UNIT-IV

Topics:
Neural Networks: Multilayer Perceptron, Back-propagation algorithm, Training strategies, Activation Functions, Gradient Descent For Machine Learning, Radial basis functions, Hopfield network, Recurrent Neural Networks.
Deep learning: Introduction to deep learning, Convolutional Neural Networks (CNN), CNN Architecture, pre-trained CNN (LeNet, AlexNet).

What is a neural network?

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. 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. It creates an adaptive system that computers use to learn from their mistakes and improve continuously. Thus, artificial neural networks attempt to solve complicated problems, like summarizing documents or recognizing faces, with greater accuracy.

Why are neural networks important?

Neural networks can help computers make intelligent decisions with limited human assistance. This is because they can learn and model the relationships between input and output data that are nonlinear and complex. For instance, they can do the following tasks.

Make generalizations and inferences

Neural networks can comprehend unstructured data and make general observations without explicit training. For instance, they can recognize that two different input sentences have a similar meaning:

- Can you tell me how to make the payment?
- How do I transfer money?

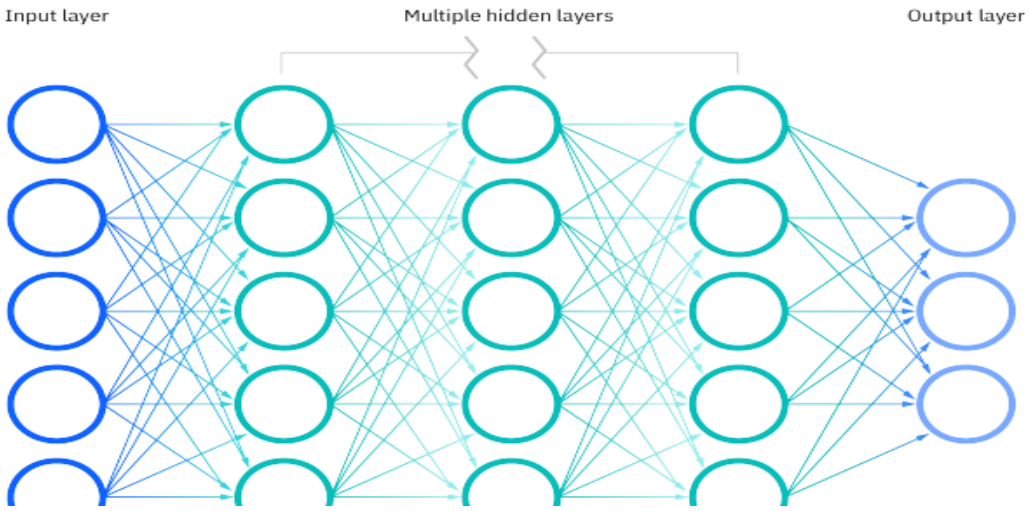
A neural network would know that both sentences mean the same thing. Or it would be able to broadly recognize that Baxter Road is a place, but Baxter Smith is a person's name.

What are neural networks used for?

Neural networks have several use cases across many industries, such as the following:

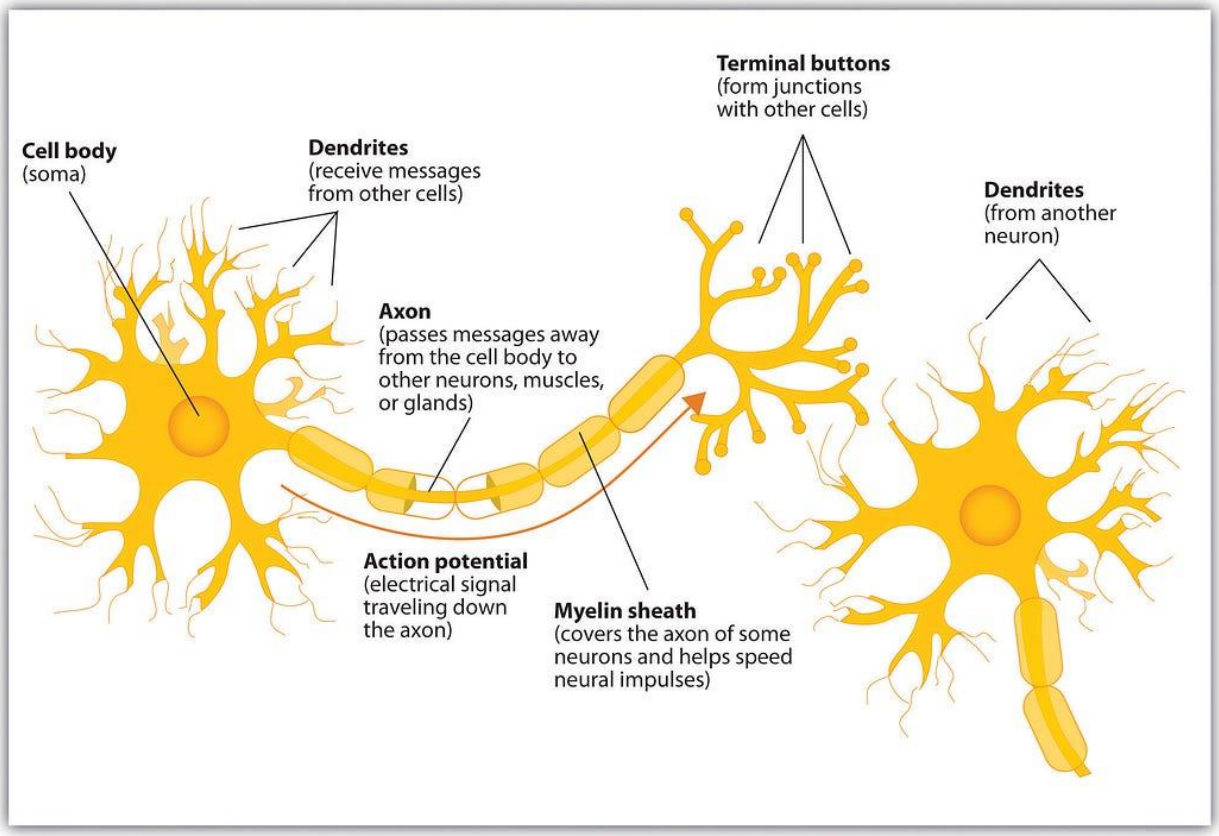
- Medical diagnosis by medical image classification
- Targeted marketing by social network filtering and behavioral data analysis
- Financial predictions by processing historical data of financial instruments
- Electrical load and energy demand forecasting
- Process and quality control

- Chemical compound identification



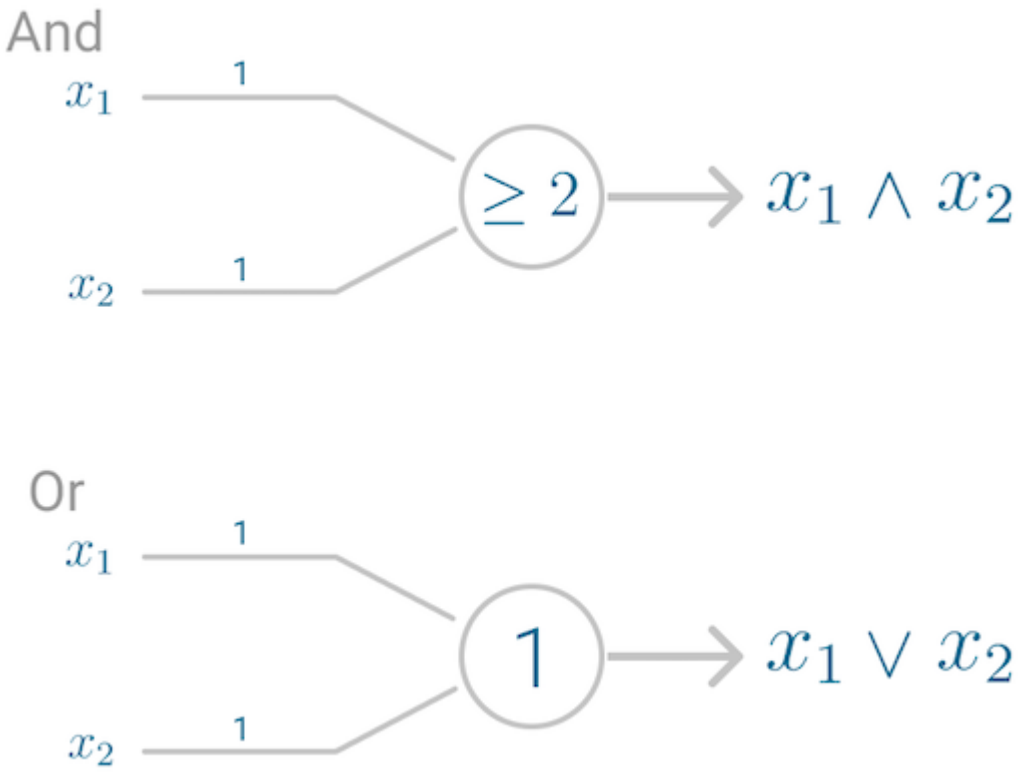
Neural Networks are inspired by, but not necessarily an exact model of, the structure of the brain. There's a lot we still don't know about the brain and how it works, but it has been serving as inspiration in many scientific areas due to its ability to develop intelligence. And although there are neural networks that were created with the sole purpose of understanding how brains work, Deep Learning as we know it today is not intended to replicate how the brain works. Instead, Deep Learning focuses on enabling systems that learn multiple levels of pattern composition.

$$\underbrace{f(x, w)}_{\text{output}} = \underbrace{x_1}_{\text{inputs}} w_1 + \cdots + x_n \overbrace{w_n}^{\text{weights}}$$



Neuron and it's different components

The first application of the neuron replicated a logic gate, where you have one or two binary inputs, and a boolean function that only gets activated given the right inputs and weights.



However, this model had a problem. It couldn't *learn* like the brain. The only way to get the desired output was if the weights, working as catalyst in the model, were set beforehand.

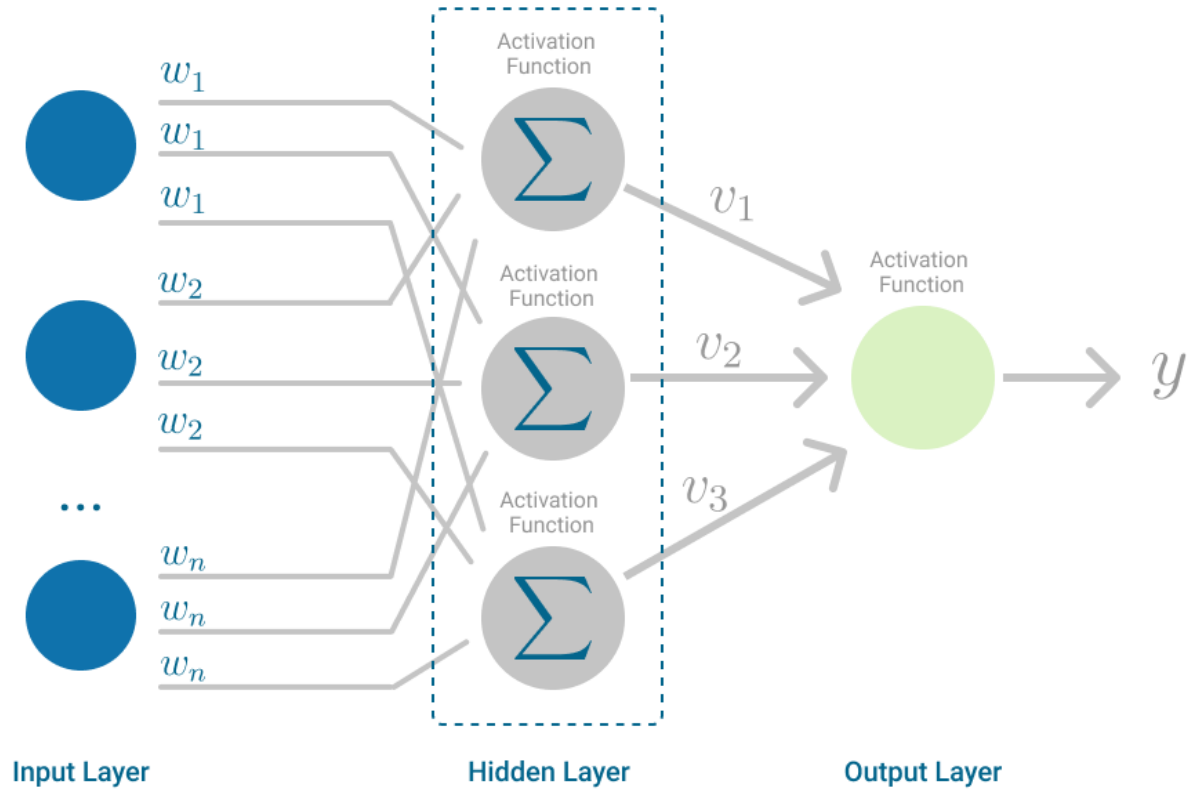
Perceptron

Although today the **Perceptron** is widely recognized as an algorithm, it was initially intended as an image recognition machine. It gets its name from performing the human-like function of *perception*, seeing and recognizing images.

Multilayer Perceptron

The **Multilayer Perceptron** was developed to tackle this limitation. It is a neural network where the mapping between inputs and output is non-linear.

A Multilayer Perceptron has input and output layers, and one or more **hidden layers** with many neurons stacked together. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.



Multilayer Perceptron falls under the category of [feedforward algorithms](), because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer.

Each layer is *feeding* the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

But it has more to it.

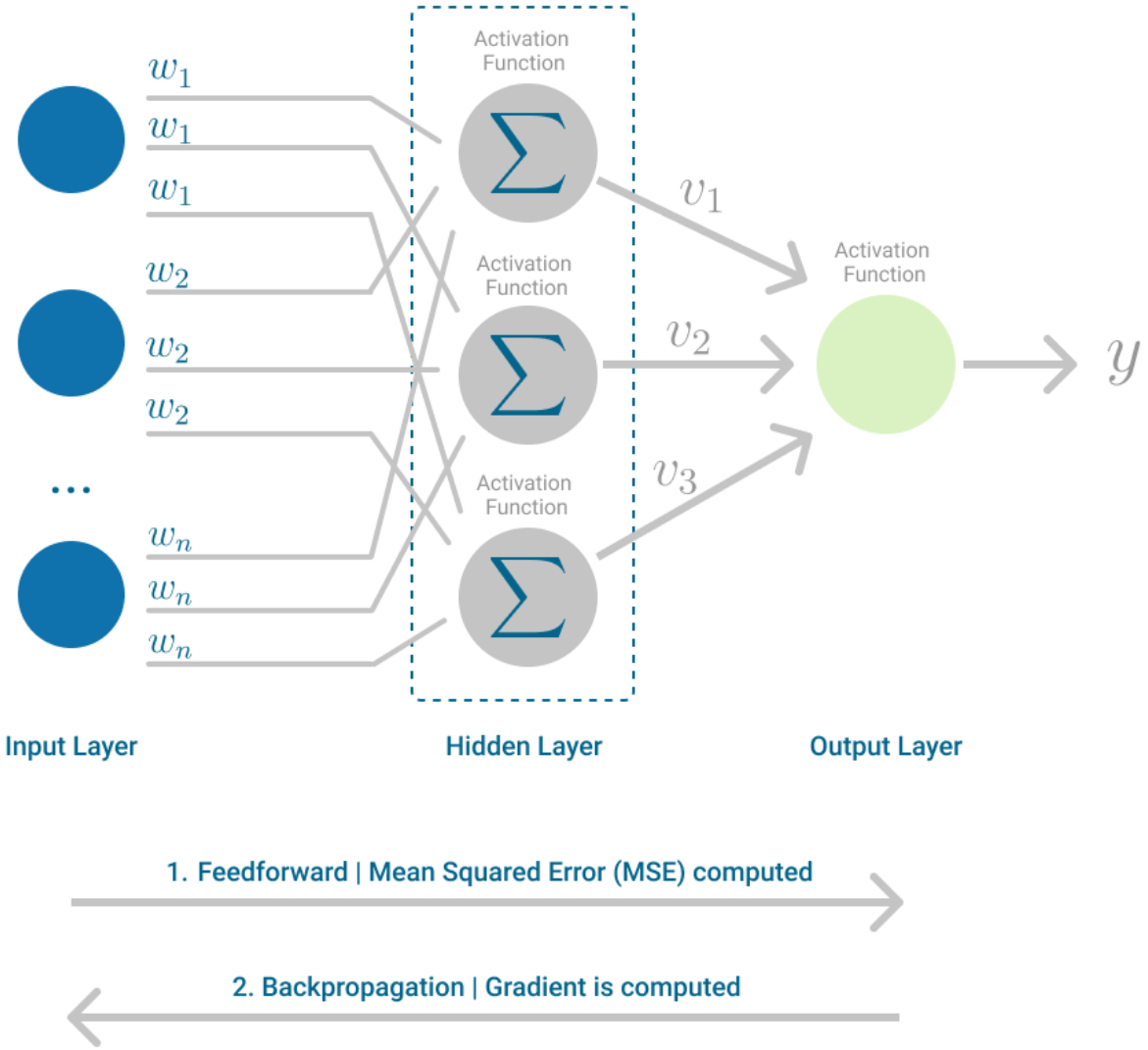
If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to *learn* the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

This is where **Backpropagation**[7] comes into play.

Backpropagation

Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.

There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable. These functions must have a **bounded derivative**, because Gradient Descent is typically the optimization function used in MultiLayer Perceptron.



Multilayer Perceptron, highlighting the Feedfoward and Backpropagation steps. (Image by author)

In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network!

$$\underbrace{\Delta_w(t)}_{\substack{\text{Gradient}\\\text{Current Iteration}}} = \overbrace{-}^{\text{Bias}}\varepsilon\frac{\overbrace{dE}^{\text{Error}}}{\underbrace{dw_{(t)}}_{\text{Weight vector}}} + \overbrace{\alpha}^{\text{Learning Rate}}\underbrace{\Delta_{w(t-1)}}_{\substack{\text{Gradient}\\\text{Previous Iteration}}}$$

One iteration of Gradient Descent.

This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified *convergence threshold*, compared to the previous iteration.

Derivation of Backpropagation

1 Introduction

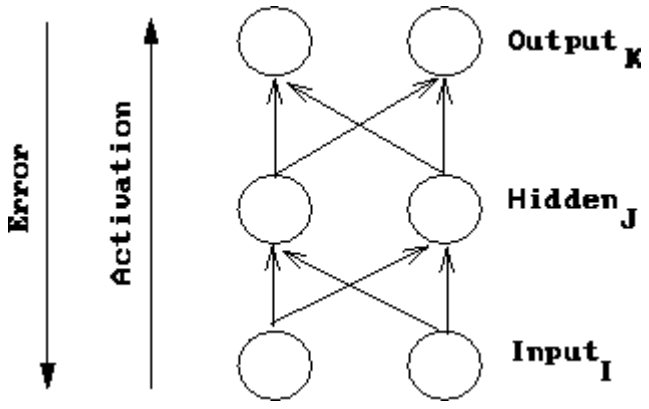


Figure 1: Neural network processing

Conceptually, a network forward propagates activation to produce an output and it backward propagates error to determine weight changes (as shown in Figure 1). The weights on the connections between neurons mediate the passed values in both directions.

The Backpropagation algorithm is used to learn the weights of a multilayer neural network with a fixed architecture. It performs gradient descent to try to minimize the sum squared error between the network's output values and the given target values.

Figure 2 depicts the network components which affect a particular weight change. Notice that all the necessary components are locally related to the weight being updated. This is one featureof backpropagation that seems biologically plausible. However, brain connections appear to be unidirectional and not bidirectional as would be required to implement backpropagation.

2 Notation

For the purpose of this derivation, we will use the following notation:

- The subscript *k* denotes the output layer.
- The subscript *j* denotes the hidden layer.
- The subscript *i* denotes the input layer.

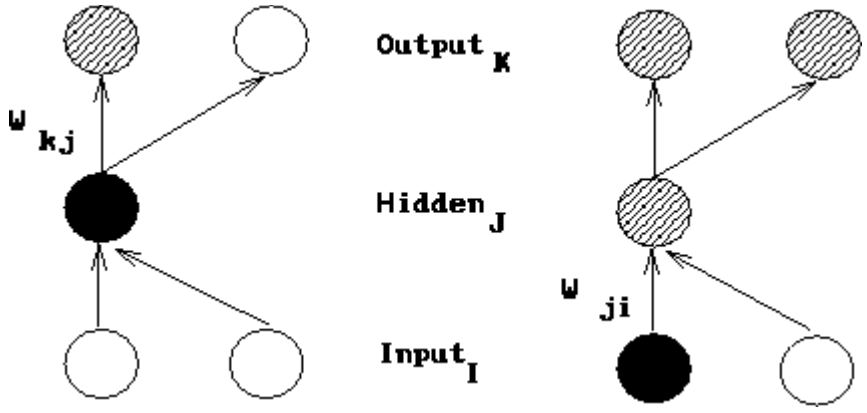


Figure 2: The change to a hidden to output weight depends on error (depicted as a lined pattern) at the output node and activation (depicted as a solid pattern) at the hidden node. While the change to a input to hidden weight depends on error at the hidden node (which in turn dependson error at all the output nodes) and activation at the input node.

- w_{kj} denotes a weight from the hidden to the output layer.
- w_{ji} denotes a weight from the input to the hidden layer.
- a denotes an activation value.
- t denotes a target value.
- *net* denotes the net input.

3 Review of Calculus Rules

$$\frac{d(e^u)}{dx} = e^u \frac{du}{dx} \qquad \frac{d(g+h)}{dx} = \frac{dg}{dx} + \frac{dh}{dx} \qquad \frac{d(g^n)}{dx} = ng^{n-1}\frac{dg}{dx}$$

4 Gradient Descent on Error

We can motivate the backpropagation learning algorithm as gradient descent on sum-squared error (we square the error because we are interested in its magnitude, not its sign). The total error in a network is given by the following equation (the $\frac{1}{2}$ will simplify things later).

$$E = \frac{1}{2}\sum_{k}(t_k - a_k)^2$$

We want to adjust the network's weights to reduce this overall error.

$$\Delta W \propto -\frac{\partial E}{\partial W}$$

We will begin at the output layer with a particular weight.

$$\Delta w_{kj} \propto -\frac{\partial E}{\partial w_{kj}}$$

However error is not directly a function of a weight. We expand this as follows.

$$\Delta w_{kj} = -\varepsilon \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}}$$

Let's consider each of these partial derivatives in turn. Note that only one term of the E summation will have a non-zero derivative: the one associated with the particular weight we are considering.

4.1 Derivative of the error with respect to the activation

$$\frac{\partial E}{\partial a_k} = \frac{\partial(\frac{1}{2}(t_k - a_k)^2)}{\partial a_k} = -(t_k - a_k)$$

Now we see why the $\frac{1}{2}$ in the E term was useful.

4.2 Derivative of the activation with respect to the net input

$$\frac{\partial a_k}{} = \frac{\partial(1 + e^{-net_k})^{-1}}{} = \frac{e^{-net_k}}{-net\ \ 2}$$

$$\frac{\partial net_k}{} \qquad \frac{\partial net_k}{} \qquad (1 + e^{\quad k})$$

We'd like to be able to rewrite this result in terms of the activation function. Notice that:

$$1 - \frac{1}{1 + e^{-net_k}} = \frac{e^{-net_k}}{1 + e^{-net_k}}$$

Using this fact, we can rewrite the result of the partial derivative as:

$$a_k(1 - a_k)$$

4.3 Derivative of the net input with respect to a weight

Note that only one term of the *net* summation will have a non-zero derivative: again the one associated with the particular weight we are considering.

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial (w_{kj} a_j)}{\partial w_{kj}} = a_j$$

4.4 Weight change rule for a hidden to output weight

Now substituting these results back into our original equation we have:

$$\Delta w_{kj} = \varepsilon \underbrace{(t_k - a_k) a_k (1 - a_k)}_{\delta_k} a_j$$

$$\Delta w_{kj} = \varepsilon (t_k - a_k) a_k (1 - a_k)\, a_j$$

Notice that this looks very similar to the Perceptron Training Rule. The only difference is the inclusion of the derivative of the activation function. This equation is typically simplified as shown below where the δ term repesents the product of the error with the derivative of the activation function.

$$\Delta w_{kj} = \varepsilon \delta_k a_j$$

4.5 Weight change rule for an input to hidden weight

Now we have to determine the appropriate weight change for an input to hidden weight. This is more complicated because it depends on the error at all of the nodes this weighted connection canlead to.

$$\Delta w_{ji} \propto -\left[\sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial net_k}\frac{\partial net_k}{\partial a_j}\right]\frac{\partial a_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= \varepsilon\left[\sum_k \underbrace{(t_k - a_k)a_k(1-a_k)}_{\delta_k} w_{kj}\right]a_j(1-a_j)a_i$$

$$= \varepsilon\left[\sum_k \underbrace{\delta_k w_{kj}}_{\delta_j}\right]a_j(1-a_j)\,a_i$$

$$\Delta w_{ji} = \varepsilon \delta_j a_i$$

Training strategy

The procedure used to carry out the learning process is called the training (or learning) strategy. The training strategy is applied to the neural network to obtain the minimum loss possible. This is done by searching for parameters that fit the neural network to the data set.

A general strategy consists of two different concepts:

- **Loss index**
- **Optimization algorithm**

4.1. Loss index

The loss index plays a vital role in the use of neural networks. It defines the task the neural network is required to do and provides a measure of the quality of the representation required to learn. The choice of a suitable loss index depends on the application.

When setting a loss index, two different terms must be chosen: an error term and a regularization term. loss_index=error_term+regularization_term

Error term

The error is the most important term in the loss expression. It measures how the neural network fits the data set.

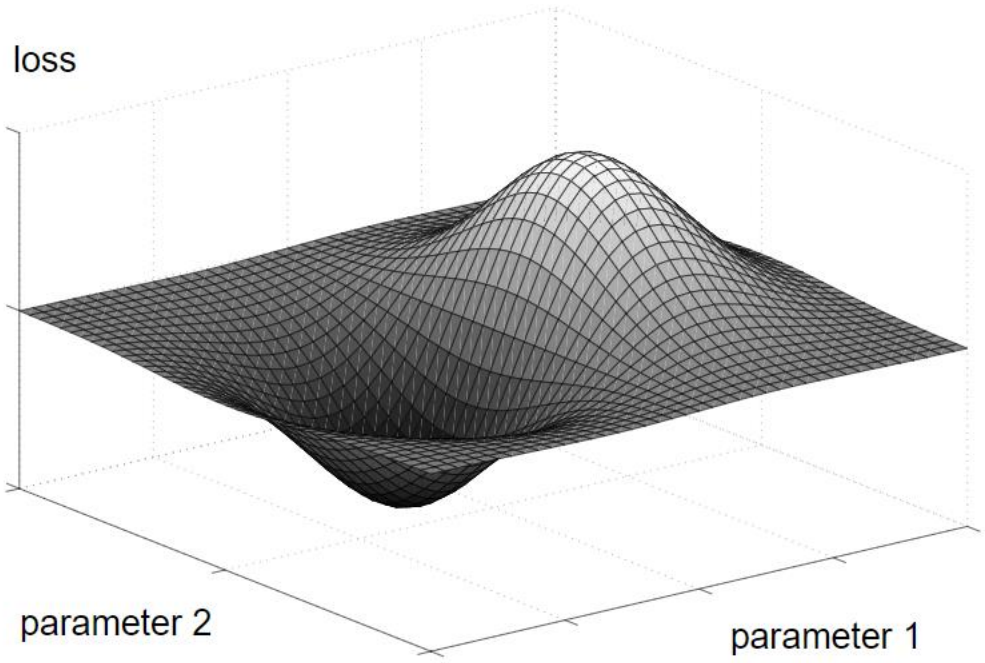
All those errors can be measured over different subsets of the data. In this regard, the training error refers to the error measured on the training samples, the selection error is measured on the selection samples, and the testing error is measured on the testing samples.

Mean squared error (MSE)

The mean squared error calculates the average squared error between the outputs from the neural network and the targets in the data set.
mean_squared_error=∑(outputs−targets)2/samples_number

Loss function

The loss index depends on the function represented by the neural network, and it is measured on the data set. It can be visualized as a hyper-surface with the parameters as coordinates. See the following figure.



The learning problem for neural networks can then be stated as finding a neural network function for which the loss index takes on a minimum value. That is, to find the set of parameters that minimize the above function.

Gradient descent (GD)

The most straightforward optimization algorithm is gradient descent. Here, the parameters are updated at each epoch in the direction of the negative gradient of the loss index.new_parameters=parameters−loss_gradient∗learning_rate.

The learning rate is usually adjusted at each epoch using line minimization.

Activation Function

Definition

In artificial neural networks, an activation function is one that outputs a smaller value for tiny inputs and a higher value if its inputs are greater than a threshold. An activation function "fires" if the inputs are big enough; otherwise, nothing happens. An activation function, then, is a gate that verifies how an incoming value is higher than a threshold value.

Because they introduce non-linearities in neural networks and enable the neural networks can learn powerful operations, activation functions are helpful. A feedforward neural network might be refactored into a straightforward linear function or matrix transformation on to its input if indeed the activation functions were taken out.

By generating a weighted total and then including bias with it, the activation function determines whether a neuron should be turned on. The activation function seeks to boost a neuron's output's nonlinearity.

Gradient Descent in Machine Learning

Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

In mathematical terminology, Optimization algorithm refers to the task of minimizing/maximizing an objective function f(x) parameterized by x. Similarly, in machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters. The main objective of gradient descent is to minimize the convex function using iteration of parameter updates. Once these machine learning models are optimized, these models can be used as powerful tools for Artificial Intelligence and various computer science applications.

In this tutorial on Gradient Descent in Machine Learning, we will learn in detail about gradient descent, the role of cost functions specifically as a barometer within Machine Learning, types of gradient descents, learning rates, etc.

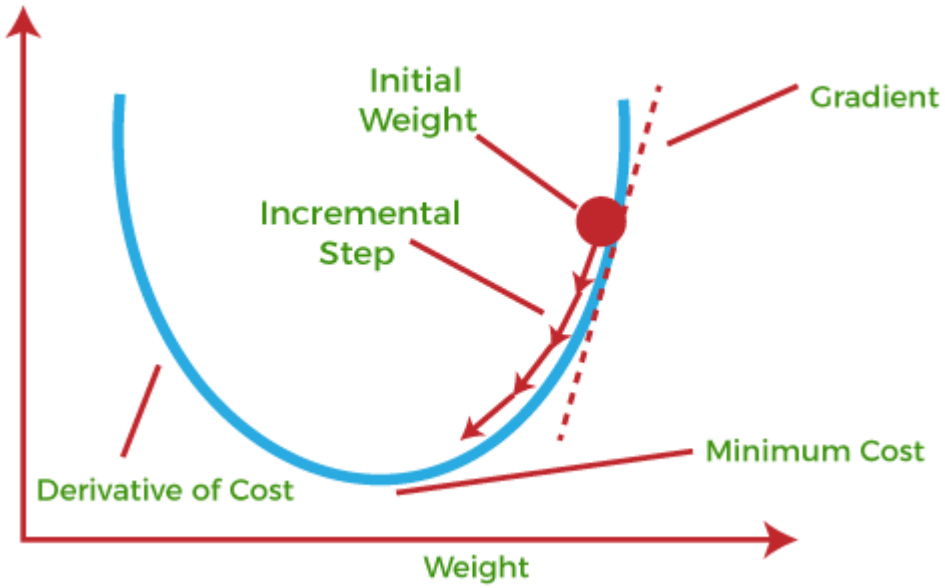
What is Gradient Descent or Steepest Descent?

Gradient descent was initially discovered by **"Augustin-Louis Cauchy"** in mid of 18th century. ***Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.***

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.

- Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.



This entire procedure is known as Gradient Ascent, which is also known as steepest descent. ***The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.*** To achieve this goal, it performs two steps iteratively:

- Calculates the first-order derivative of the function to compute the gradient or slope of that function.
- Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.

What is Cost-function?

The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number. It helps to increase and improve machine learning efficiency by providing feedback to this model so that it can minimize error and find the local or global minimum. Further, it continuously iterates along the direction of the negative gradient until the cost function approaches zero. At this steepest descent point, the model will stop learning further. Although cost function and loss function are considered synonymous, also there is a minor difference between them. The slight difference between the loss function and the cost function is about the error within the training of machine learning

models, as loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

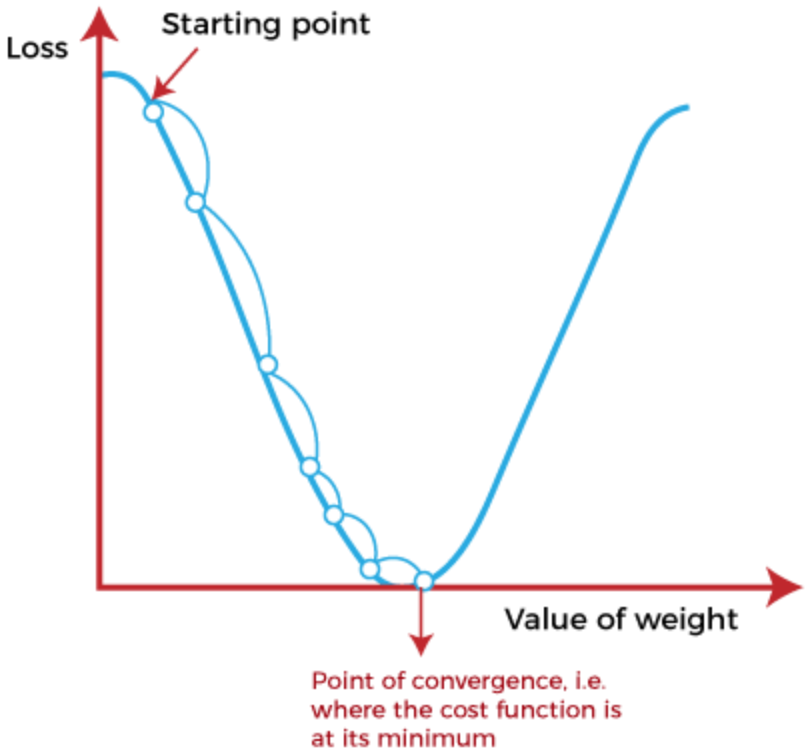
The cost function is calculated after making a hypothesis with initial parameters and modifying these parameters using gradient descent algorithms over known data to reduce the cost function.

How does Gradient Descent work?

Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as:

1. $Y=mX+c$

Where 'm' represents the slope of the line, and 'c' represents the intercepts on the y-axis.



The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias).

The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called **a point of convergence.**

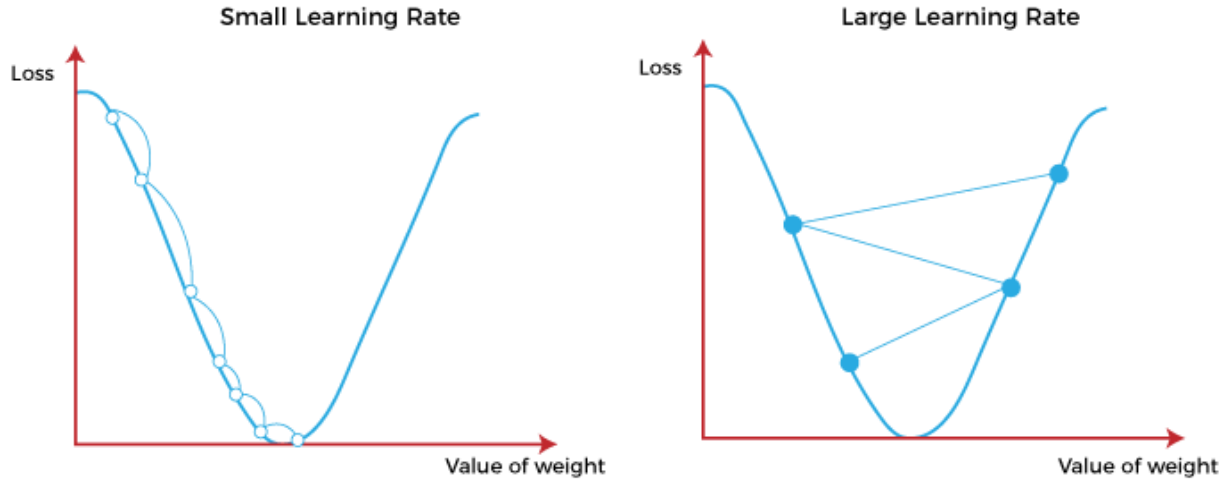
The main objective of gradient descent is to minimize the cost function or the error between expected and actual. To minimize the cost function, two data points are required:

- **Direction & Learning Rate**

These two factors are used to determine the partial derivative calculation of future iteration and allow it to the point of convergence or local minimum or global minimum. Let's discuss learning rate factors in brief;

Learning Rate:

It is defined as the step size taken to reach the minimum or lowest point. This is typically a small value that is evaluated and updated based on the behavior of the cost function. If the learning rate is high, it results in larger steps but also leads to risks of overshooting the minimum. At the same time, a low learning rate shows the small step sizes, which compromises overall efficiency but gives the advantage of more precision.



Radial Basis Kernel is a kernel function that is used in machine learning to find a non-linear classifier or regression line.

What is Kernel Function?

Kernel Function is used to transform n-dimensional input to m-dimensional input, where m is much higher than n then find the dot product in higher dimensional efficiently. The main idea to use kernel is: A linear classifier or

regression curve in higher dimensions becomes a Non-linear classifier or regression curve in lower dimensions.

Mathematical Definition of Radial Basis Kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Radial Basis Kernel

where *x, x'* are vector point in any fixed dimensional space.

But if we expand the above exponential expression, It will go upto infinite power of *x* and *x'*, as expansion of e^x contains infinite terms upto infinite power of *x* hence it involves terms upto infinite powers in infinite dimension.

If we apply any of the algorithms like perceptron Algorithm or linear regression on this kernel, actually we would be applying our algorithm to new infinite-dimensional datapoint we have created. Hence it will give a hyperplane in infinite dimensions, which will give a very strong non-linear classifier or regression curve after returning to our original dimensions.

$$a_1 X^{inf} + a_2 X^{inf-1} + a_3 X^{inf-2} + \ldots + a_{inf} X + C$$

polynomial of infinite power

So, Although we are applying linear classifier/regression it will give a non-linear classifier or regression line, that will be a polynomial of infinite power. And being a polynomial of infinite power, Radial Basis kernel is a very powerful kernel, which can give a curve fitting any complex dataset.

Why Radial Basis Kernel Is much powerful?

The main motive of the kernel is to do calculations in any d-dimensional space where *d > 1*, so that we can get a quadratic, cubic or any polynomial equation of large degree for our classification/regression line. Since Radial basis kernel uses exponent and as we know the expansion of e^x gives a polynomial equation of infinite power, so using this kernel, we make our regression/classification line infinitely powerful too.

Hopfield Neural Network

The Hopfield Neural Networks, invented by Dr John J. Hopfield consists of one layer of *'n'* fully connected recurrent neurons. It is generally used in performing auto-association and optimization tasks. It is calculated using a converging

interactive process and it generates a different response than our normal neural nets.

Discrete Hopfield Network

It is a fully interconnected neural network where each unit is connected to every other unit. It behaves in a discrete manner, i.e. it gives finite distinct output, generally of two types:

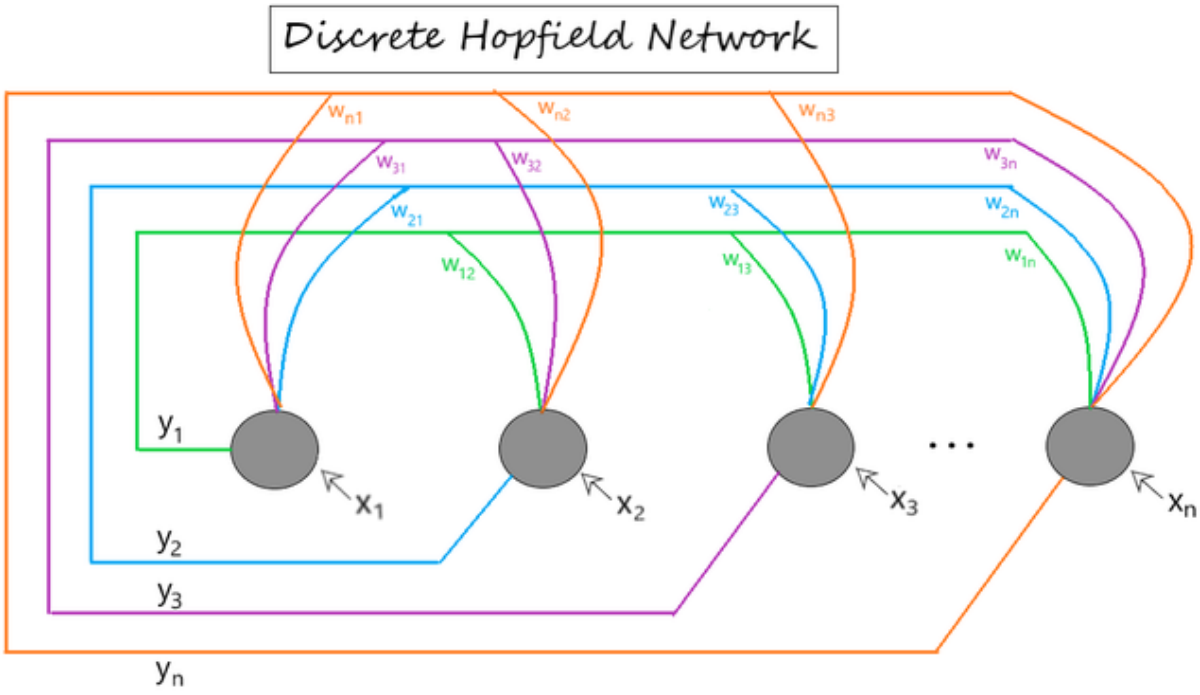
- **Binary (0/1)**
- **Bipolar (-1/1)**

The weights associated with this network are symmetric in nature and have the following properties.

Structure & Architecture of Hopfield Network

- Each neuron has an inverting and a non-inverting output.
- Being fully connected, the output of each neuron is an input to all other neurons but not the self.

The below figure shows a sample representation of a Discrete Hopfield Neural Network architecture having the following elements.



Discrete Hopfield Network Architecture

$[x_1 , x_2 , \dots , x_n]$ -> Input to the n given neurons.
$[y_1 , y_2 , \dots , y_n]$ -> Output obtained from the n given neurons
W_{ij} -> weight associated with the connection between the i^{th} and the j^{th} neuron.

Continuous Hopfield Network

Unlike the discrete Hopfield networks, here the time parameter is treated as a continuous variable. So, instead of getting binary/bipolar outputs, we can obtain values that lie between 0 and 1. It can be used to solve constrained optimization and associative memory problems. The output is defined as:

where,

- v_i = *output from the continuous hopfield network*
- u_i = *internal activity of a node in continuous hopfield network.*

Energy Function

The Hopfield networks have an energy function associated with them. It either diminishes or remains unchanged on update (feedback) after every iteration. The energy function for a continuous Hopfield network is defined as:

To determine if the network will converge to a stable configuration, we see if the energy function reaches its minimum by:

The network is bound to converge if the activity of each neuron wrt time is given by the following differential equation:

What is Recurrent Neural Network (RNN)?

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases when it is required to predict the next word of a sentence, the previous

words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence. The state is also referred to as *Memory State* since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

Architecture Of Recurrent Neural Network

RNNs have the same input and output architecture as any other deep neural architecture. However, differences arise in the way information flows from input to output. Unlike Deep neural networks where we have different weight matrices for each Dense network in RNN, the weight across the network remains the same. It calculates state hidden state H_i for every input X_i. **By using the following formulas:**

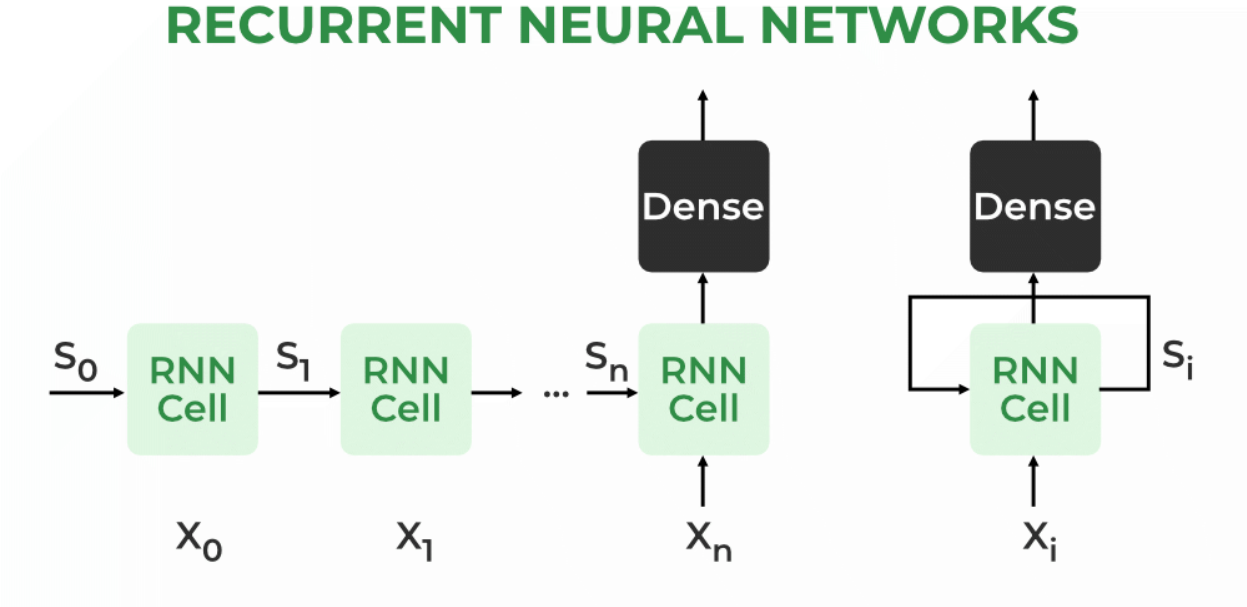
$$h = \sigma(UX + Wh_{-1} + B)$$

$$Y = O(Vh + C) \text{ Hence}$$

$$Y = f(X, h, W, U, V, B, C)$$

Here S is the State matrix which has element si as the state of the network at timestep i

The parameters in the network are W, U, V, c, b which are shared across timestep



What is Recurrent Neural Network

How RNN works

The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the

hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step. This hidden state is updated at every time step to signify the change in the knowledge of the network about the past. The hidden state is updated using the following recurrence relation:-

The formula for calculating the current state:

$$h_t = f(h_{t-1}, x_t)$$

where:

h_t -> current state
h_{t-1} -> previous state
x_t -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{xh}x_t\right)$$

where:

w_{hh} -> weight at recurrent neuron
w_{xh} -> weight at input neuron

The formula for calculating output:

$$y_t = W_{hy}h_t$$

Y_t -> output
W_{hy} -> weight at output layer

These parameters are updated using Backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as Backpropagation through time.

DEEP LEARNING

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—

albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy.

Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services (such as digital assistants, voice-enabled TV remotes, and credit card fraud detection) as well as emerging technologies (such as self-driving cars).

Deep learning neural networks, or artificial neural networks, attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data.

Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called *visible* layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

Another process called backpropagation uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and backpropagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate.

CONVOLUTION NEURAL NETWORKS

To reiterate from the Neural Networks Learn Hub article, neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight an

threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

While we primarily focused on feedforward networks in that article, there are various types of neural nets, which are used for different use cases and data types. For example, recurrent neural networks are commonly used for natural language processing and speech recognition whereas convolutional neural networks (ConvNets or CNNs) are more often utilized for classification and computer vision tasks. Prior to CNNs, manual, time-consuming feature extraction methods were used to identify objects in images. However, convolutional neural networks now provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. That said, they can be computationally demanding, requiring graphical processing units (GPUs) to train models.

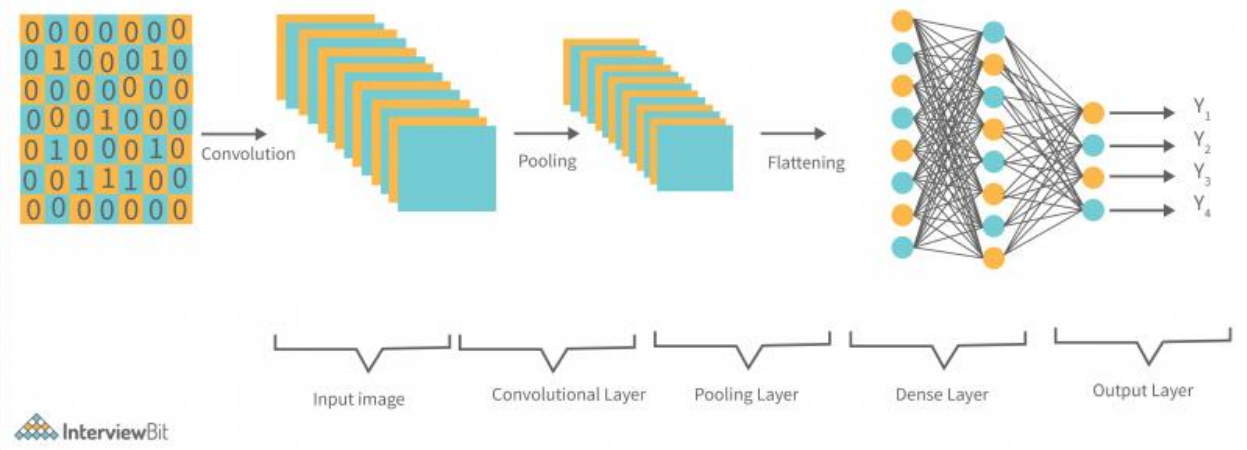
WORKING OF CNN

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

Convolutional layer
Pooling layer
Fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

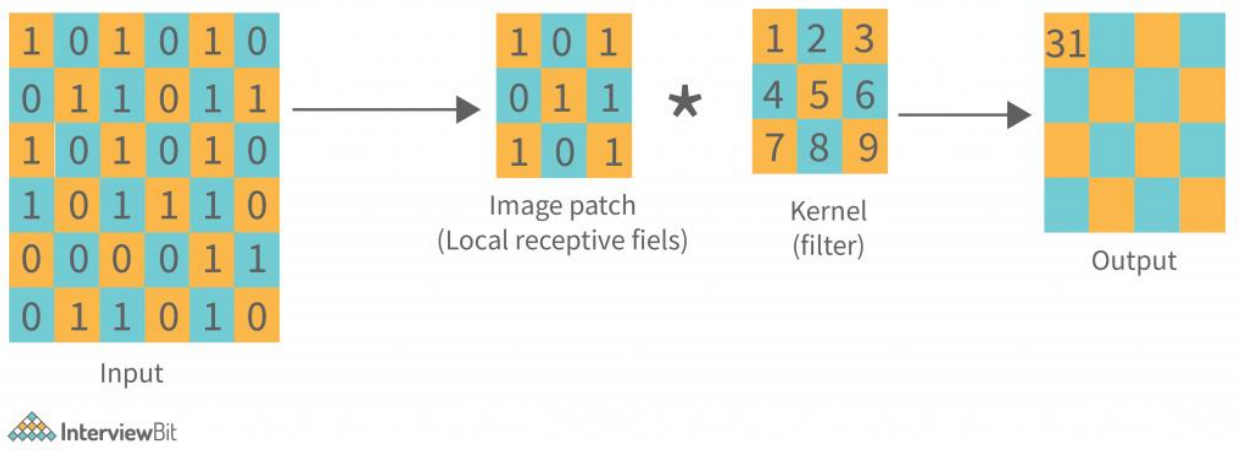
ARCHITECTURE OF CNN



The ConvNet's job is to compress the images into a format that is easier to process while preserving elements that are important for obtaining a decent prediction. This is critical for designing an architecture that is capable of learning features while also being scalable to large datasets.

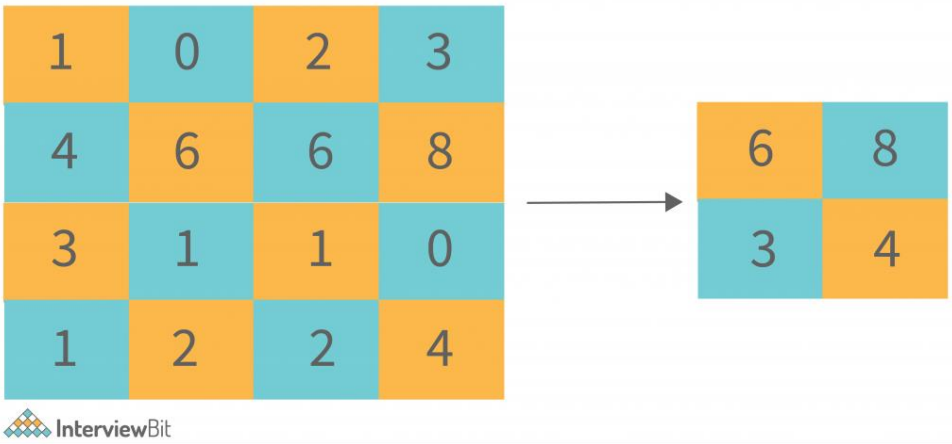
A convolutional neural network, ConvNets in short has three layers which are its building blocks, let's have a look:

Convolutional Layer (CONV): They are the foundation of CNN, and they are in charge of executing convolution operations. The Kernel/Filter is the component in this layer that performs the convolution operation (matrix). Until the complete image is scanned, the kernel makes horizontal and vertical adjustments dependent on the stride rate. The kernel is less in size than a picture, but it has more depth. This means that if the image has three (RGB) channels, the kernel height and width will be modest spatially, but the depth will span all three.

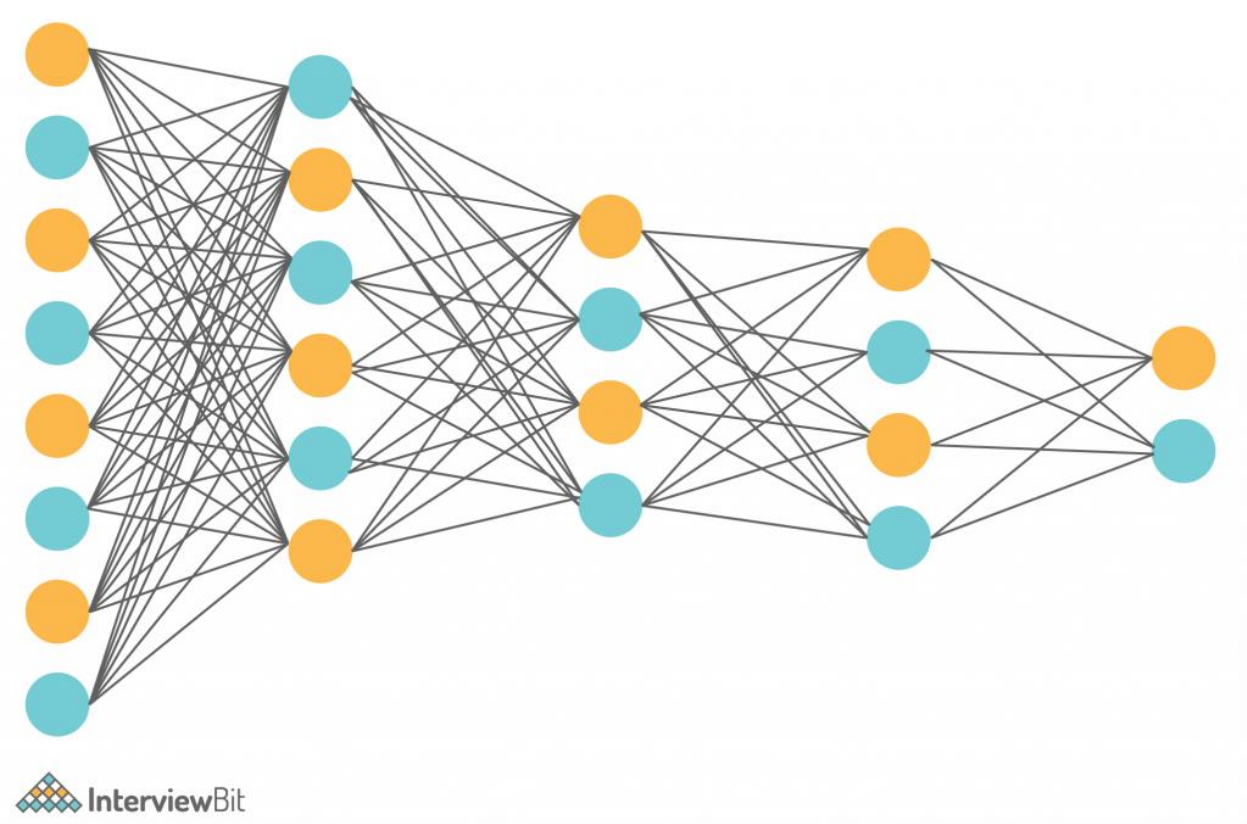


Other than convolution, there is another important part of convolutional layers, known as the Non-linear activation function. The outputs of the linear operations like convolution are passed through a non-linear activation function. Although smooth nonlinear functions such as the sigmoid or hyperbolic tangent (tanh) function were formerly utilized because they are mathematical representations of biological neuron actions. The rectified linear unit (ReLU) is now the most commonly used non-linear activation function. f(x) = max(0, x)

Pooling Layer (POOL): This layer is in charge of reducing dimensionality. It aids in reducing the amount of computing power required to process the data. Pooling can be divided into two types: maximum pooling and average pooling. The maximum value from the area covered by the kernel on the image is returned by max pooling. The average of all the values in the part of the image covered by the kernel is returned by average pooling.



Fully Connected Layer (FC): The fully connected layer (FC) works with a flattened input, which means that each input is coupled to every neuron. After that, the flattened vector is sent via a few additional FC layers, where the mathematical functional operations are normally performed. The classification procedure gets started at this point. FC layers are frequently found near the end of CNN architectures if they are present.



Along with the above layers, there are some additional terms that are part of a CNN architecture.

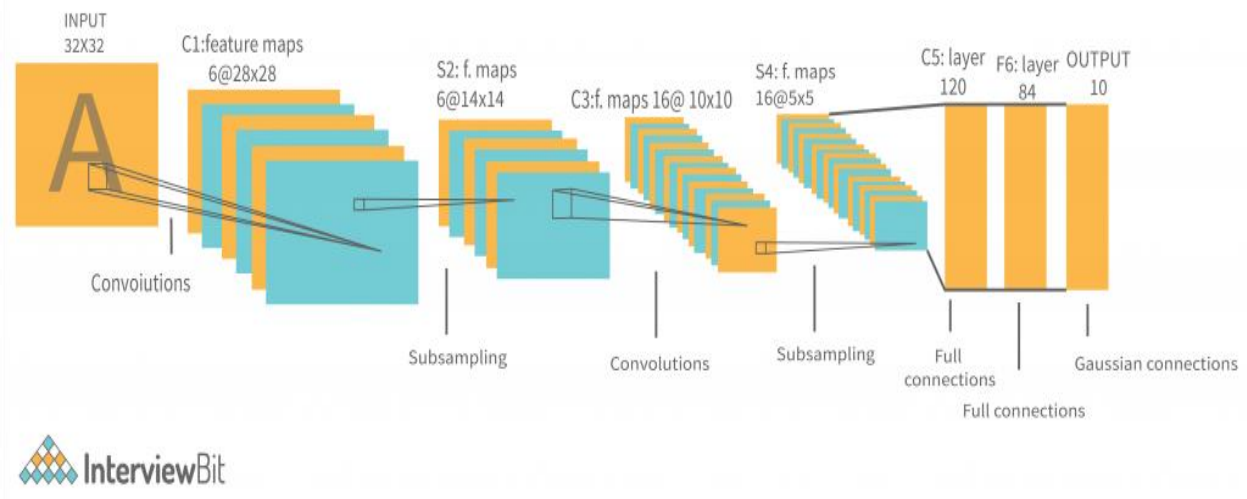
Activation Function: The last fully connected layer's activation function is frequently distinct from the others. Each activity necessitates the selection of an appropriate activation function. The softmax function, which normalizes output real values from the last fully connected layer to target class probabilities, where each value ranges between 0 and 1 and all values total to 1, is an activation function used in the multiclass classification problem.

Dropout Layers: The Dropout layer is a mask that nullifies some neurons' contributions to the following layer while leaving all others unchanged. A Dropout layer can be applied to the input vector, nullifying some of its properties; however, it can also be applied to a hidden layer, nullifying some hidden neurons. Dropout layers are critical in CNN training because they prevent the training data from overfitting. If they aren't there, the first batch of training data has a disproportionately large impact on learning. As a result, learning of traits that occur only in later samples or batches would be prevented:

Now you have got a good understanding of the building blocks of CNN, let's have a look to some of the popular CNN architecture.

LeNet ARCHITECTURE

The LeNet architecture is simple and modest making it ideal for teaching the fundamentals of CNNs. It can even run on the CPU (if your system lacks a decent GPU), making it an excellent "first CNN." It's one of the first and most extensively used CNN designs, and it's been used to successfully recognize handwritten digits. The LeNet-5 CNN architecture has seven layers. Three convolutional layers, two subsampling layers, and two fully linked layers make up the layer composition.



AlexNet ARCHITECTURE

AlexNet's architecture was extremely similar to LeNet's. It was the first convolutional network to employ the graphics processing unit (GPU) to improve performance.

Convolutional filters and a non-linear activation function termed ReLU are used in each convolutional layer (Rectified Linear Unit). Max pooling is done using the pooling layers. Due to the presence of fully connected layers, the input size is fixed. The AlexNet architecture was created with large-scale image datasets in mind, and it produced state-of-the-art results when it was first released. It has 60 million characteristics in all.