step size for navigating the cost function using the learning rate.
- We are then able to obtain a new weight using the gradient descent formula.
- We repeat this process until we reach the minimum point of the cost function.
- Once we've reached the minimum point, we find the weights that correspond to the minimum of the cost function.

Need for Backpropagation

- To minimize the errors
  - By adjusting the weights





How Backpropagation Algorithm Works