# Artificial neural network (ANN)

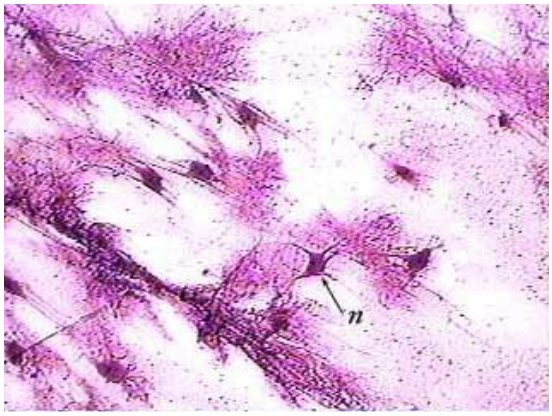
Artificial neural network (ANN) model involves computations and mathematics, which simulate the human–brain processes. Many of the recently achieved advancements are related to the artificial intelligence research area such as image and voice recognition, robotics, and using ANNs. The ANN models have the specific architecture format, which is inspired by a biological nervous system. Like the structure of the human brain, the ANN models consist of neurons in a complex and nonlinear form.

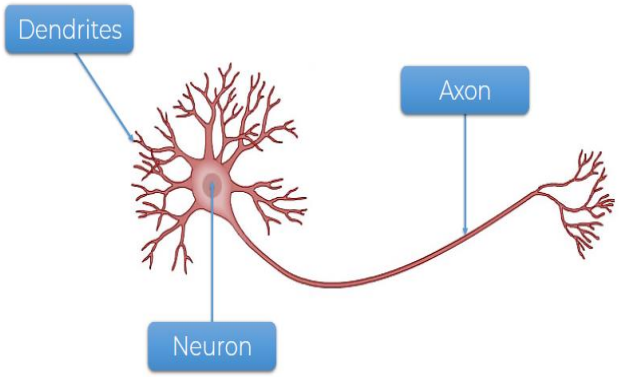
What we will learn in this section:
• The Neuron
• The Activation Function
• How do Neural Networks work? (example)
• How do Neural Networks learn?
• Gradient Descent
• Stochastic Gradient Descent
• Backpropagation

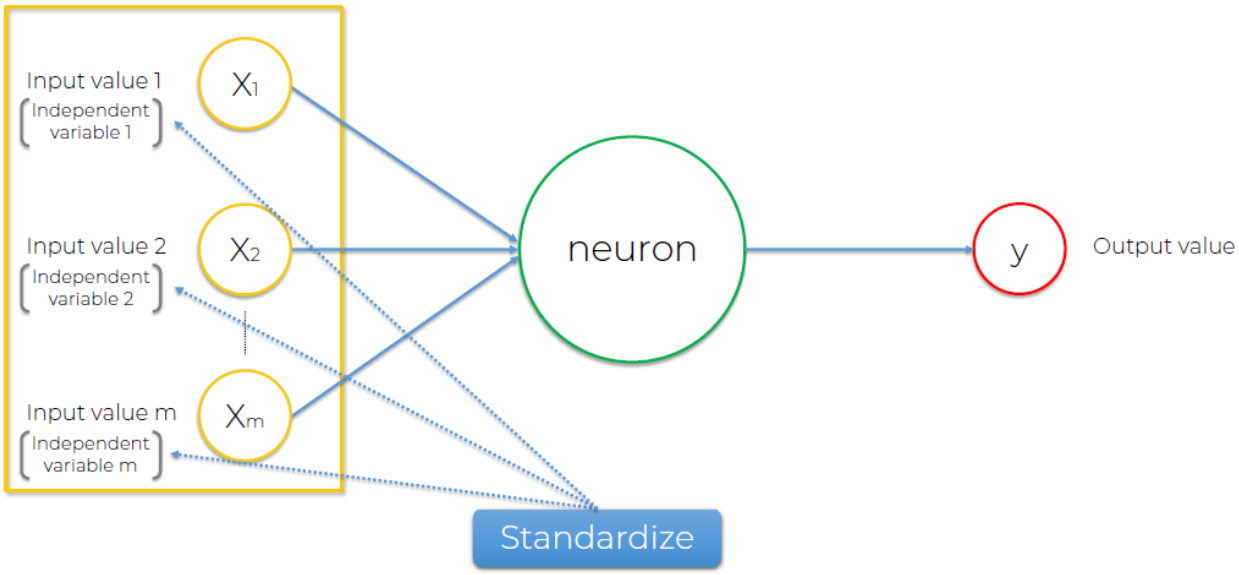
## The Neuron

Neurons (also called neurones or nerve cells) are the fundamental units of the brain and nervous system, the cells responsible for receiving sensory input from the external world, for sending motor commands to our muscles, and for transforming and relaying the electrical signals at every step in between. A neuron has three main parts: dendrites, an axon, and a cell body or soma. A dendrite (tree branch) is where a neuron receives input from other cells. The axon (tree roots) is the output structure of the neuron; when a neuron wants to talk to another neuron, it sends an electrical message called an action potential throughout the entire axon. The soma (tree trunk) is where the nucleus lies, where the neuron's DNA is housed, and where proteins are made to be transported throughout the axon and dendrites.

Artificial neurons are software modules, called nodes, and artificial neural networks are software programs or algorithms that, at their core, use computing systems to solve mathematical calculations.





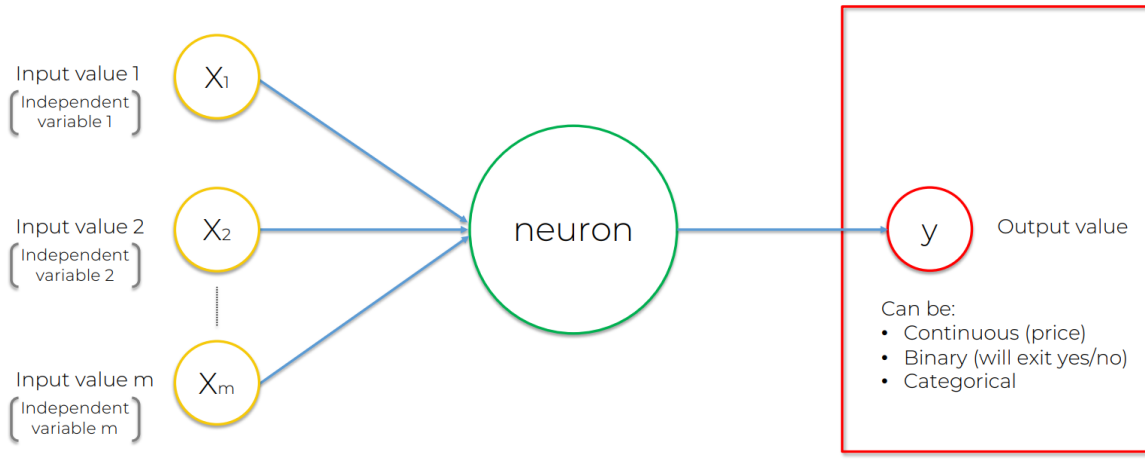




Input Layer Neuron:Neuron which yellow color;

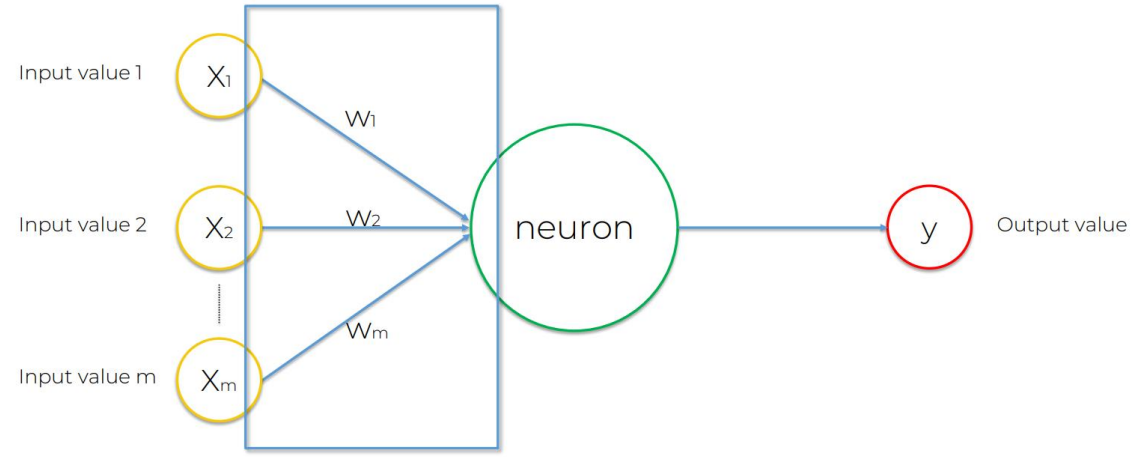
Hidden Layer Neuron: Neuron which green color;

Red Layer Neuron: Neuron which red color;

## The Neuron



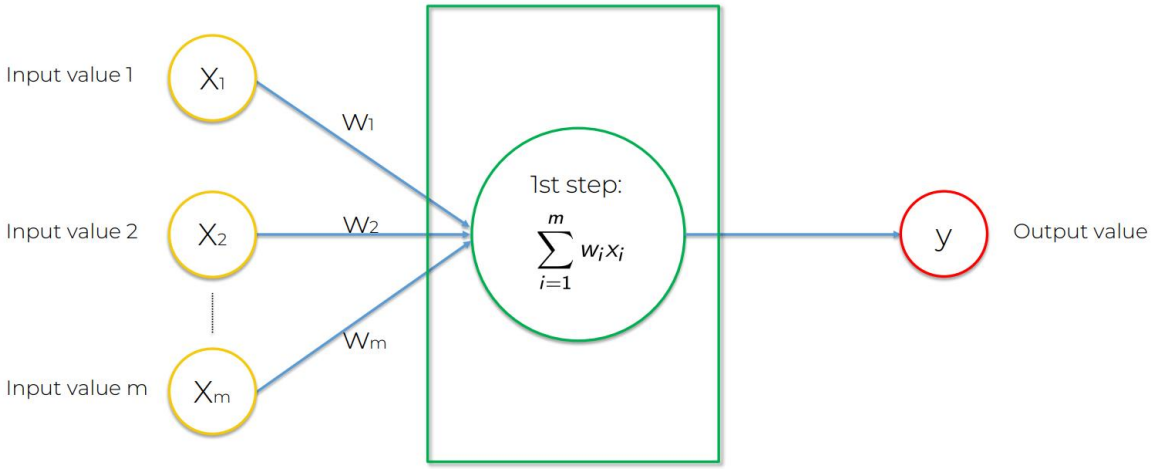

Output value may be continuous like price / Binary as Yes or No / Categorical that time may beseveral output values because your dummy variable which will be represent your category. Above just like simple linear regression or multivalue linear regression. It is a single observation.



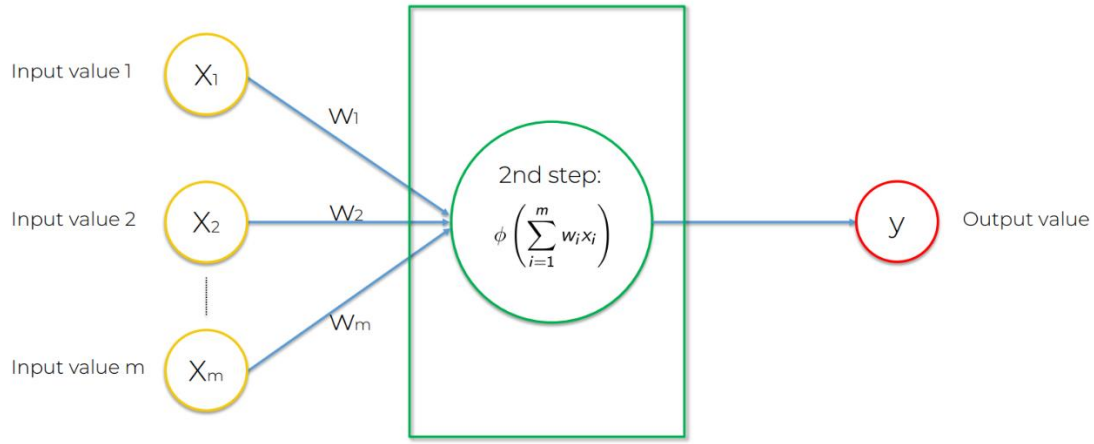

Here is synopsis. All are getting weight. Weights are very crucial to Artificial Neuron Network functioning because how Neural Network learn by adjusting the weights. Neural Network decide what signal is important or what not important neuron signal or what signal pass along or what does not pass along. Weights are crucial that their thing to processing learning. Training on ANN basic train of adjusting the weights in all the synopsis

across the in all neuron network. Where gradient descent or backpropagation come to play.

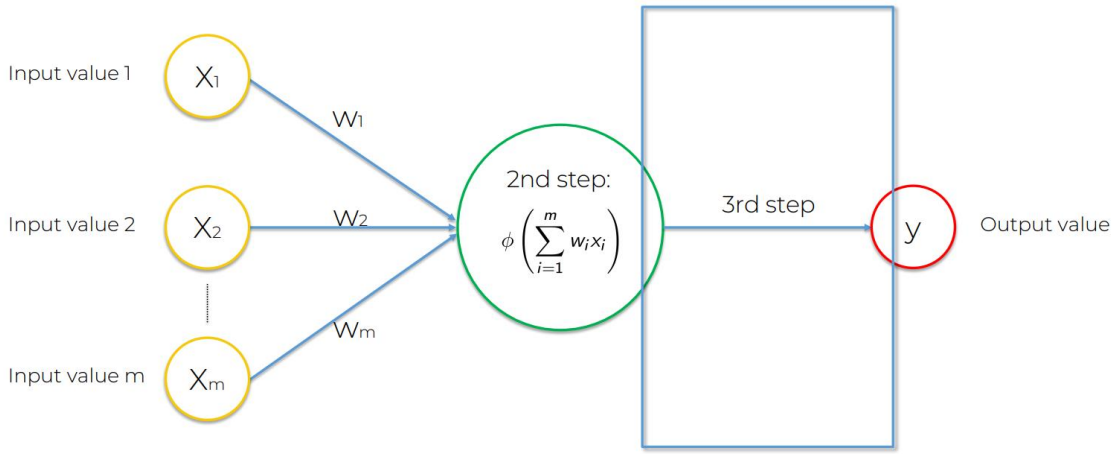
Signal goes into the neuron what happen inside the neuron. Few things happen- 1st thing: all to the value gets to added up. Its takes added weighted sum of all of the input values.




2nd Step: Then applied the activation function that neuron understand if the pass the signal or not.Basically decide the pass the signal to next neuron or decide the step-3.

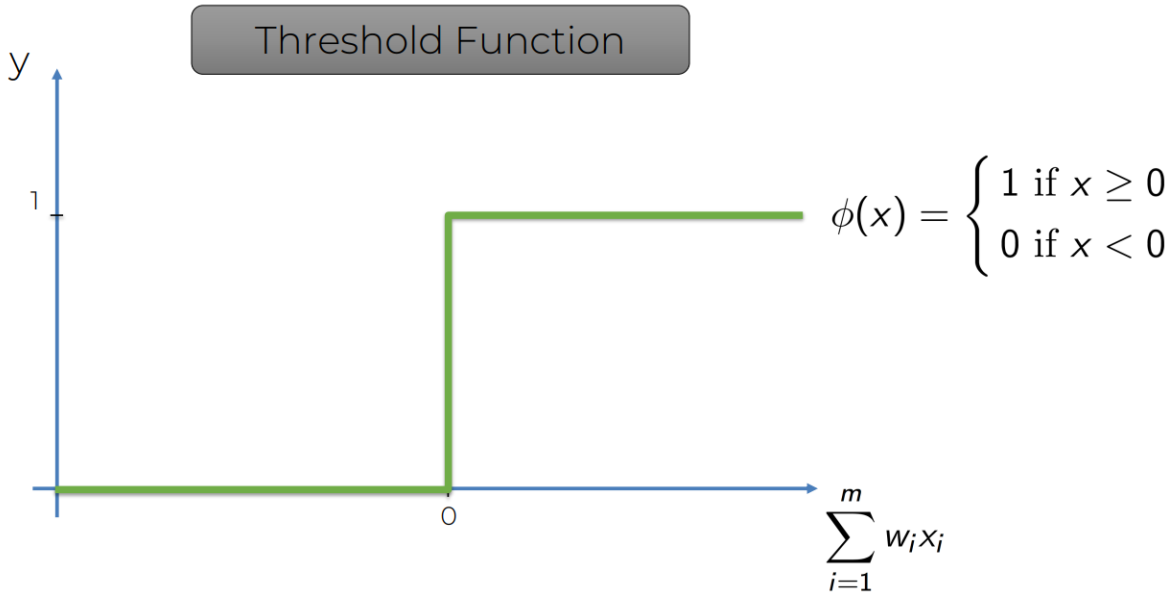



3rd Step: That is repeated throughout whole neuron network thousands of neurons.


Input value 1
X1
W1
Input value 2
X2
W2
Input value m
Xm
Wm
2nd step:
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
3rd step
y
Output value

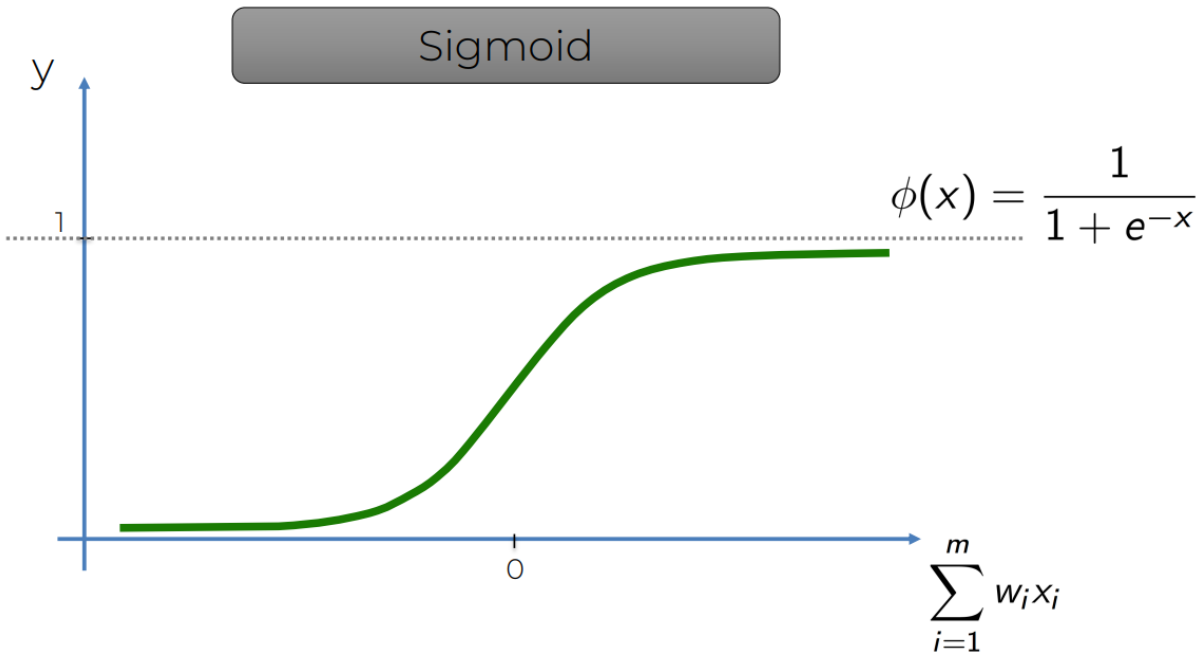
## Activation Functions

An activation function in a neural network is a mathematical function applied to the output of a neuron. Its primary purpose is to introduce non-linearity into the model, allowing the network to learn and represent complex patterns in the data. Without non-linearity, a neural network would essentially behave like a linear regression model, regardless of the number of layers it has.



Here are some common activation functions:

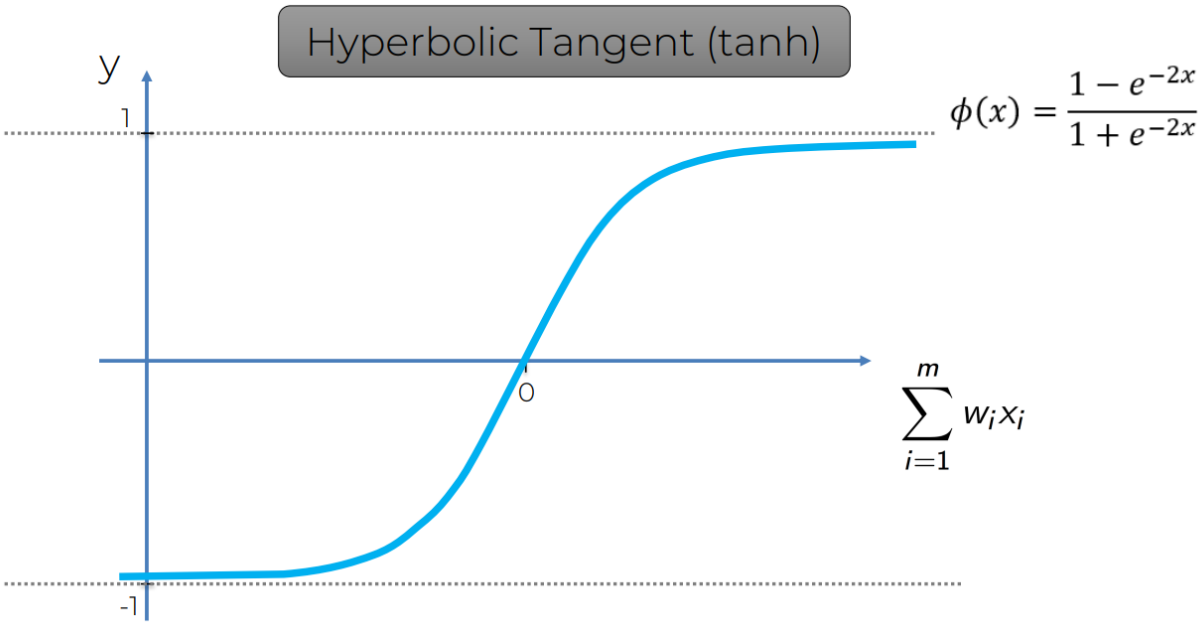
1. **Sigmoid Function**: This function maps any input to a value between 0 and 1. It's often used in the output layer of binary classification problems.




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

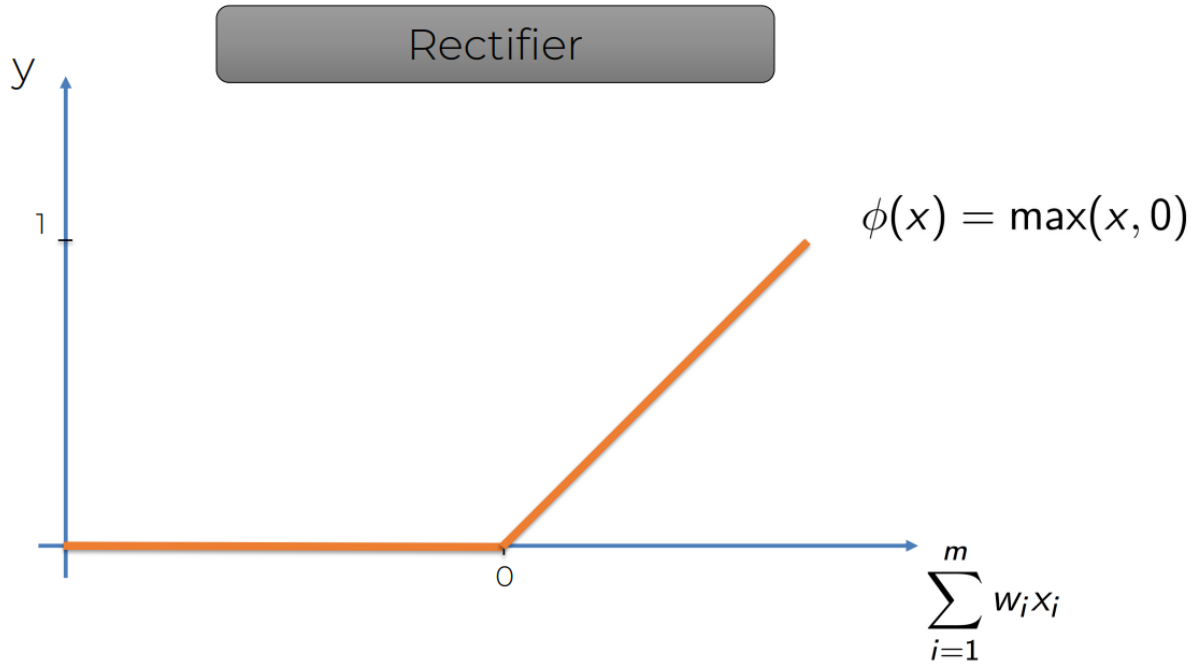
2. **Tanh Function**: Similar to the sigmoid function but maps inputs to values between -1 and 1. It is often used in hidden layers.The **hyperbolic tangent activation function (tanh)** is commonly used in artificial neural networks for hidden layers. It

transforms input values to produce output values between -1 and 1. The tanh function has an S-shape similar to the sigmoid activation function, but its output range is -1 to 1. It is useful for normalizing the output of a neuron and improving network performance.




$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

3. **ReLU (Rectified Linear Unit)**: This function outputs the input directly if it is positive; otherwise, it outputs zero. It's widely used in hidden layers of neural networks.




$$\text{ReLU}(x) = \max(0, x)$$

4. **Leaky ReLU**: A variant of ReLU that allows a small, non-zero gradient when the unit is not active.

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

where ( \alpha ) is a small constant.

5. **Softmax Function**: Used in the output layer of neural networks for multi-class classification problems. It converts logits into probabilities.

The **softmax function** is a mathematical operation that converts a vector of *K* real numbers into a probability distribution of *K* possible outcomes. Here's how it works:

A. Given an input vector **z** of *K* real numbers, the softmax function computes the exponential of each element in **z**.

B. It then normalizes these exponentials by dividing each value by the sum of all exponentials.

The result is a probability distribution consisting of *K* probabilities, where each probability corresponds to one of the possible outcomes.

Mathematically, the softmax function is defined as follows:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

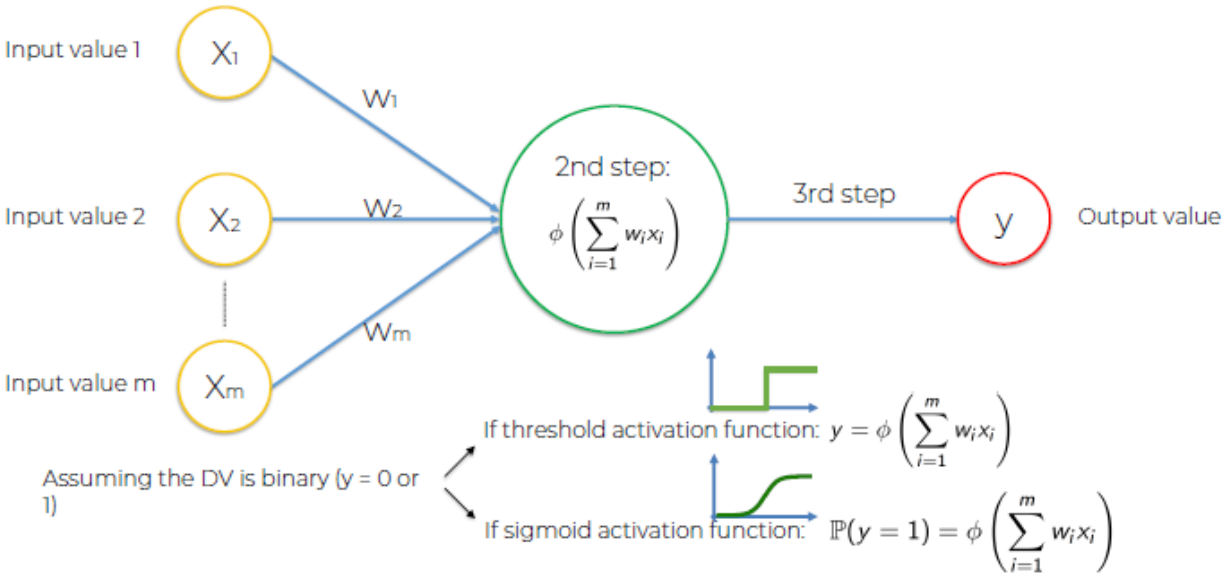
Here:

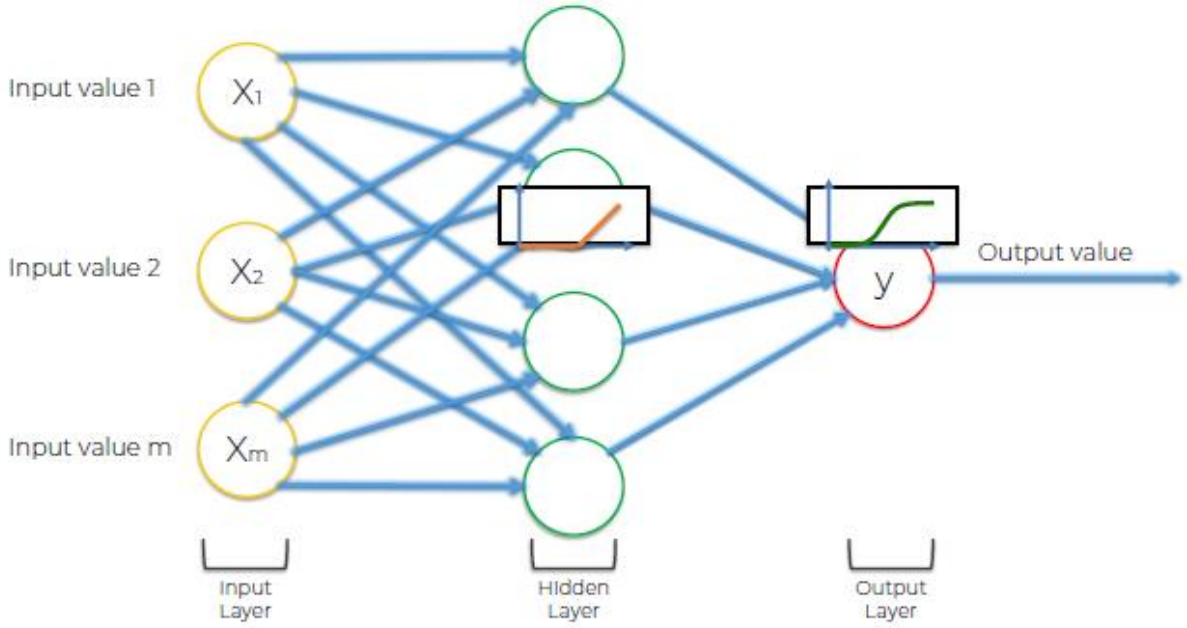
- (z_i) represents the *i*-th element of the input vector **z**.
- The denominator sums up the exponentials of all elements in **z**.

The softmax function is commonly used in machine learning, especially in neural networks, for tasks like classification. It ensures that the output values are non-negative and sum up to 1, making them interpretable as probabilities.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j} e^{x_j}}$$

Each activation function has its own advantages and is chosen based on the specific requirements of the neural.


Input value 1
$X_1$
$W_1$
Input value 2
$X_2$
$W_2$
Input value m
$X_m$
$W_m$
2nd step:
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
3rd step
y
Output value
Assuming the DV is binary (y = 0 or 1)
If threshold activation function: $y = \phi\left(\sum_{i=1}^{m} w_i x_i\right)$
If sigmoid activation function: $\mathbb{P}(y = 1) = \phi\left(\sum_{i=1}^{m} w_i x_i\right)$


Input value 1
$X_1$
Input value 2
$X_2$
Input value m
$X_m$
y
Output value
Input Layer
Hidden Layer
Output Layer

## How do neural networks work?

One way to understand how ANNs work is to examine how neural networks work in the human brain. The history of ANNs comes from biological inspiration and extensive study on how the brain works to process information.

### Artificial neural network structure

An individual neuron is a cell with an input and output structure. The input structure of a neuron is formed by dendrites, which receive signals from other nerve cells. The output structure is an axon that branches out from the cell body, connecting to the dendrites of another neuron via a synapse. Neurons communicate using electrochemical signals. Neurons only fire an output signal if the input signal meets a certain threshold in a specified amount of time.

ANNs operate similarly. They receive input signals that reach a threshold using sigmoid functions, process the information, and then generate an output signal. Like human neurons, ANNs receive multiple inputs, add them up, and then process the sum with a sigmoid function. If the sum fed into the sigmoid function produces a value that works, that value becomes the output of the ANN.

This is the structure of an individual neuron in an ANN, but networks have multiple layers and neurons that create the network. The structure of an entire artificial neural network consists of:

- **Input layer:** takes in the input data and transfers it to the second (hidden) layer of neurons using synapses. An input layer has as many nodes as features or columns of data in the matrix.
- **Hidden layer:** takes data from the input layer to categorize or detect desired aspects of the data. Nodes in the hidden layer send the data to more hidden layers or, finally, to the output layer. The hidden layer of an ANN is a “black box” because researchers cannot determine its results.
- **Output layer:** takes data from the hidden layer and outputs the results. It has as many nodes as the model desires.
- **Synapses:** connect nodes in layers and in between layers.

Deep neural networks, which are used in deep learning, have a similar structure to a basic neural network, except they use multiple hidden layers and require significantly more time and data to train.

### Types of neural networks

Neural networks vary in type based on how they process information and how many hidden layers they contain. Three types of neural networks include the following:

- Feed-forward neural networks

- Backpropagation neural networks
- Convolution neural networks

Let’s take a closer look at how each neural network type works.

## Convolution Neural Network

A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

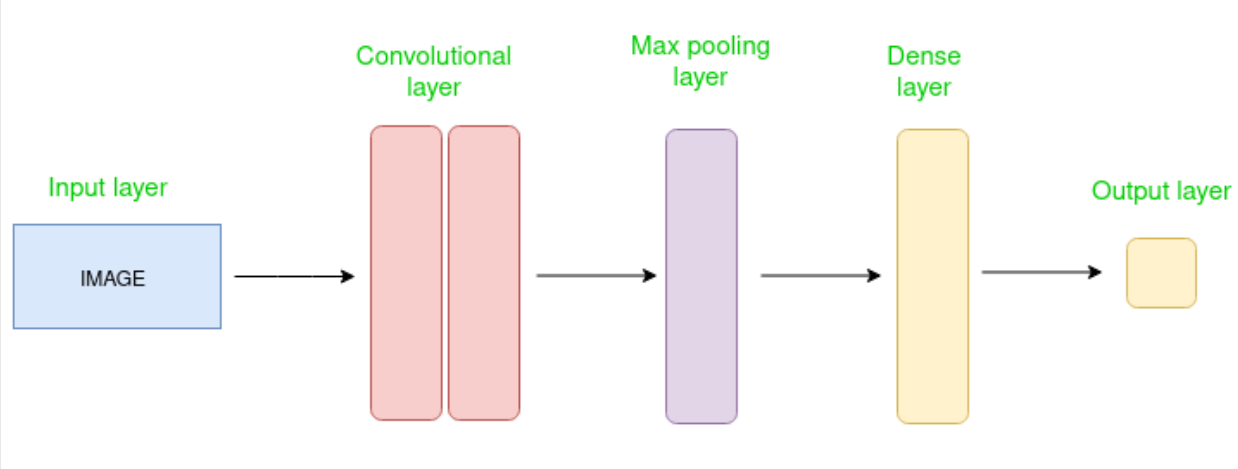
When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use **Recurrent Neural Networks** more precisely an LSTM, similarly for image classification we use Convolution Neural networks.

### Convolution Neural Network

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example visual datasets like images or videos where data patterns play an extensive role.

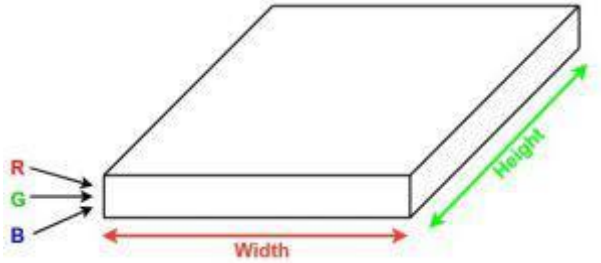
### CNN architecture

Convolutional Neural Network consists of multiple layers like the input layer, Convolutional layer, Pooling layer, and fully connected layers.



**How Convolutional Layers works**

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e the channel as images generally have red, green, and blue channels).




Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called **Convolution**. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

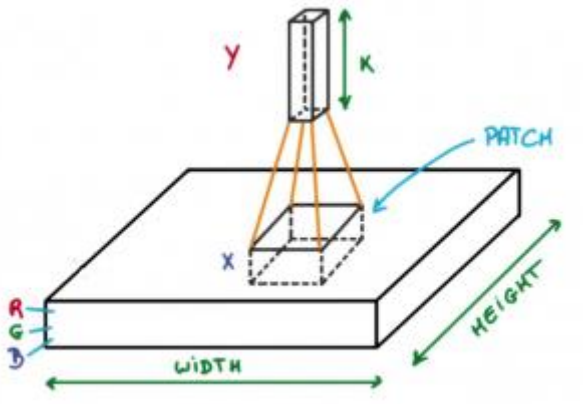



*Image source: Deep Learning Udacity*

Now let's talk about a bit of mathematics that is involved in the whole convolution process.

- Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can be axax3, where

‘a’ can be anything like 3, 5, or 7 but smaller as compared to the image dimension.

- During the forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have a value of 2, 3, or even 4 for high-dimensional images) and compute the dot product between the kernel weights and patch from input volume.
- As we slide our filters we’ll get a 2-D output for each filter and we’ll stack them together as a result, we’ll get output volume having a depth equal to the number of filters. The network will learn all the filters.

**Layers used to build ConvNets**

A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

**Types of layers:** datasets

Let’s take an example by running a covnets on of image of dimension 32 x 32 x 3.

- **Input Layers:** It’s the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.
- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. it slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we’ll get an output volume of dimension 32 x 32 x 12.
- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are **RELU**: max(0, x), **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.
- **Pooling layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.

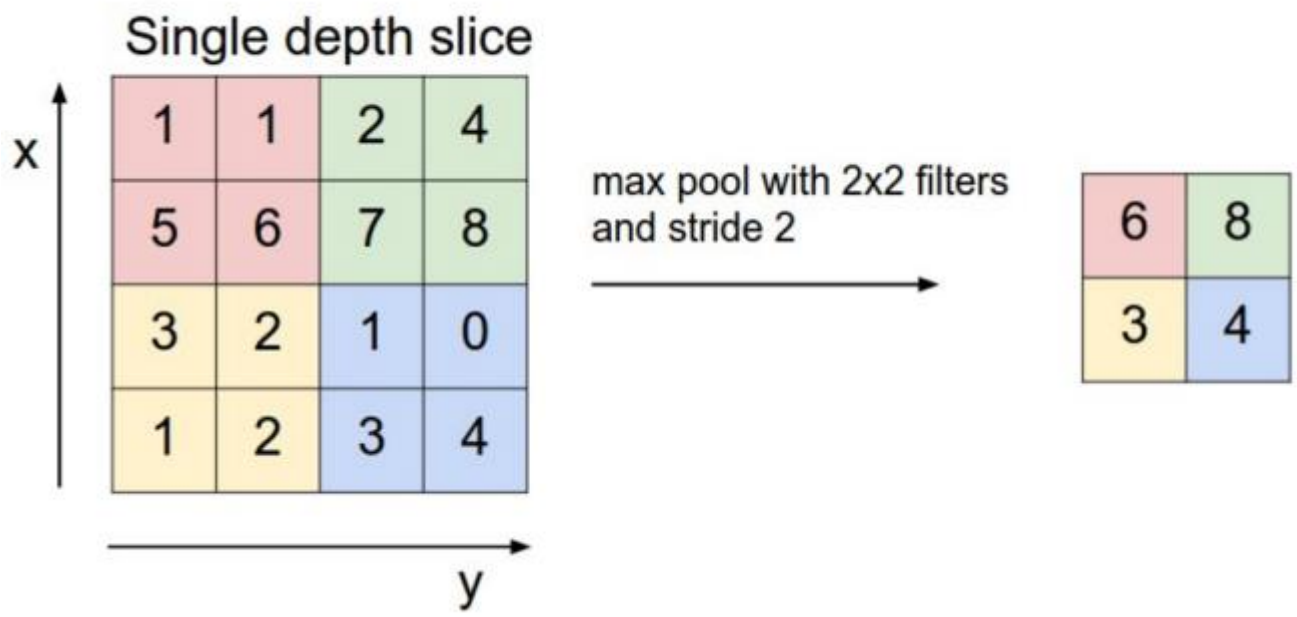



*Image source: cs231n.stanford.edu*

- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.

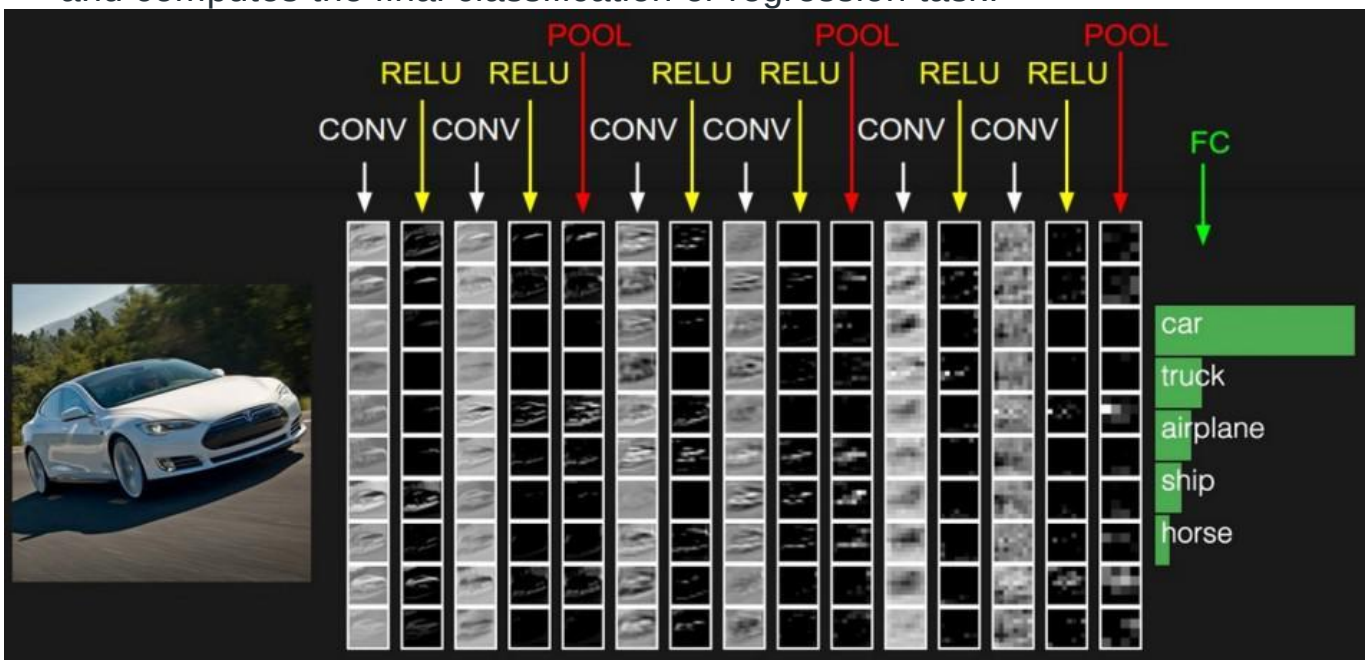



*Image source: cs231n.stanford.edu*

- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

# Gradient Descent

## What Is Gradient Descent in Machine Learning?

Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent in machine learning is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

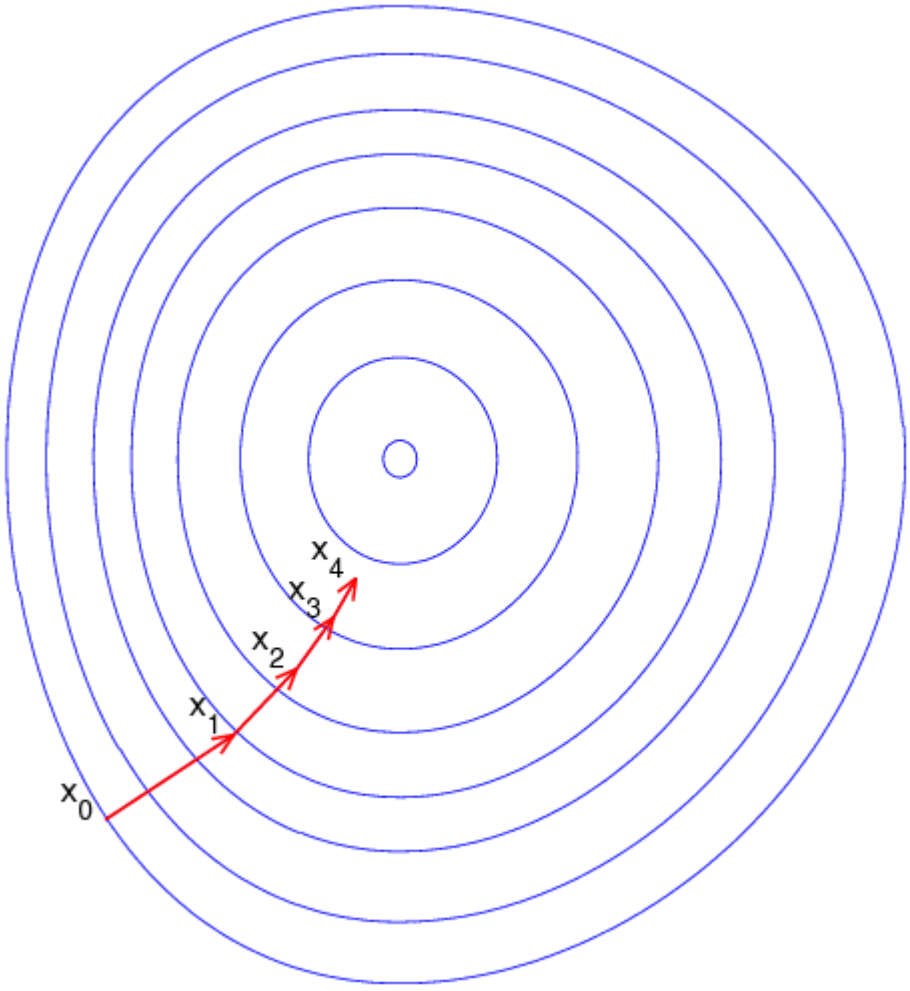
Gradient descent is an optimization algorithm that's used when training a machine learning model. It's based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.

You start by defining the initial parameter's values and from there the gradient descent algorithm uses calculus to iteratively adjust the values so they minimize the given cost-function.

### What Is a Gradient?

A gradient simply measures the change in all weights with regard to the change in error. 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. But if the slope is zero, the model stops learning. In mathematical terms, a gradient is a partial derivative with respect to its inputs.

Imagine a blindfolded man who wants to climb to the top of a hill with the fewest steps possible. He might start climbing the hill by taking really big steps in the steepest direction. But as he comes closer to the top, his steps will get smaller and smaller to avoid overshooting it. Imagine the image below illustrates our hill from a top-down view and the red arrows are the steps of our climber. A gradient in this context is a vector that contains the direction of the steepest step the blindfolded man can take and how long that step should be.




Note that the gradient ranging from X0 to X1 is much longer than the one reaching from X3 to X4. This is because the steepness/slope of the hill, which determines the length of the vector, is less. This perfectly represents the example of the hill because the hill is getting less steep the higher it's climbed, so a reduced gradient goes along with a reduced slope and a reduced step size for the hill climber.

### How Does Gradient Descent Work?

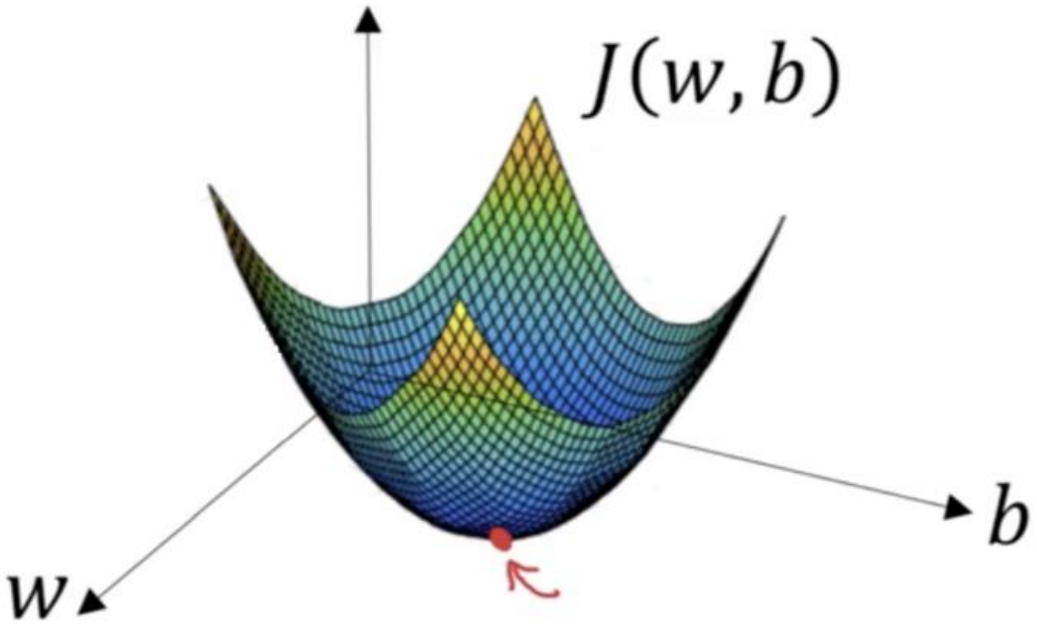
Instead of climbing up a hill, think of gradient descent as hiking down to the bottom of a valley. The equation below describes what the gradient descent algorithm does: *b* is the next position of our climber, while *a* represents his current position. The minus sign refers to the minimization part of the gradient descent algorithm. The gamma in the middle is a waiting factor

and the gradient term ( Δf(a) ) is simply the direction of the steepest descent.

$$\mathbf{b} = \mathbf{a} - \gamma \nabla \mathbf{f(a)}$$

This formula basically tells us the next position we need to go, which is the direction of the steepest descent. Let's look at another example to really drive the concept home.

Imagine you have a machine learning problem and want to train your algorithm with gradient descent to minimize your cost-function $J(w, b)$ and reach its local minimum by tweaking its parameters ($w$ and $b$). The image below shows the horizontal axes representing the parameters ($w$ and $b$), while the cost function $J(w, b)$ is represented on the vertical axes.



We want to find the values of *w* and *b* that correspond to the minimum of the cost function (marked with the red arrow). To start, we initialize *w* and *b* with some random numbers. Gradient descent then starts at that point (somewhere around the top of our illustration), and it takes one step after another in the steepest downside direction (i.e., from the top to the bottom of the illustration) until it reaches the point where the cost function is as small as possible.

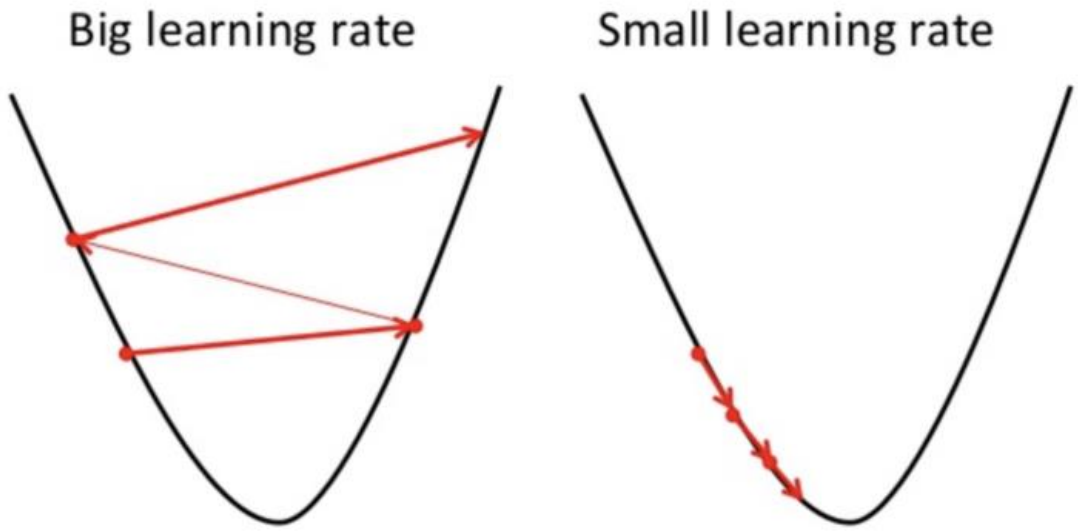
**Types of Gradient Descent:**

- **Batch Gradient Descent**
- **Stochastic Gradient Descent**
- **Mini-Batch Gradient Descent**

## Gradient Descent Learning Rate

How big the steps gradient descent takes in the direction of the local minimum is determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.

For the gradient descent algorithm to reach the local minimum, we must set the learning rate to an appropriate value, which is neither too low nor too high. This is important because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent (see left image below). If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while (see the right image).

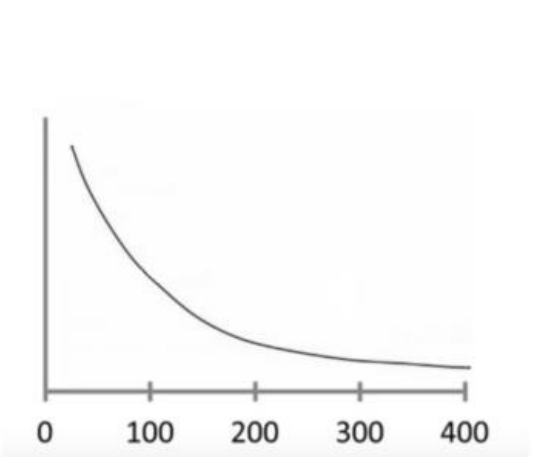


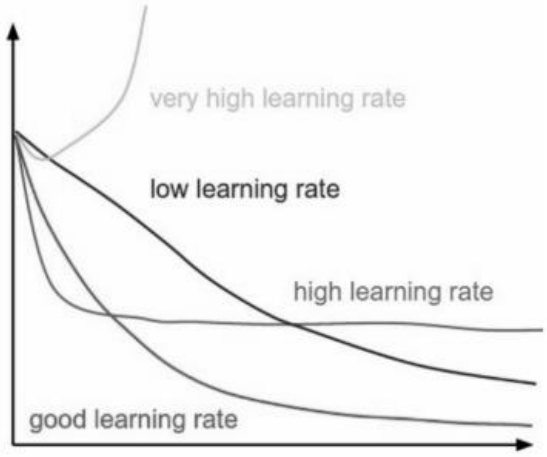

The learning rate should never be too high or too low for this reason. You can check if your learning rate is doing well by plotting it on a graph.

## How to Solve Gradient Descent Challenges

To make sure the gradient descent algorithm runs properly, plot the cost function as the optimization runs. Put the number of iterations on the x-axis and the value of the cost function on the y-axis. This helps you see the value of your cost function after each iteration of gradient descent, and provides a way to easily spot how appropriate your learning rate is. Try different values for it and plot them all together. The left image below

shows such a plot, while the image on the right illustrates the difference between good and bad learning rates.




If the gradient descent algorithm is working properly, the cost function should decrease after every iteration.

When gradient descent can't decrease the cost function anymore and remains more or less on the same level, it has converged. The number of iterations gradient descent needs to converge can sometimes vary a lot. It can take 50 iterations, 60,000 or maybe even 3 million, making the number of iterations to convergence hard to estimate in advance.

There are some algorithms that can automatically tell you if gradient descent has converged, but you must define a threshold for the convergence beforehand, which is also pretty hard to estimate. For this reason, simple plots are the preferred convergence test.

Another advantage of monitoring gradient descent via plots is it allows us to easily spot if it doesn't work properly, for example if the cost function is increasing. Most of the time the reason for an increasing cost-function when using gradient descent is a learning rate that's too high.

If the plot shows the learning curve going up and down without really reaching a lower point, try decreasing the learning rate. Also, when starting out with gradient descent on a given problem, simply try 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, etc., as the learning rates and look at which one performs the best.