UNIT III

DEEP LEARNING

INTRODUCTION NEURAL NETWORKS AND DEEP LEARNING

Neural Networks:

- Neural networks are artificial systems that were inspired by biological neural networks. These systems learn to perform tasks by being exposed to various datasets and examples without any task-specific rules.
- The idea is that the system generates identifying characteristics from the data they have been passed without being programmed with a pre-programmed understanding of these datasets. Neural networks are based on computational models for threshold logic.
- Threshold logic is a combination of algorithms and mathematics.
- Neural networks are based either on the study of the brain or on the application of neural networks to artificial intelligence.
- The work has led to improvements in finite automata theory. Components of a typical neural network involve neurons, connections which are known as synapses, weights, biases, propagation function, and a learning rule.
- Neurons will receive an input from predecessor neurons that have an activation , threshold , an activation function f, and an output function .
- Connections consist of connections, weights and biases which rules how neuron transfers output to neuron .
- Propagation computes the input and outputs the output and sums the predecessor neurons function with the weight.
- The learning of neural network basically refers to the adjustment in the free parameters i.e. weights and bias.
- There are basically three sequence of events of learning process.

These includes:

1. The neural network is simulated by an new environment.

2. Then the free parameters of the neural network is changed as a result of this simulation.
3. The neural network then responds in a new way to the environment because of the changes in its free parameters.

Supervised vs Unsupervised Learning:

- Neural networks learn via supervised learning; Supervised machine learning involves an input variable x and corresponding desired output variable y.
- Here we introduce the concept of teacher who has knowledge about the environment.
- Thus we can say that the teacher has both input-output set.
- The neural network is unaware of the environment. The input is exposed to both teacher and neural network, the neural network generates an output based on the input.
- This output is then compared with the desired output that teacher has and simultaneously an error signal is produced.
- The free parameters of the network is then step by step adjusted so that error is minimum. The learning stops when the algorithm reaches an acceptable level of performance. Unsupervised machine learning has input data X and no corresponding output variables.
- The goal is to model the underlying structure of the data to understand more about the data. The keywords for supervised machine learning are classification and regression.
- For unsupervised machine learning, the keywords are clustering and association.

Evolution of Neural Networks:

- Hebbian learning deals with neural plasticity.
- Hebbian learning is unsupervised and deals with long-term potentiation.
- Hebbian learning deals with pattern recognition and exclusive-or circuits deal with if-then rules.
- Backpropagation solved the exclusive-or issue that Hebbian learning could not handle.
- This also allowed for multi-layer networks to be feasible and efficient.
- If an error was found, the error was solved at each layer by modifying the weights at each node.

- This led to the development of support vector machines, linear classifiers, and max-pooling. The vanishing gradient problem affects feedforward networks that use back propagation and recurrent neural network.
- This is known as deep-learning.
- Hardware-based designs are used for biophysical simulation and neurotrophic computing. They have large scale component analysis and convolution creates new class of neural computing with analog.
- This also solved back-propagation for many-layered feedforward neural networks. Convolutional networks are used for alternating between convolutional layers and max-pooling layers with connected layers (fully or sparsely connected) with a final classification layer.
- The learning is done without unsupervised pre-training. Each filter is equivalent to a weights vector that has to be trained.
- The shift variance has to be guaranteed to dealing with small and large neural networks. This is being resolved in Development Networks.
- Some of the other learning techniques involve error-correction learning, memory-based learning and competitive learning.

Types of Neural Networks

There are seven types of neural networks that can be used:

- **Multilayer Perceptron (MLP):** A type of feedforward neural network with three or more layers, including an input layer, one or more hidden layers, and an output layer. It uses nonlinear activation functions.
- **Convolutional Neural Network (CNN):** A neural network that is designed to process input data that has a grid-like structure, such as an image. It uses convolutional layers and pooling layers to extract features from the input data.
- **Recursive Neural Network (RNN):** A neural network that can operate on input sequences of variable length, such as text. It uses weights to make structured predictions.
- **Recurrent Neural Network (RNN):** A type of neural network that makes connections between the neurons in a directed cycle, allowing it to process sequential data.

- **Long Short-Term Memory (LSTM):** A type of RNN that is designed to overcome the vanishing gradient problem in training RNNs. It uses memory cells and gates to selectively read, write, and erase information.
- **Sequence-to-Sequence (Seq2Seq):** A type of neural network that uses two RNNs to map input sequences to output sequences, such as translating one language to another.
- **Shallow Neural Network:** A neural network with only one hidden layer, often used for simpler tasks or as a building block for larger networks.

These neural networks are applications of the basic neural network demonstrated below.

For the example, the neural network will work with three vectors: a vector of attributes X, a vector of classes Y, and a vector of weights W. The code will use 100 iterations to fit the attributes to the classes. The predictions are generated, weighed, and then outputted after iterating through the vector of weights W. The neural network handles backpropagation.

What is neural network Representation?

- A neural network can be understood as a network of hidden layers, an input layer and an output layer that tries to mimic the working of a human brain.
- The hidden layers can be visualized as an abstract representation of the input data itself.
- A neural network can be understood as a network of hidden layers, an input layer and an output layer that tries to mimic the working of a human brain.
- The hidden layers can be visualized as an abstract representation of the input data itself.
- These layers help the neural network understand various features of the data with the help of its own internal logic.
- These neural networks are non-interpretable models.
- Non-interpretable models are those which can’t be interpreted or understood even if we observe the hidden layers.
- This is because the neural networks have an internal logic working on its own, that can’t be comprehended by us.

- We can just see then as a vector of numerical values. Since the output of a neural network is a numerical vector, we need to have an explicit output layer that bridges the gap between the actual data and the representation of the data by the network.
- An output layer can be understood as a translator that helps us to understand the logic of the network and convert the target values.
- A theorem named ‘Universal approximation theorem’ tells that a feedforward network that contains one hidden layer can be used to represent any function.
- This means there is no limit on the functioning of a neural network that contains one hidden layer. But in real life situations, a neural network with one hidden layer can’t be used well.
- A neural network is a mathematical model that helps in processing information. It is not a set of lines of code, but a model or a system that helps process the inputs/information and gives result.
- The information is processed in the simplest form over basic elements known as ‘neurons’. Neurons are connected and help exchange signals/information between them with the help of connection links.
- This connection links between neurons could be strong or weak, and this strength of the connection links determines the method in which information is processed.
- Every neuron has an internal state which can be determined by the incoming connections from other neurons.
- Every neuron has an activation function which is calculated on its state, and this helps determine its output signal.
- A neural network can be understood as a computational graph of mathematical operations.

Two main characteristics of a neural network −

- Architecture
- Learning

Architecture:

It tells about the connection type: whether it is feedforward, recurrent, multi-layered, convolutional, or single layered. It also tells about the number of layers and the number of neurons in every layer.

Learning:

It tells about the method in which the neural network is trained. A common way to train a neural network is to use gradient descent and backpropagation.

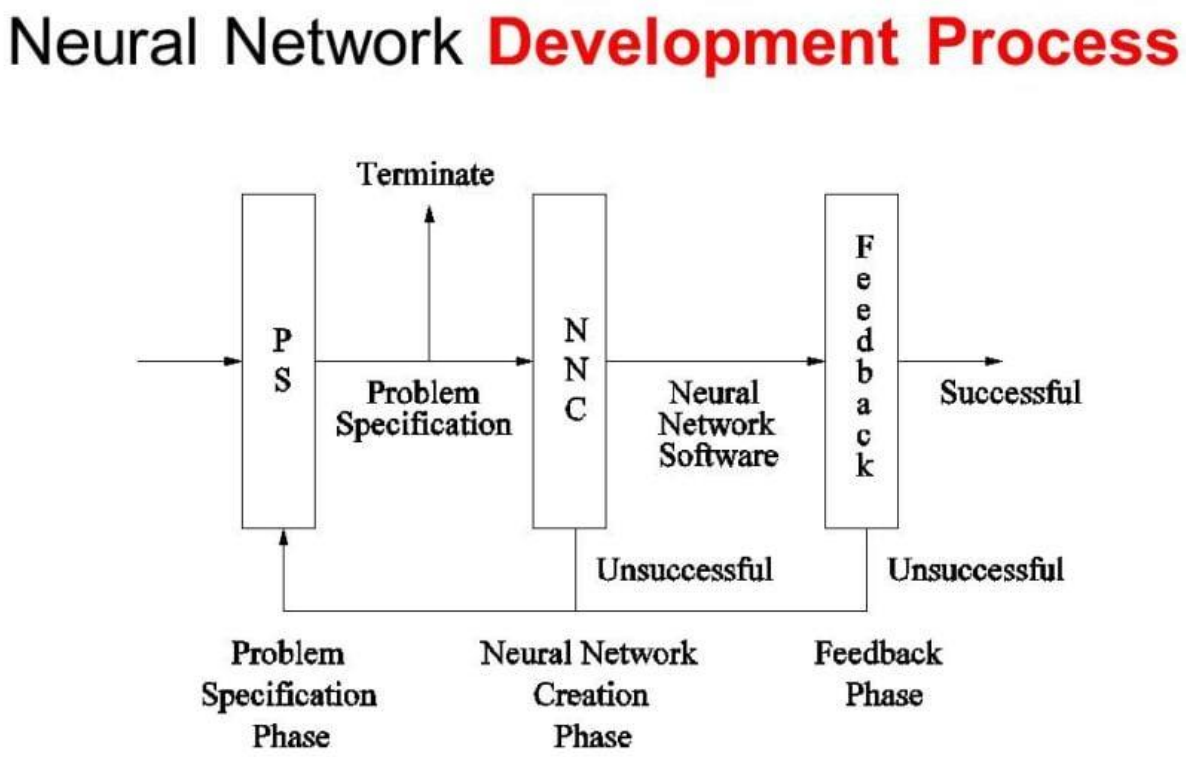
Problems in neural networks:

4 Disadvantages of Neural Networks & Deep Learning:

- Black box.
- Duration of development.
- Amount of data.
- Computationally expensive.

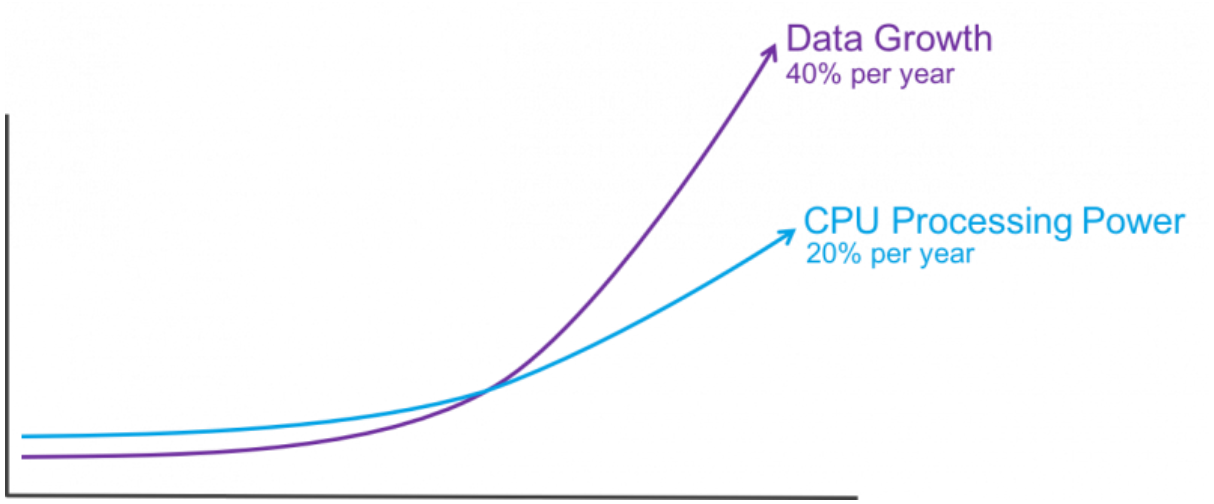
1. BLACK BOX

2. DURATION OF DEVELOPMENT:



3. AMOUNT OF DATA:

- Neural networks usually require much more data than traditional machine learning algorithms, as in at least thousands if not millions of labeled samples.
- This isn't an easy problem to deal with and many machine learning problems can be solved well with less data if you use other algorithms.
- Although there are some cases where neural networks do well with little data, most of the time they don't.
- In this case, a simple algorithm like naive Bayes, which deals much better with little data, would be the appropriate choice.



4. COMPUTATIONALLY EXPENSIVE:

- Usually, neural networks are also more computationally expensive than traditional algorithms.

- State of the art deep learning algorithms, which realize successful training of really deep neural networks, can take several weeks to train completely from scratch.

- By contrast, most traditional machine learning algorithms take much less time to train, ranging from a few minutes to a few hours or days.

- The amount of computational power needed for a neural network depends heavily on the size of your data, but also on the depth and complexity of your network.

- For example, a neural network with one layer and 50 neurons will be much faster than a random forest with 1,000 trees.

- By comparison, a neural network with 50 layers will be much slower than a random forest with only 10 trees.

Perceptron in Machine Learning

- In Machine Learning and Artificial Intelligence, Perceptron is the most commonly used term for all folks.

- It is the primary step to learn Machine Learning and Deep Learning technologies, which consists of a set of weights, input values or scores, and a threshold.

- Perceptron is a building block of an Artificial Neural Network. Initially, in the mid of 19^{th} century, Mr. Frank Rosenblatt invented the Perceptron for performing certain calculations to detect input data capabilities or business intelligence.

- Perceptron is a linear Machine Learning algorithm used for supervised learning for various binary classifiers.

- This algorithm enables neurons to learn elements and processes them one by one during preparation.

- In this tutorial, "Perceptron in Machine Learning," we will discuss in-depth knowledge of Perceptron and its basic functions in brief. Let's start with the basic introduction of Perceptron.

What is the Perceptron model in Machine Learning?

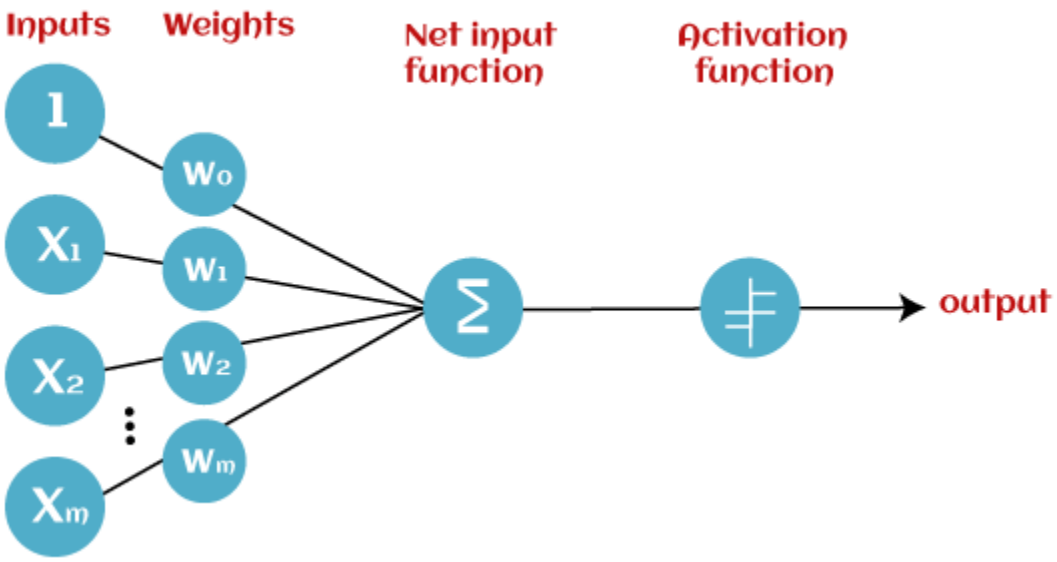
- Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks.
- Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence.
- Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks.
- However, it is a supervised learning algorithm of binary classifiers.
- Hence, we can consider it as a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.

What is Binary classifier in Machine Learning?

- In Machine Learning, binary classifiers are defined as the function that helps in deciding whether input data can be represented as vectors of numbers and belongs to some specific class.
- Binary classifiers can be considered as linear classifiers. In simple words, we can understand it as a classification algorithm that can predict linear predictor function in terms of weight and feature vectors.

Basic Components of Perceptron

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



- **Input Nodes or Input Layer:**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias:**

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength

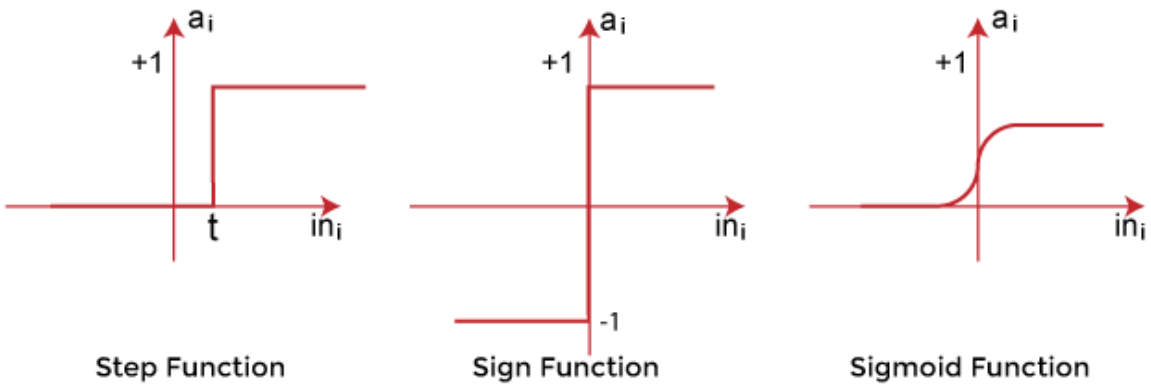
of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function:**

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function

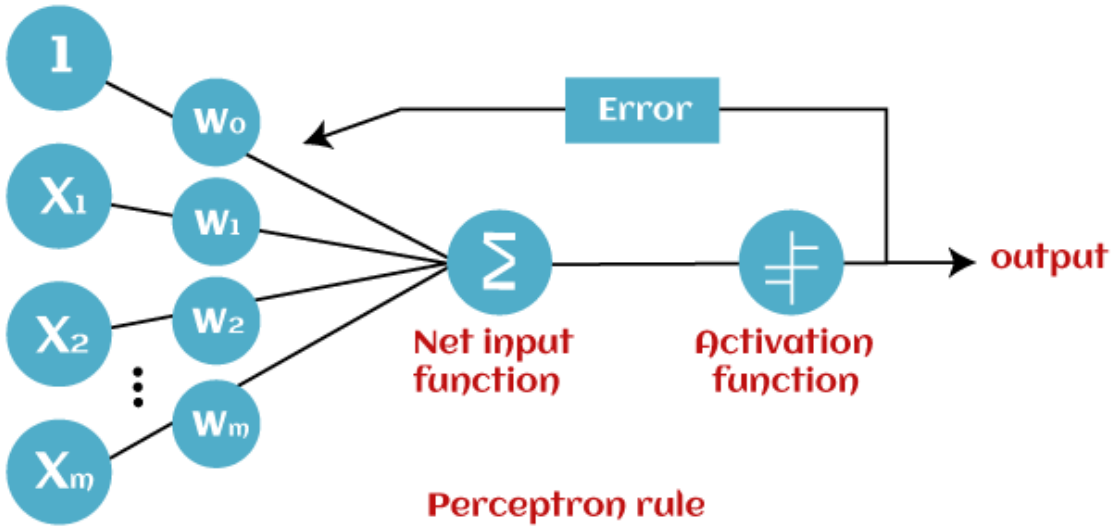


- The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs.
- Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

- In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function.
- The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum.

- Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the step function and is represented by 'f'.



- This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node.
- Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$\sum wi*xi = x1*w1 + x2*w2 + \ldots wn*xn$

Add a special term called bias 'b' to this weighted sum to improve the model's performance.

$\sum wi*xi + b$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$Y = f(\sum wi*xi + b)$

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Single Layer Perceptron Model:

- This is one of the easiest Artificial neural networks (ANN) types.
- A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model.
- The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.
- In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters.
- Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.
- If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change.
- However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model.
- Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.
- "Single-layer perceptron can learn only linearly separable patterns."

Multi-Layered Perceptron Model:

- Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- **Forward Stage:** Activation functions start from the input layer in the forward stage and terminate on the output layer.
- **Backward Stage:** In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.
- Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model. Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.
- A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages of Multi-Layer Perceptron:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages of Multi-Layer Perceptron:

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

f(x)=1; if w.x+b>0

otherwise, f(x)=0

- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

Characteristics of Perceptron

The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

Limitations of Perceptron Model

A perceptron model has limitations as follows:

- The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function.
- Perceptron can only be used to classify the linearly separable sets of input vectors. If input vectors are non-linear, it is not easy to classify them properly.

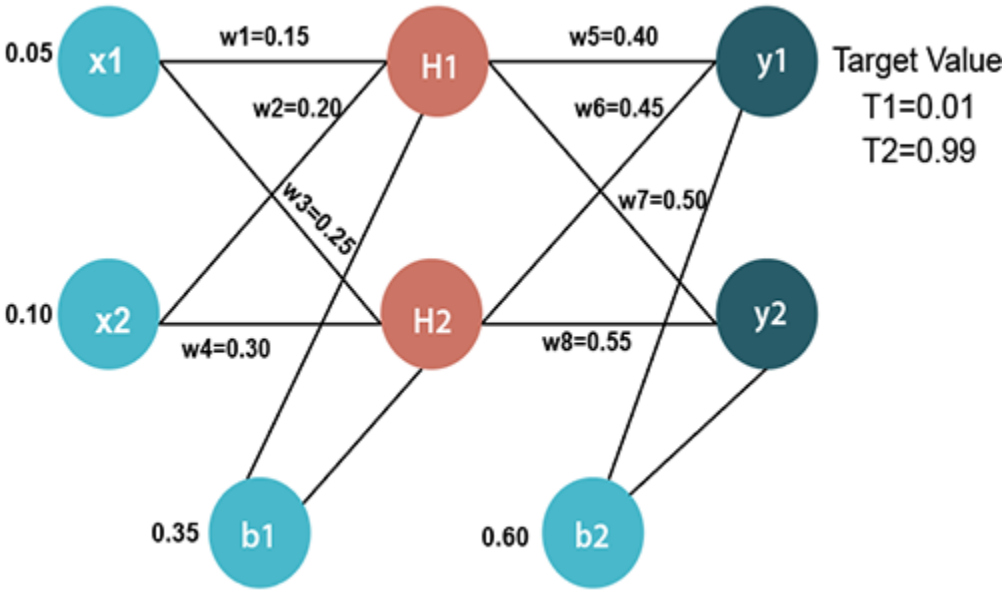
Future of Perceptron

- The future of the Perceptron model is much bright and significant as it helps to interpret data by building intuitive patterns and applying them in the future.
- Machine learning is a rapidly growing technology of Artificial Intelligence that is continuously evolving and in the developing phase; hence the future of perceptron technology will continue to support and facilitate analytical behavior in machines that will, in turn, add to the efficiency of computers.
- The perceptron model is continuously becoming more advanced and working efficiently on complex problems with the help of artificial neurons.

Back propagation Process in Deep Neural Network

- Back propagation is one of the important concepts of a neural network.
- Our task is to classify our data best.
- For this, we have to update the weights of parameter and bias, but how can we do that in a deep neural network? In the linear regression model, we use gradient descent to optimize the parameter.
- Similarly here we also use gradient descent algorithm using Back propagation.
- For a single training example, Back propagation algorithm calculates the gradient of the error function.
- Back propagation can be written as a function of the neural network. back propagation algorithms are a set of methods used to efficiently train artificial neural networks following a gradient descent approach which exploits the chain rule.
- The main features of Backpropagation are the iterative, recursive and efficient method through which it calculates the updated weight to improve the network until it is not able to perform the task for which it is being trained.
- Derivatives of the activation function to be known at network design time is required to Backpropagation.

- Now, how error function is used in Backpropagation and how Backpropagation works? Let start with an example and do it mathematically to understand how exactly updates the weight using Backpropagation.



Input values

X1=0.05

X2=0.10

Initial weight

W1=0.15 w5=0.40

W2=0.20 w6=0.45

W3=0.25 w7=0.50

W4=0.30 w8=0.55

Bias Values

b1=0.35 b2=0.60

Target Values

T1=0.01

T2=0.99

Now, we first calculate the values of H1 and H2 by a forward pass.

Forward Pass

To find the value of H1 we first multiply the input value from the weights as

$$H1=x1\times w_1+x2\times w_2+b1$$

$$H1=0.05\times 0.15+0.10\times 0.20+0.35$$

$$H1=0.3775$$

To calculate the final result of H1, we performed the sigmoid function as

$$H1_{final}=\frac{1}{1+\frac{1}{e^{H1}}}$$

$$H1_{final}=\frac{1}{1+\frac{1}{e^{0.3775}}}$$

$$\mathbf{H1_{final}=0.593269992}$$

We will calculate the value of H2 in the same way as H1

$$H2=x1\times w_3+x2\times w_4+b1$$

$$H2=0.05\times 0.25+0.10\times 0.30+0.35$$

$$H2=0.3925$$

To calculate the final result of H1, we performed the sigmoid function as

$$H2_{final} = \frac{1}{1 + \frac{1}{e^{H2}}}$$

$$H2_{final} = \frac{1}{1 + \frac{1}{e^{0.3925}}}$$

$$\mathbf{H2_{final} = 0.596884378}$$

Now, we calculate the values of y1 and y2 in the same way as we calculate the H1 and H2.

To find the value of y1, we first multiply the input value i.e., the outcome of H1 and H2 from the weights as

$$y1 = H1 \times w_5 + H2 \times w_6 + b2$$

$$y1 = 0.593269992 \times 0.40 + 0.596884378 \times 0.45 + 0.60$$

$$y1 = 1.10590597$$

To calculate the final result of y1 we performed the sigmoid function as

$$y1_{final} = \frac{1}{1 + \frac{1}{e^{y1}}}$$

$$y1_{final} = \frac{1}{1 + \frac{1}{e^{1.10590597}}}$$

$$\mathbf{y1_{final} = 0.75136507}$$

We will calculate the value of y2 in the same way as y1

$$y2 = H1 \times w_7 + H2 \times w_8 + b2$$

$$y2 = 0.593269992 \times 0.50 + 0.596884378 \times 0.55 + 0.60$$

$$y2 = 1.2249214$$

To calculate the final result of H1, we performed the sigmoid function as

$$y2_{final} = \frac{1}{1 + \frac{1}{e^{y2}}}$$

$$y2_{final} = \frac{1}{1 + \frac{1}{e^{1.2249214}}}$$

$$\mathbf{y2_{final} = 0.772928465}$$

Our target values are 0.01 and 0.99. Our y1 and y2 value is not matched with our target values T1 and T2.

Now, we will find the total error, which is simply the difference between the outputs from the target outputs. The total error is calculated as

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

So, the total error is

$$= \frac{1}{2}(t1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

$$= \frac{1}{2}(0.01 - 0.75136507)^2 + \frac{1}{2}(0.99 - 0.772928465)^2$$

$$= 0.274811084 + 0.0235600257$$

$$\mathbf{E_{total} = 0.29837111}$$

Now, we will backpropagate this error to update the weights using a backward pass.

Backward pass at the output layer

To update the weight, we calculate the error correspond to each weight with the help of a total error. The error on weight w is calculated by differentiating total error with respect to w.

$$Error_w = \frac{\partial E_{total}}{\partial w}$$

We perform backward process so first consider the last weight w5 as

$$\text{Error}_{w5} = \frac{\partial E_{total}}{\partial w5} \dots\dots\dots (1)$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2 \dots\dots\dots (2)$$

From equation two, it is clear that we cannot partially differentiate it with respect to w5 because there is no any w5. We split equation one into multiple terms so that we can easily differentiate it with respect to w5 as

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1\_final}{\partial y1} \times \frac{\partial y1}{\partial w5} \dots\dots\dots (3)$$

Now, we calculate each term one by one to differentiate E_{total} with respect to w5 as

$$\frac{\partial E_{total}}{\partial y1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial y1_{final}}$$

$$= 2 \times \frac{1}{2} \times (T1 - y1_{final})^{2-1} \times (-1) + 0$$

$$= -(T1 - y1_{final})$$

$$= -(0.01 - 0.75136507)$$

$$\mathbf{\frac{\partial E_{total}}{\partial y1_{final}} = 0.74136507} \dots\dots\dots (4)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}} \dots\dots\dots (5)$$

$$\frac{\partial y1_{final}}{\partial y1} = \frac{\partial(\frac{1}{1 + e^{-y1}})}{\partial y1}$$

$$= \frac{e^{-y1}}{(1 + e^{-y1})^2}$$

$$= e^{-y1} \times (y1_{final})^2 \dots\dots\dots (6)$$

$$y1_{final} = \frac{1}{1 + e^{-y1}}$$

$$e^{-y1} = \frac{1 - y1_{final}}{y1_{final}} \dots\dots\dots (7)$$

Putting the value of e^{-y} in equation (5)

$$= \frac{1 - y1_{final}}{y1_{final}} \times (y1_{final})^2$$

$$= y1_{final} \times (1 - y1_{final})$$

$$= 0.75136507 \times (1 - 0.75136507)$$

$$\mathbf{\frac{\partial y1_{final}}{\partial y1} = 0.186815602 \ldots\ldots\ldots (8)}$$

$$y1 = H1_{final} \times w5 + H2_{final} \times w6 + b2 \ldots\ldots\ldots\ldots\ldots\ldots (9)$$

$$\frac{\partial y1}{\partial w5} = \frac{\partial (H1_{final} \times w5 + H2_{final} \times w6 + b2)}{\partial w5}$$

$$= H1_{final}$$

$$\mathbf{\frac{\partial y1}{\partial w5} = 0.596884378 \ldots\ldots\ldots (10)}$$

So, we put the values of $\frac{\partial E_{total}}{\partial y1_{final}}, \frac{\partial y1_{final}}{\partial y1}$, and $\frac{\partial y1}{\partial w5}$ in equation no (3) to find the final result.

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1} \times \frac{\partial y1}{\partial w5}$$

$$= 0.74136507 \times 0.186815602 \times 0.593269992$$

$$\mathbf{Error_{w5} = \frac{\partial E_{total}}{\partial w5} = 0.0821670407 \ldots\ldots\ldots (11)}$$

Now, we will calculate the updated weight $w5_{new}$ with the help of the following formula

$$w5_{new} = w5 - \eta \times \frac{\partial E_{total}}{\partial w5} \text{ Here, } \eta = \text{learning rate} = 0.5$$

$$= 0.4 - 0.5 \times 0.0821670407$$

$$\mathbf{w5_{new} = 0.35891648 \ldots\ldots\ldots (12)}$$

In the same way, we calculate $w6_{new}$,$w7_{new}$, and $w8_{new}$ and this will give us the following values

$w5_{new}$=0.35891648

$w6_{new}$=408666186

$w7_{new}$=0.511301270

$w8_{new}$=0.561370121

Multilayer Networks:

- A multilayer perceptron (MLP) is a feed forward artificial neural network that generates a set of outputs from a set of inputs.
- An MLP is characterized by several layers of input nodes connected as a directed graph between the input nodes connected as a directed graph between the input and output layers.

Backward pass at Hidden layer

- Now, we will backpropagate to our hidden layer and update the weight w1, w2, w3, and w4 as we have done with w5, w6, w7, and w8 weights.
- We will calculate the error at w1 as

$$\text{Error}_{w1} = \frac{\partial E_{total}}{\partial w1}$$

$$E_{total} = \frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2$$

From equation (2), it is clear that we cannot partially differentiate it with respect to w1 because there is no any w1. We split equation (1) into multiple terms so that we can easily differentiate it with respect to w1 as

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1\_final}{\partial H1} \times \frac{\partial H1}{\partial w1} \dots\dots\dots\dots (13)$$

Now, we calculate each term one by one to differentiate E_{total} with respect to w1 as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2 + \frac{1}{2}(T2 - y2_{final})^2)}{\partial H1} \dots\dots\dots\dots (14)$$

We again split this because there is no any $H1^{final}$ term in E^{toatal} as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}} \dots\dots\dots (15)$$

$\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ will again split because in E1 and E2 there is no H1 term. Splitting is done as

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}} \dots\dots\dots (16)$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final} \dots\dots\dots (17)$$

We again Split both $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ because there is no any y1 and y2 term in E1 and E2. We split it as

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1} \dots\dots\dots (18)$$

$$\frac{\partial E_2}{\partial y2} = \frac{\partial E_2}{\partial y2_{fianl}} \times \frac{\partial y2_{final}}{\partial y2} \dots\dots\dots (19)$$

Now, we find the value of $\frac{\partial E_1}{\partial y1}$ and $\frac{\partial E_2}{\partial y2}$ by putting values in equation (18) and (19) as

From equation (18)

$$\frac{\partial E_1}{\partial y1} = \frac{\partial E_1}{\partial y1_{fianl}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= \frac{\partial(\frac{1}{2}(T1 - y1_{final})^2)}{\partial y1_{final}} \times \frac{\partial y1_{final}}{\partial y1}$$

$$= 2 \times \frac{1}{2}(T1 - y1_{final}) \times (-1) \times \frac{\partial y1_{final}}{\partial y1}$$

From equation (8)

$$= 2 \times \frac{1}{2}(0.01 - 0.75136507) \times (-1) \times 0.186815602$$

$$\frac{\partial \mathbf{E_1}}{\partial \mathbf{y1}} = \mathbf{0.138498562} \ldots\ldots\ldots (20)$$

From equation (19)

$$\frac{\partial \mathrm{E_2}}{\partial \mathrm{y2}} = \frac{\partial \mathrm{E_2}}{\partial \mathrm{y2_{fianl}}} \times \frac{\partial \mathrm{y2_{final}}}{\partial \mathrm{y2}}$$

$$= \frac{\partial(\frac{1}{2}(\mathrm{T2} - \mathrm{y2_{final}})^2)}{\partial \mathrm{y2_{final}}} \times \frac{\partial \mathrm{y2_{final}}}{\partial \mathrm{y2}}$$

$$= 2 \times \frac{1}{2}(\mathrm{T2} - \mathrm{y2_{final}}) \times (-1) \times \frac{\partial \mathrm{y2_{final}}}{\partial \mathrm{y2}} \ldots\ldots\ldots (21)$$

$$\mathrm{y2_{final}} = \frac{1}{1 + \mathrm{e}^{-\mathrm{y2}}} \ldots\ldots\ldots\ldots\ldots\ldots (22)$$

$$\frac{\partial \mathrm{y2_{fianl}}}{\partial \mathrm{y2}} = \frac{\partial(\frac{1}{1 + \mathrm{e}^{-\mathrm{y2}}})}{\partial \mathrm{y2}}$$

$$= \frac{\mathrm{e}^{-\mathrm{y2}}}{(1 + \mathrm{e}^{-\mathrm{y2}})^2}$$

$$= \mathrm{e}^{-\mathrm{y2}} \times (\mathrm{y2_{final}})^2 \ldots\ldots\ldots\ldots\ldots (23)$$

$$\mathrm{y2_{final}} = \frac{1}{1 + \mathrm{e}^{-\mathrm{y2}}}$$

$$\mathrm{e}^{-\mathrm{y2}} = \frac{1 - \mathrm{y2_{final}}}{\mathrm{y2_{final}}} \ldots\ldots\ldots\ldots\ldots (24)$$

Putting the value of e^{-y2} in **equation (23)**

$$= \frac{1 - y2_{final}}{y2_{final}} \times (y2_{final})^2$$

$$= y2_{final} \times (1 - y2_{final})$$

$$= 0.772928465 \times (1 - 0.772928465)$$

$$\frac{\partial y2_{fianl}}{\partial y2} = \mathbf{0.175510053} \ldots\ldots\ldots (\mathbf{25})$$

From equation (21)

$$= 2 \times \frac{1}{2}(0.99 - 0.772928465) \times (-1) \times 0.175510053$$

$$\frac{\partial E_1}{\partial y1} = \mathbf{-0.0380982366126414} \ldots\ldots\ldots (\mathbf{26})$$

Now from **equation (16) and (17)**

$$\frac{\partial E_1}{\partial H1_{final}} = \frac{\partial E_1}{\partial y1} \times \frac{\partial y1}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial(H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times \frac{\partial(H1_{final} \times w_5 + H2_{final} \times w_6 + b2)}{\partial H1_{final}}$$

$$= 0.138498562 \times w5$$

$$= 0.138498562 \times 0.40$$

$$\mathbf{\frac{\partial E_1}{\partial H1_{final}} = 0.0553994248 \ldots\ldots\ldots (27)}$$

$$\frac{\partial E_2}{\partial H1_{final}} = \frac{\partial E_2}{\partial y2} \times \frac{\partial y2}{\partial H1\_final}$$

$$= -0.0380982366126414 \times \frac{\partial(H1_{final} \times w_7 + H2_{final} \times w_8 + b2)}{\partial H1_{final}}$$

$$= -0.0380982366126414 \times w7$$

$$= -0.0380982366126414 \times 0.50$$

$$\mathbf{\frac{\partial E_2}{\partial H1_{final}} = -0.0190491183063207 \ldots\ldots\ldots (28)}$$

Put the value of $\frac{\partial E_1}{\partial H1_{final}}$ and $\frac{\partial E_2}{\partial H1_{final}}$ in **equation (15)** as

$$\frac{\partial E_{total}}{\partial H1_{final}} = \frac{\partial E_1}{\partial H1_{final}} + \frac{\partial E_2}{\partial H1_{final}}$$

$$= 0.0553994248 + (-0.0190491183063207)$$

$$\mathbf{\frac{\partial E_{total}}{\partial H1_{final}} = 0.0364908241736793 \ldots\ldots\ldots (29)}$$

We have $\frac{\partial E_{total}}{\partial H1_{final}}$, we need to figure out $\frac{\partial H1\_final}{\partial H1}$, $\frac{\partial H1}{\partial w1}$ as

$$\frac{\partial H1_{final}}{\partial H1} = \frac{\partial(\frac{1}{1+e^{-H1}})}{\partial H1}$$

$$= \frac{e^{-H1}}{(1+e^{-H1})^2}$$

$$e^{-H1} \times (H1_{final})^2 \dots\dots\dots (30)$$

$$H1_{final} = \frac{1}{1+e^{-H1}}$$

$$e^{-H1} = \frac{1 - H1_{final}}{H1_{final}} \dots\dots\dots\dots (31)$$

Putting the value of e^{-H1} in **equation (30)**

$$= \frac{1 - H1_{final}}{H1_{final}} \times (H1_{final})^2$$

$$= H1_{final} \times (1 - H1_{final})$$

$$= 0.593269992 \times (1 - 0.593269992)$$

$$\mathbf{\frac{\partial H1_{final}}{\partial H1} = 0.2413007085923199}$$

We calculate the partial derivative of the total net input to H1 with respect to w1 the same as we did for the output neuron:

$$H1 = H1_{final} \times w5 + H2_{final} \times w6 + b2 \dots\dots\dots\dots\dots (32)$$

$$\frac{\partial y1}{\partial w1} = \frac{\partial(x1 \times w1 + x2 \times w3 + b1 \times 1)}{\partial w1}$$

$$= x1$$

$$\mathbf{\frac{\partial H1}{\partial w1} = 0.05} \dots\dots\dots (33)$$

So, we put the values of $\frac{\partial E_{total}}{\partial H1_{final}}$, $\frac{\partial H1_{final}}{\partial H1}$, and $\frac{\partial H1}{\partial w1}$ in **equation (13)** to find the final result.

$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial H1_{final}} \times \frac{\partial H1_{final}}{\partial H1} \times \frac{\partial H1}{\partial w1}$$

$$= 0.0364908241736793 \times 0.2413007085923199 \times 0.05$$

$$\mathbf{Error_{w1} = \frac{\partial E_{total}}{\partial w1} = 0.000438568 \ldots\ldots\ldots (34)}$$

Now, we will calculate the updated weight $w1_{new}$ with the help of the following formula

$$w1_{new} = w1 - \eta \times \frac{\partial E_{total}}{\partial w1} \text{ Here } \eta = \text{learning rate} = 0.5$$

$$= 0.15 - 0.5 \times 0.000438568$$

$$\mathbf{w1_{new} = 0.149780716 \ldots\ldots\ldots (35)}$$

In the same way, we calculate $w2_{new}$,$w3_{new}$, and w4 and this will give us the following values

$w1_{new}$=0.149780716

$w2_{new}$=0.19956143

$w3_{new}$=0.24975114

$w4_{new}$=0.29950229

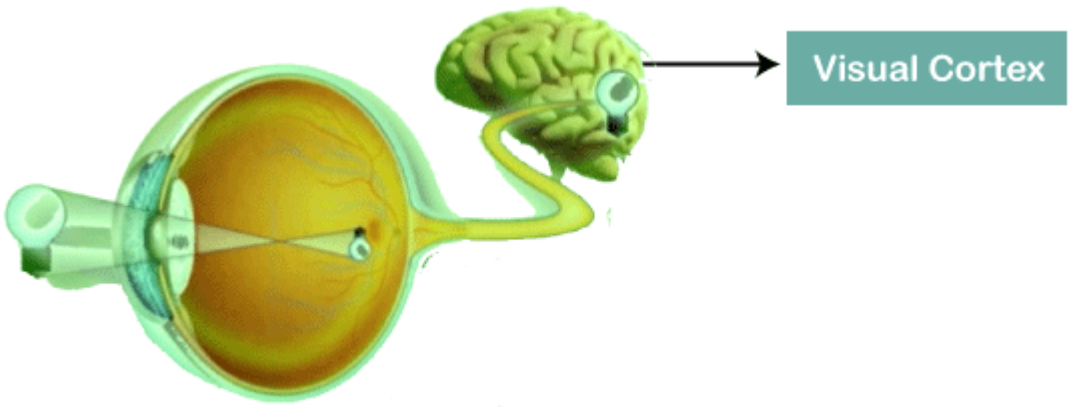
We have updated all the weights. We found the error 0.298371109 on the network when we fed forward the 0.05 and 0.1 inputs.

In the first round of back propagation, the total error is down to 0.291027924. After repeating this process 10,000, the total error is down to 0.0000351085.

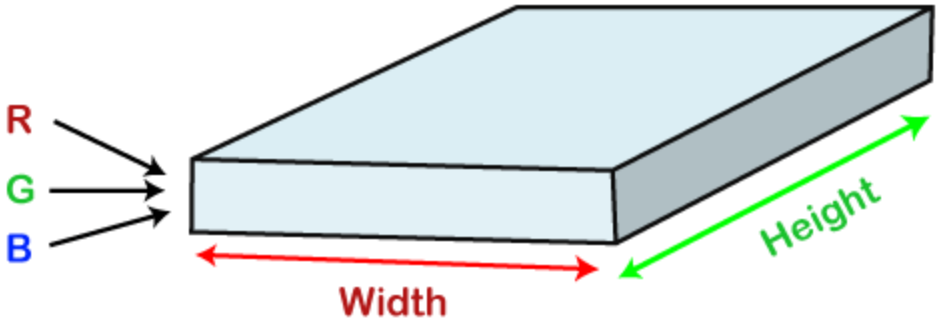
At this point, the outputs neurons generate 0.159121960 and 0.984065734 i.e., nearby our target value when we feed forward the 0.05 and 0.1.

Convolutional Neural Network

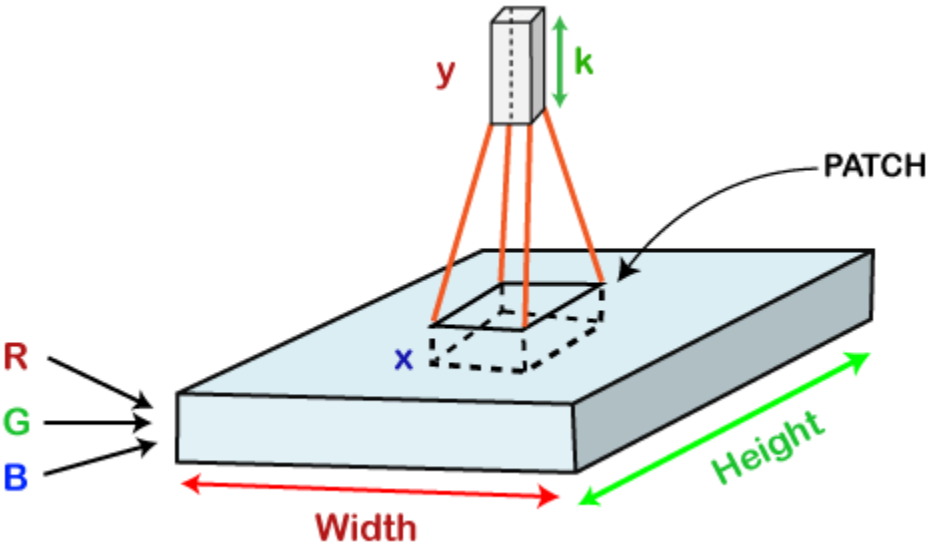
- Convolutional Neural Networks are a special type of feed-forward artificial neural network in which the connectivity pattern between its neuron is inspired by the visual cortex.



- The visual cortex encompasses a small region of cells that are region sensitive to visual fields. In case some certain orientation edges are present then only some individual neuronal cells get fired inside the brain such as some neurons responds as and when they get exposed to the vertical edges, however some responds when they are shown to horizontal or diagonal edges, which is nothing but the motivation behind Convolutional Neural Networks.
- The Convolutional Neural Networks, which are also called as covnets, are nothing but neural networks, sharing their parameters.
- Suppose that there is an image, which is embodied as a cuboid, such that it encompasses length, width, and height.
- Here the dimensions of the image are represented by the Red, Green, and Blue channels, as shown in the image given below.



- Now assume that we have taken a small patch of the same image, followed by running a small neural network on it, having k number of outputs, which is represented in a vertical manner.
- Now when we slide our small neural network all over the image, it will result in another image constituting different width, height as well as depth.
- We will notice that rather than having R, G, B channels, we have come across some more channels that, too, with less width and height, which is actually the concept of Convolution.
- In case, if we accomplished in having similar patch size as that of the image, then it would have been a regular neural network. We have some wights due to this small patch.



Mathematically it could be understood as follows;

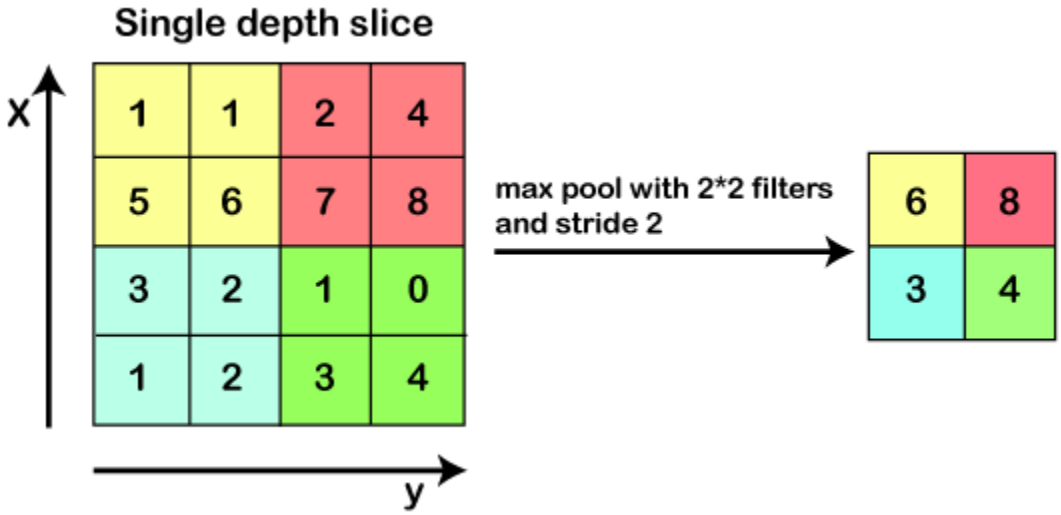
- The Convolutional layers encompass a set of learnable filters, such that each filter embraces small width, height as well as depth as that of the provided input volume (if the image is the input layer then probably it would be 3).
- Suppose that we want to run the convolution over the image that comprises of 34x34x3 dimension, such that the size of a filter can be axax3. Here a can be any of the above 3, 5, 7, etc. It must be small in comparison to the dimension of the image.
- Each filter gets slide all over the input volume during the forward pass. It slides step by step, calling each individual step as a stride that encompasses a value of 2 or 3 or 4 for higher-dimensional images, followed by calculating a dot product in between filter's weights and patch from input volume.
- It will result in 2-Dimensional output for each filter as and when we slide our filters followed by stacking them together so as to achieve an output volume to have a similar depth value as that of the number of filters. And then, the network will learn all the filters.

Working of CNN

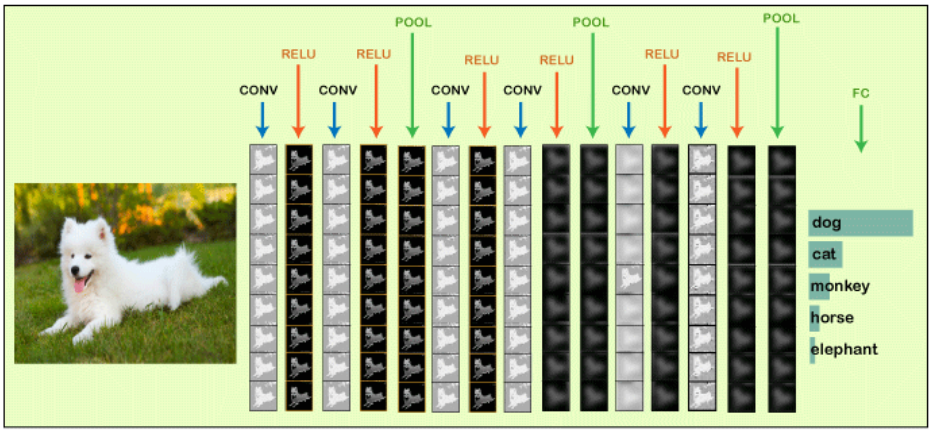
Generally, a Convolutional Neural Network has three layers, which are as follows;

- Input: If the image consists of 32 widths, 32 height encompassing three R, G, B channels, then it will hold the raw pixel([32x32x3]) values of an image.
- Convolution: It computes the output of those neurons, which are associated with input's local regions, such that each neuron will calculate a dot product in between weights and a small region to which they are actually linked to in the input volume. For example, if we choose to incorporate 12 filters, then it will result in a volume of [32x32x12].
- ReLU Layer: It is specially used to apply an activation function elementwise, like as max (0, x) thresholding at zero. It results in ([32x32x12]), which relates to an unchanged size of the volume.

- Pooling: This layer is used to perform a downsampling operation along the spatial dimensions (width, height) that results in [16x16x12] volume.

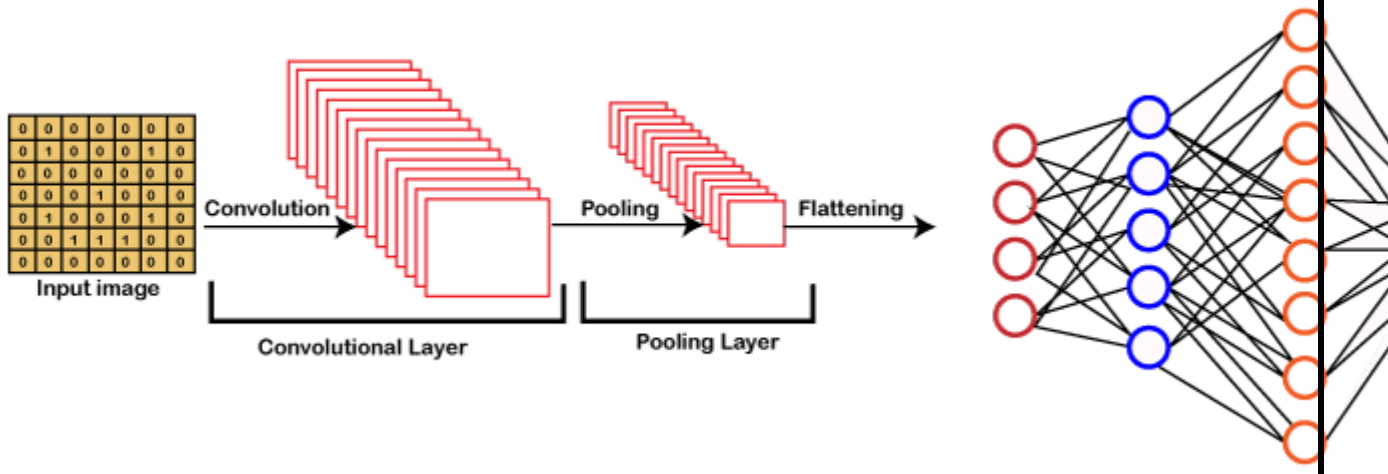


- Locally Connected: It can be defined as a regular neural network layer that receives an input from the preceding layer followed by computing the class scores and results in a 1-Dimensional array that has the equal size to that of the number of classes.



- We will start with an input image to which we will be applying multiple feature detectors, which are also called as filters to create the feature maps that comprises of a Convolution layer.
- Then on the top of that layer, we will be applying the ReLU or Rectified Linear Unit to remove any linearity or increase non-linearity in our images.
- Next, we will apply a Pooling layer to our Convolutional layer, so that from every feature map we create a Pooled feature map as the main purpose of the pooling layer is to make sure that we have spatial invariance in our images.
- It also helps to reduce the size of our images as well as avoid any kind of overfitting of our data.

- After that, we will flatten all of our pooled images into one long vector or column of all of these values, followed by inputting these values into our artificial neural network.
- Lastly, we will feed it into the locally connected layer to achieve the final output.



Building a CNN

- Basically, a Convolutional Neural Network consists of adding an extra layer, which is called convolutional that gives an eye to the Artificial Intelligence or Deep Learning model because with the help of it we can easily take a 3D frame or image as an input as opposed to our previous artificial neural network that could only take an input vector containing some features as information.
- But here we are going to add at the front a convolutional layer which will be able to visualize images just like humans do.
- In our dataset, we have all the images of cats and dogs in training as well as in the test set folders.
- We are going to train our CNN model on 4000 images of cats as well as 4000 images of dogs, each respectively that are present in the training set followed by evaluating our model with the new 1000 images of cats and 1000 images of dogs, each respectively in the test set on which our model was not trained.
- So, we are actually going to build and train a Convolutional Neural network to recognize if there is a dog or cat in the image.
- For the implementation of CNN, we are going to use the Jupyter notebook.
- So, we will start with importing the libraries, data preprocessing followed by building a CNN, training the CNN and lastly, we will make a single prediction.

- All the steps will be carried out in the same way as we did in ANN, the only difference is that now we are not pre-processing the classic dataset, but some images, which is why the data preprocessing is different and will consist of doing two steps, i.e., in the first, we will pre-process the training set and then will pre-process the test set.
- In the second part, we will build the whole architecture of CNN. We will initialize the CNN as a sequence of layers, and then we will add the convolution layer followed by adding the max-pooling layer.
- Then we will add the second convolutional layer to make it a deep neural network as opposed to a shallow neural network.
- Next, we will proceed to the flattening layer to flatten the result of all the convolutions and pooling into a one-dimensional vector, which will become the input of a fully connected neural network. Finally, we will connect all this to the output layer.
- In the third part, we will first compile the CNN, and then we will train the CNN on the training set.
- And then, finally, we will make a single prediction to test our model in a prediction that is when we will deploy our CNN on to different images, one that has a dog and the other that has a cat.
- So, this was just a brief description of how we will build our CNN model, let's get started with its practical implementation.
- We will start by importing the TensorFlow library and actually the preprocessing module by Keras library. And then, we will import the image sub-module of the preprocessing module of the Keras library, which will allow us to do image pre-processing in part 1.

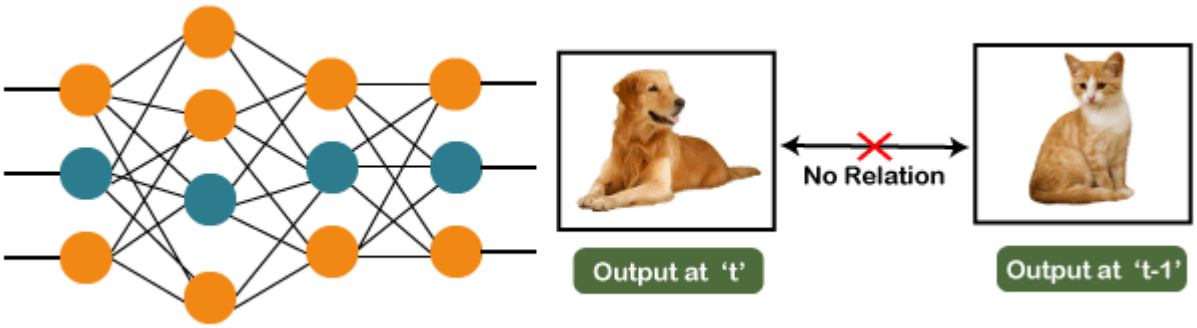
Recurrent Neural Networks

Why not Feedforward Networks?

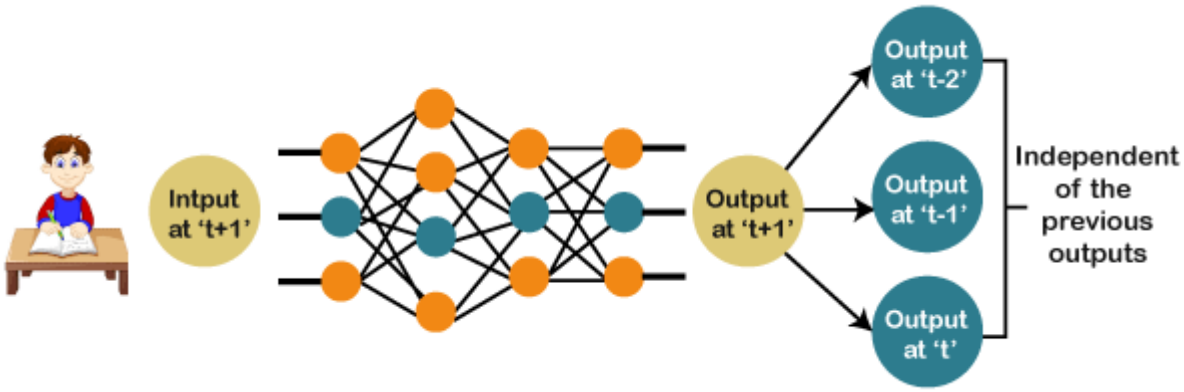
- Feedforward networks are used to classify images.
- Let us understand the concept of a feedforward network with an example given below in which we trained our network for classifying various images of animals.
- If we feed an image of a cat, it will identify that image and provide a relevant label to that particular image.

- Similarly, if you feed an image of a dog, it will provide a relevant label to that image a particular image as well.

Consider the following diagram:



- And if you notice the new output that we have got is classifying, a dog has no relation to the previous output that is of a cat, or you can say that the output at the time 't' is independent of output at a time 't-1'.
- It can be clearly seen that there is no relation between the new output and the previous output. So, we can say that in feedforward networks, the outputs are independent of each other.
- There are a few scenarios where we will actually need the previous output to get the new output. Let us discuss one such scenario where we will necessitate using the output that has been previously obtained.

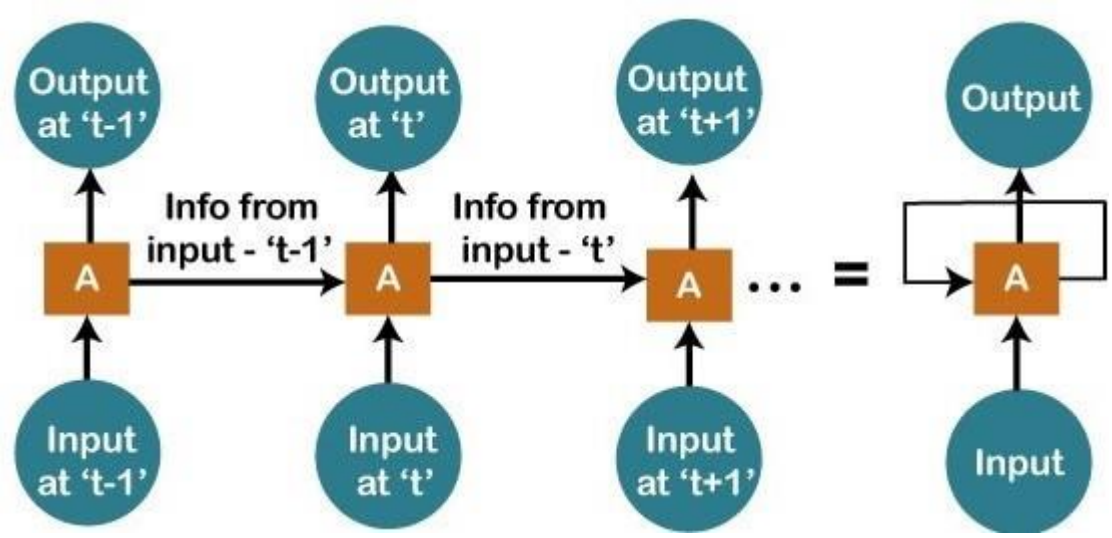


Now, what happens when you read a book.

You will understand that book only on the understanding of the previous words. So, if we use a feedforward network and try to predict the next word in the sentence, then in such a case, we will not be able to do that because our output will actually depend on previous outputs. But in the feedforward network, the new output is independent of the previous outputs, i.e., output at 't+1' has no relation with the output at 't-2', 't-1', and 't.' Therefore, it can be concluded that we cannot use feedforward networks for predicting the next word in the sentence. Similarly, many other examples can also be taken where we need the previous output or some information from the previous output, so as to infer the new output.

How to overcome this challenge?

Consider the following diagram:

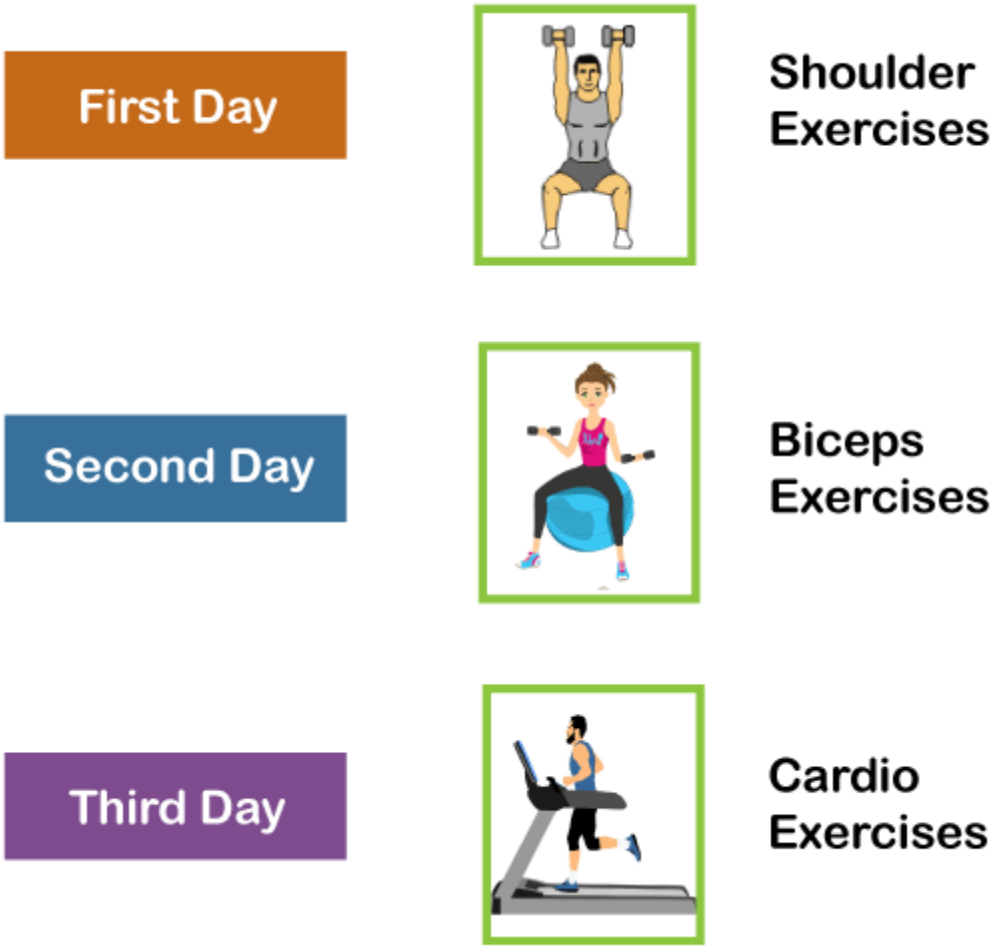


We have input at 't-1', which we will feed to the network, and then we will get the output at 't-1'. Then at the next timestamp that is at a time 't', we have an input at a time 't', which will be again given to the network along with the information from the previous timestamp, i.e., 't-1' and that will further help us to get the output at 't'. Similarly, at the output for 't+1', we have two inputs; one is the new input that we give, and the other is the information coming from the previous timestamps, i.e., 't' in order to get the output at a time 't+1'. In the same way, it will go on further like this. Here we have embodied in a more generalized way to represent it. There is a loop where the information from the previous timestamp is flowing, and this is how we can solve a particular challenge.

What are Recurrent Neural Networks?

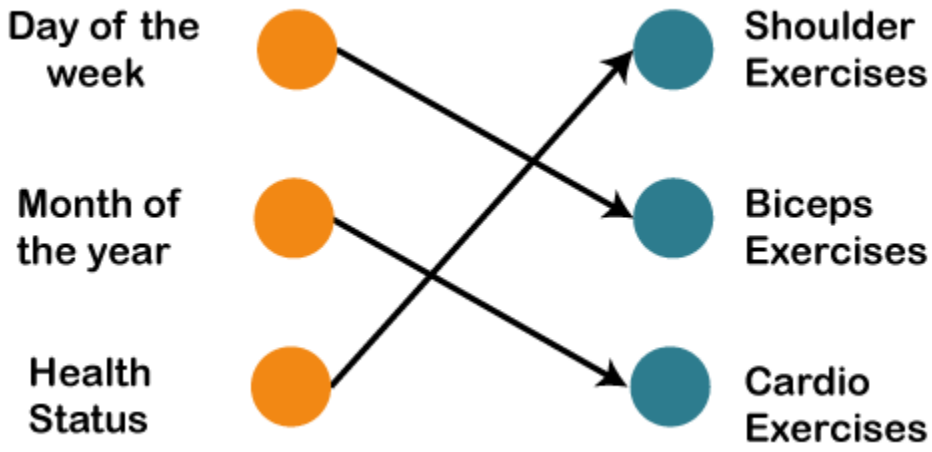
"Recurrent Networks are one such kind of artificial neural network that are mainly intended to identify patterns in data sequences, such as text, genomes, handwriting, the spoken word, numerical times series data emanating from sensors, stock markets, and government agencies".

In order to understand the concept of Recurrent Neural Networks, let's consider the following analogy.

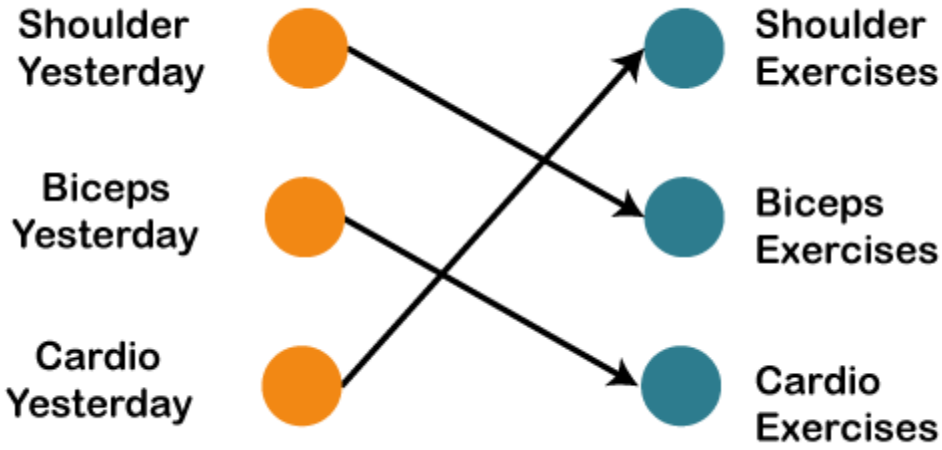


Suppose that your gym trainer has made a schedule for you. The exercises are repeated every third day. The above image includes the order of your exercises; on your very first day, you will be doing shoulders, the second day you will be doing biceps, the third day you will be doing cardio, and all these exercises are repeated in proper order.

Let's see what happens if we use a feedforward network for predicting the exercises today.

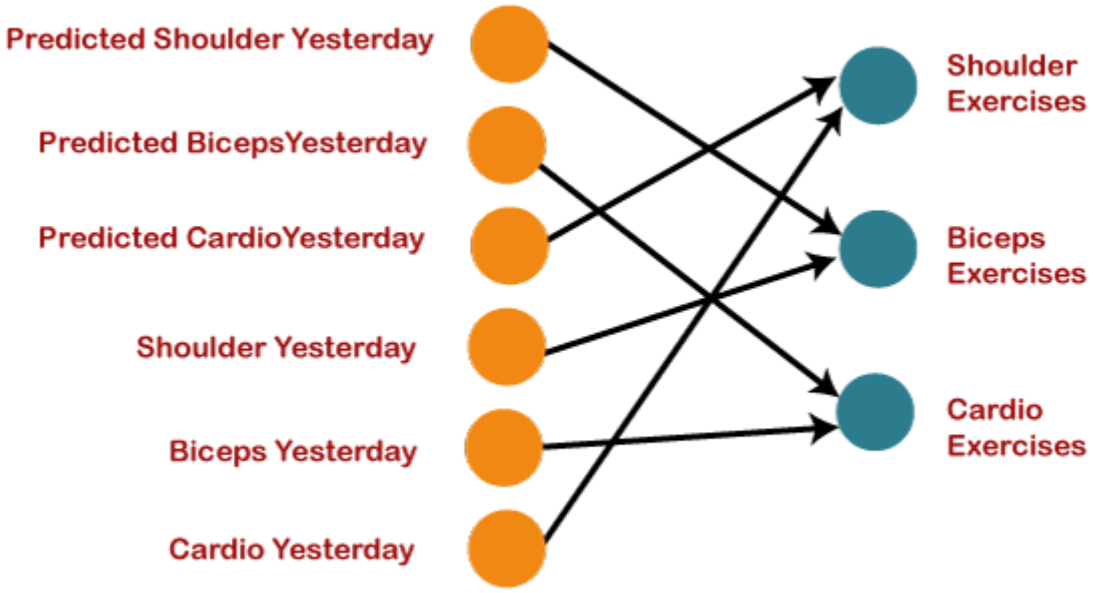


We have provided in the input such as day of the week, the month of the year, and health status. Also, we need to train our model or the network on the basis of the exercises that we have done in the past. After that, there will be a complex voting procedure involved, which will predict the exercises for us, and that procedure won't be that accurate. In that case, whatever output we will get would be as accurate as we want it to be. Now, what if the inputs get changed, and we make the inputs as the exercises that we have done the previous day.



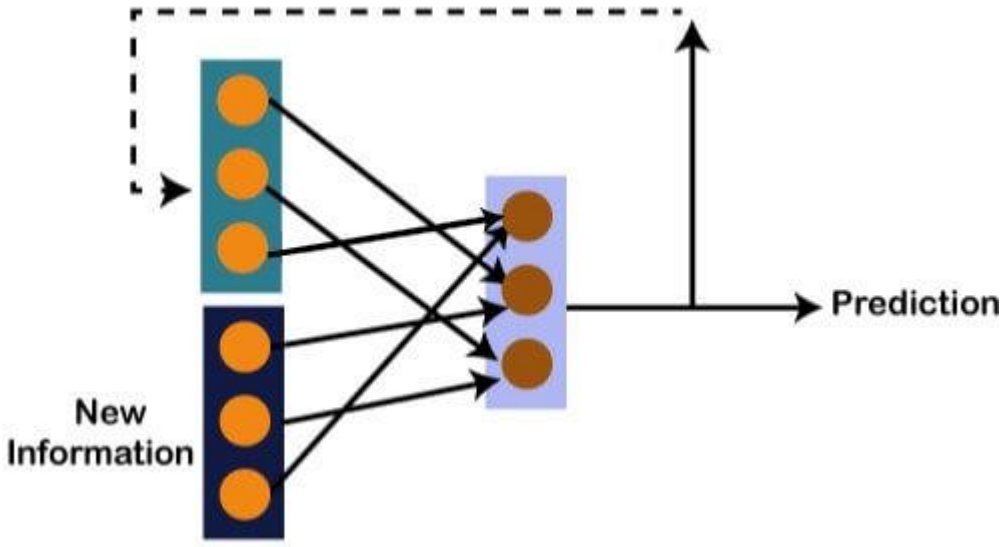
Therefore, if shoulders were done yesterday, then definitely today will be biceps day. Similarly, if biceps were done yesterday, then today will be the cardio day, and if yesterday was the cardio day, then today, we will need to undergo shoulder.

Now there can be one such scenario, where you are unable to go to the gym for one day due to some personal reasons, then it that case, we will go one timestamp back and will feed in what exercise happened day before yesterday as shown below.



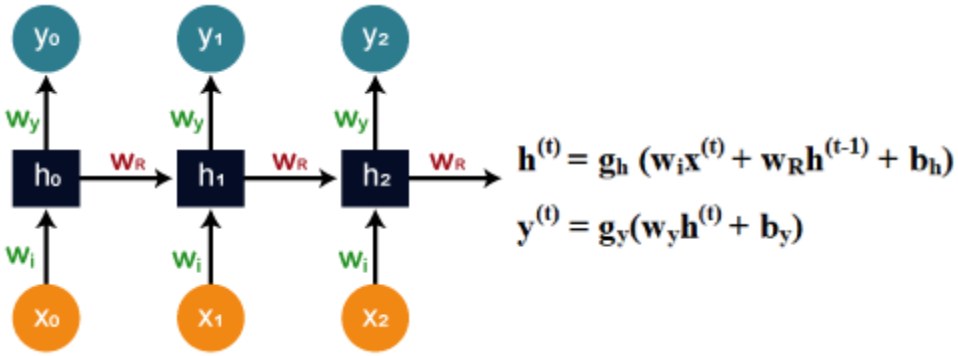
So, if the exercise that happened the day before yesterday was the shoulder, then yesterday there were biceps exercises. Similarly, if biceps happened the day before yesterday, then yesterday would have been cardio exercises, and if cardio would have happened the day before yesterday, then yesterday would have been shoulder exercises. And this prediction for the exercises that happened yesterday will be fed back to our network so that these predictions can be used as inputs in order to predict what exercise will happen today. Similarly, if you have missed your gym say for two days, three days or even one week, you will actually need to roll back, which means that you will need to go to the last day when you went to the gym, you need to figure out what exercises you did on that day and then only you will be getting the relevant output as to what exercises will happen today.

Next, we will convert all these things into a vector, which is nothing but a list of numbers.



So, there is new information along with the information which we got from the prediction at the previous timestamp because we need all of these in order to get the prediction at a time 't'. Imagine that you did shoulder exercises yesterday, then, in that case, the prediction will be biceps exercise because if the shoulder was done yesterday, then today it will definitely be biceps and output will be 0, 1, and 0, which is actually the work of our vectors.

Let's understand the math behind the Recurrent Neural Network by simply having a look at the image given below.



Assume that 'w' is the weight matrix, and 'b' is the bias. Consider at time t=0, our input is 'x_o', and we need to figure out what exactly is the 'h_o'. We will substitute t=0 in the equation, as shown in the image, so as to procure the function h^t value.

After that, we will find out the value of 'y_o' by using values that were previously calculated when we applied it to the new formula.

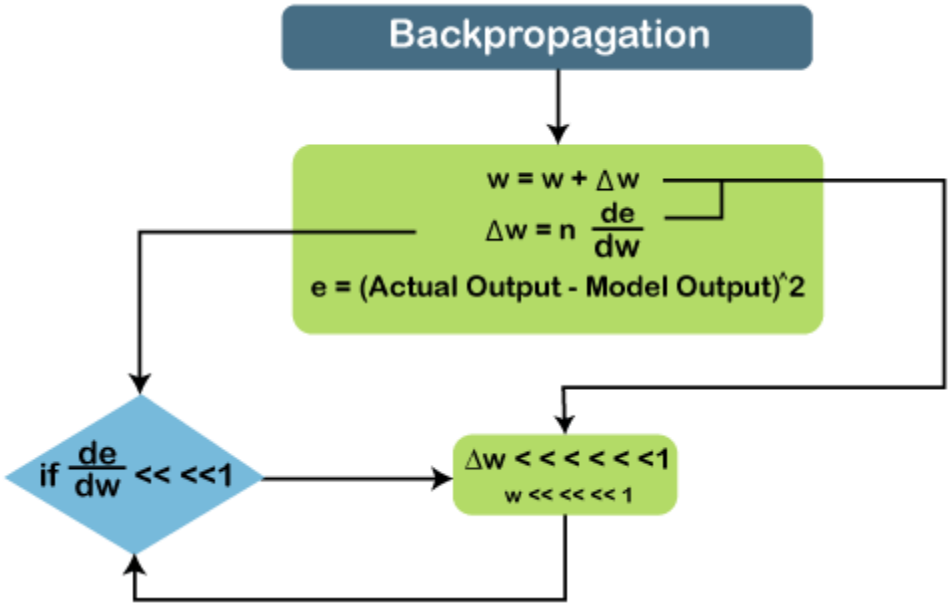
The same process is repeated again and again through all the timestamps within the model so as to train it. So, this how a Recurrent Neural Networks works.

Training a Recurrent Neural Network

A recurrent neural network uses a backpropagation algorithm for training, but backpropagation happens for every timestamp, which is why it is commonly called as backpropagation through time. With backpropagations, there are certain issues, namely vanishing and exploding gradients, that we will see one by one.

Vanishing Gradient

Consider the following diagram:



In vanishing gradient when we use backpropagation, we tend to calculate the error which is nothing but the actual output that we already know subtracted by the model output that we got through the model and the square of that, so we can figure out the error, and with the help of that error, we tend to find out the change in error with respect to change in weight or any variable, which is here called as weight.

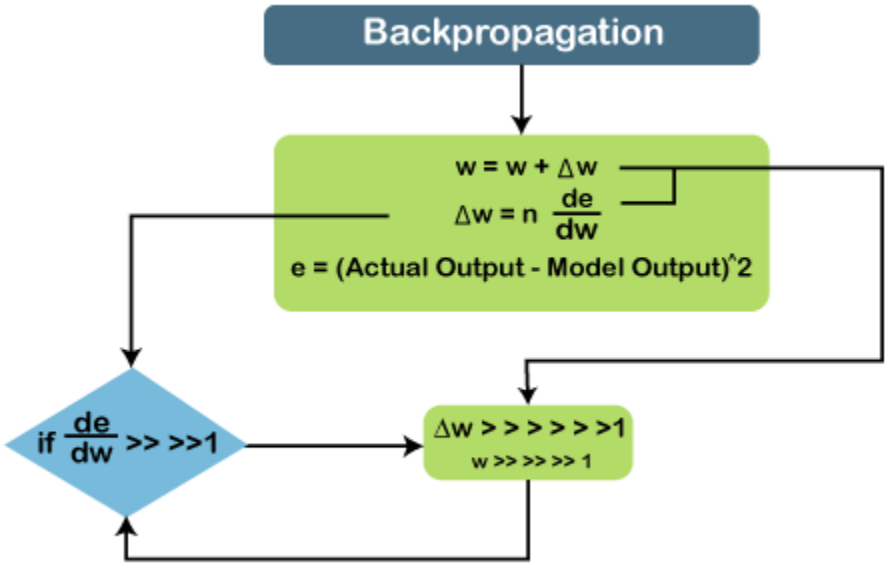
So, the change of error with respect to change in weight multiplied by learning rate will give us the change in rate. And then we will add this change in weight to the old weight to get a new

weight. Basically, here we are trying to reduce the error, and for that, we need to figure out what will be the change in error if variables get changed, by which we can get the change in the variable and add it to our old variable to get the new variable.

Now over here what can happen if the value de⁄dw, i.e., gradient or simply we can say the rate of change of error with respect to weight variable becomes very smaller than 1 and if we multiply that with the learning rate, which is the definitely smaller than 1, then, in that case, we will get the change in weight, which is negligible.

Consider a scenario where you need to predict the next word in the sentence, and your sentence is something like "I have been to USA". Then are a lot of words after that few people speak, and then you need to predict what comes after speak. Now, if you need to do that, then you will have to go back and understand the context of what it is talking about, which is nothing but our long-term dependencies. During the long-term dependencies, de⁄dw becomes very small, and then when you multiply it with n, which is again smaller than 1, you get Δw, which will be very small or simply negligible. So, the new weight that you will get here will be almost equal to your old weight, such that the weight will not get updated further. Also, there will be no learning here, which is nothing but the problem of vanishing gradient.

Similarly, if we talk about the exploding gradient, it is actually opposite to that of the vanishing gradient. Consider the below diagram to have a better understanding of it.



If de⁄dw becomes very large or greater than 1 and we have some long-term dependencies, then, in that case, de⁄dw will keep on increasing, and Δw will become very large that will make the new weight different than that of the old weights. So, these were the two problems with backpropagation, and now we will see how to solve these problems.

Tensor Flow Tutorial

- TensorFlow tutorial is designed for both beginners and professionals.
- Our tutorial provides all the basic and advanced concept of machine learning and deep learning concept such as deep neural network, image processing and sentiment analysis.
- ensorFlow is one of the famous deep learning framework, developed by Google Team.
- It is a free and open source software library and designed in Python programming language, this tutorial is designed in such a way that we can easily implement deep learning project on TensorFlow in an easy and efficient way.

Prerequisite

TensorFlow is completely based on Python. So, it is essential to have basic knowledge of Python. Good understanding of basic mathematics and artificial intelligence concept is allow us to understand TensorFlow easily.

Audience

This tutorial is helpful for the students who are interested in python and focused on research and development with many machine learning and deep learning algorithms. The aim of the tutorial is to describe all TensorFlow objects and methods.

Problems

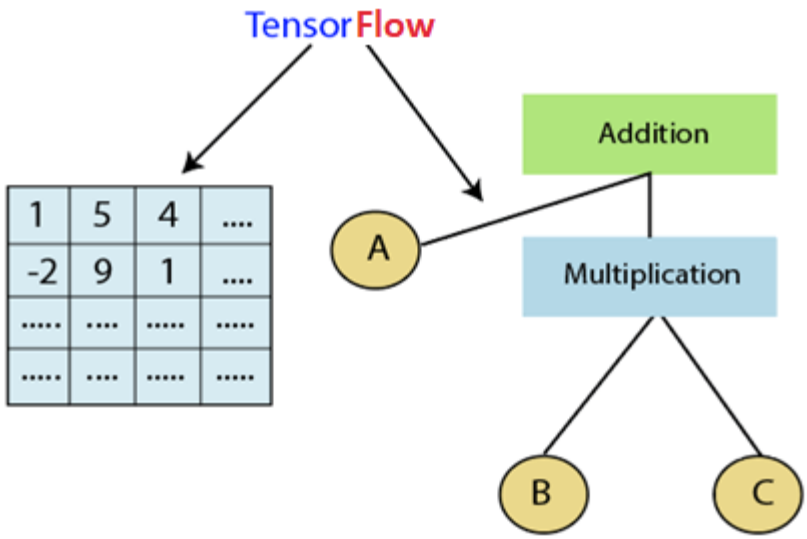
We assure that we will not find any problem with this TensorFlow tutorial. But if there is any mistake, please post the problem in the contact form.

TensorFlow can train and run the deep neural networks for image recognition, handwritten digit classification, recurrent neural network, word embedding, natural language processing, video detection, and many more. TensorFlow is run on multiple CPUs or GPUs and also mobile operating systems.

The word TensorFlow is made by two words, i.e., Tensor and Flow

1. Tensor is a multidimensional array
2. Flow is used to define the flow of data in operation.

TensorFlow is used to define the flow of data in operation on a multidimensional array or Tensor.



History of TensorFlow

Many years ago, deep learning started to exceed all other machine learning algorithms when giving extensive data. Google has seen it could use these deep neural networks to upgrade its services:

- Google search engine
- Gmail
- Photo

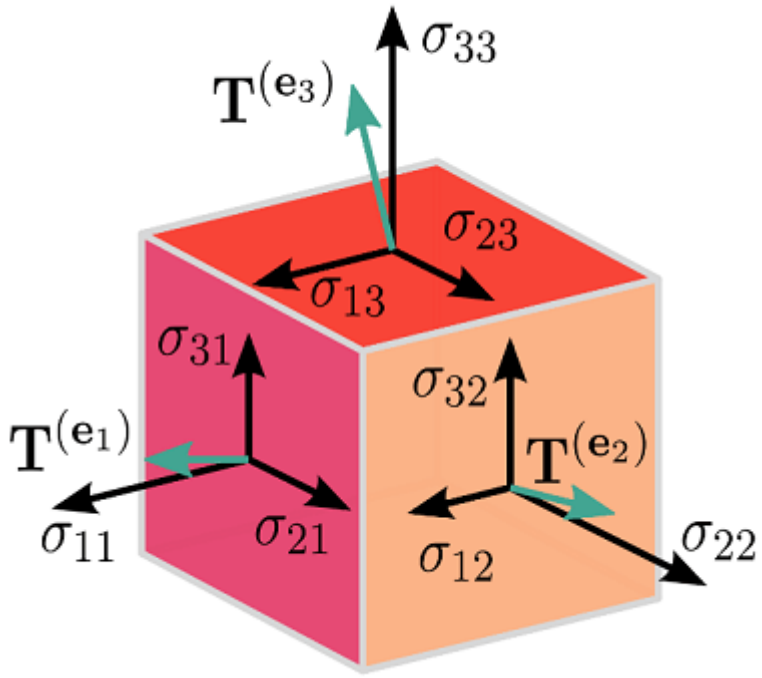
They build a framework called TensorFlow to permit researchers and developers to work together in an AI model. Once it approved and scaled, it allows lots of people to use it.

It was first released in 2015, while the first stable version was coming in 2017. It is an open-source platform under Apache Open Source License. We can use it, modify it, and reorganize the revised version for free without paying anything to Google.

Components of TensorFlow

Tensor

- The name TensorFlow is derived from its core framework, "Tensor."
- A tensor is a vector or a matrix of n-dimensional that represents all type of data.
- All values in a tensor hold similar data type with a known shape. The shape of the data is the dimension of the matrix or an array.
- A tensor can be generated from the input data or the result of a computation. In TensorFlow, all operations are conducted inside a graph.
- The group is a set of calculation that takes place successively. Each transaction is called an op node are connected.



Graphs

TensorFlow makes use of a graph framework. The chart gathers and describes all the computations done during the training.

Advantages

- It was fixed to run on multiple CPUs or GPUs and mobile operating systems.
- The portability of the graph allows to conserve the computations for current or later use. The graph can be saved because it can be executed in the future.
- All the computation in the graph is done by connecting tensors together.

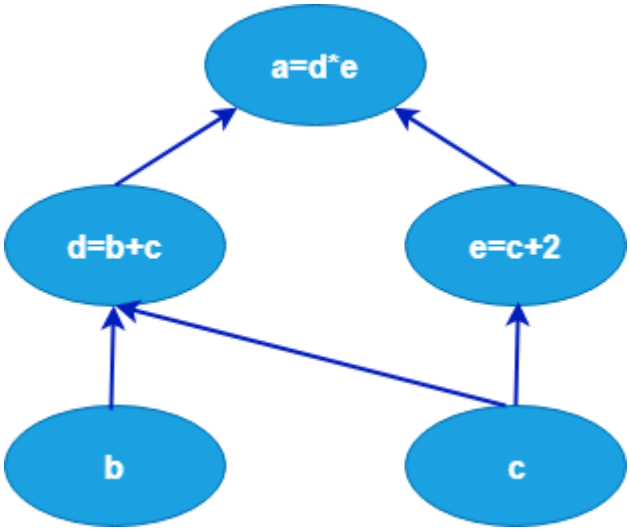
Consider the following expression a= (b+c)*(c+2)

d=b+c

e=c+2

a=d*e

Now, we can represent these operations graphically below:



Session

A session can execute the operation from the graph. To feed the graph with the value of a tensor, we need to open a session. Inside a session, we must run an operator to create an output.

Why is TensorFlow popular?

- TensorFlow is the better library for all because it is accessible to everyone. TensorFlow library integrates different API to create a scale deep learning architecture like CNN (Convolutional Neural Network) or RNN (Recurrent Neural Network).
- TensorFlow is based on graph computation; it can allow the developer to create the construction of the neural network with Tensorboard. This tool helps debug our program. It runs on CPU (Central Processing Unit) and GPU (Graphical Processing Unit).

Keras Tutorial

- Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK.
- It was developed by one of the Google engineers, Francois Chollet.
- It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks.
- It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.
- It cannot handle low-level computations, so it makes use of the Backend library to resolve it.
- The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.
- Initially, it had over 4800 contributors during its launch, which now has gone up to 250,000 developers.
- It has a 2X growth ever since every year it has grown.
- Big companies like Microsoft, Google, NVIDIA, and Amazon have actively contributed to the development of Keras.
- It has an amazing industry interaction, and it is used in the development of popular firms likes Netflix, Uber, Google, Expedia, etc.

What makes Keras special?

- Focus on user experience has always been a major part of Keras.

- Large adoption in the industry.
- It is a multi backend and supports multi-platform, which helps all the encoders come together for coding.
- Research community present for Keras works amazingly with the production community.
- Easy to grasp all concepts.
- It supports fast prototyping.
- It seamlessly runs on CPU as well as GPU.
- It provides the freedom to design any architecture, which then later is utilized as an API for the project.
- It is really very simple to get started with.
- Easy production of models actually makes Keras special.

Keras user experience

1. Keras is an API designed for humans
   Best practices are followed by Keras to decrease cognitive load, ensures that the models are consistent, and the corresponding APIs are simple.
2. Not designed for machines
   Keras provides clear feedback upon the occurrence of any error that minimizes the number of user actions for the majority of the common use cases.
3. Easy to learn and use.
4. Highly Flexible
   Keras provide high flexibility to all of its developers by integrating low-level deep learning languages such as TensorFlow or Theano, which ensures that anything written in the base language can be implemented in Keras.

How Keras support the claim of being multi-backend and multi-platform:

Keras can be developed in R as well as Python, such that the code can be run with TensorFlow, Theano, CNTK, or MXNet as per the requirement. Keras can be run on CPU, NVIDIA GPU, AMD GPU, TPU, etc. It ensures that producing models with Keras is really simple as it totally

supports to run with TensorFlow serving, GPU acceleration (WebKeras, Keras.js), Android (TF, TF Lite), iOS (Native CoreML) and Raspberry Pi.

Keras Backend:

- Keras being a model-level library helps in developing deep learning models by offering high-level building blocks.
- ll the low-level computations such as products of Tensor, convolutions, etc. are not handled by Keras itself, rather they depend on a specialized tensor manipulation library that is well optimized to serve as a backend engine.
- Keras has managed it so perfectly that instead of incorporating one single library of tensor and performing operations related to that particular library, it offers plugging of different backend engines into Keras.

Keras consist of three backend engines, which are as follows:

- ### TensorFlow:

  TensorFlow is a Google product, which is one of the most famous deep learning tools widely used in the research area of machine learning and deep neural network. It came into the market on 9th November 2015 under the Apache License 2.0. It is built in such a way that it can easily run on multiple CPUs and GPUs as well as on mobile operating systems. It consists of various wrappers in distinct languages such as Java, C++, or Python.

- ### Theano:

  Theano was developed at the University of Montreal, Quebec, Canada, by the MILA group. It is an open-source python library that is widely used for performing mathematical operations on multi-dimensional arrays by incorporating scipy and numpy. It utilizes GPUs for faster computation and efficiently computes the gradients

by building symbolic graphs automatically. It has come out to be very suitable for unstable expressions, as it first observes them numerically and then computes them with more stable algorithms.

- **CNTK**

  Microsoft Cognitive Toolkit is deep learning's open-source framework. It consists of all the basic building blocks, which are required to form a neural network. The models are trained using C++ or Python, but it incorporates C# or Java to load the model for making predictions.

Advantages of Keras

Keras encompasses the following advantages, which are as follows:

- It is very easy to understand and incorporate the faster deployment of network models.
- It has huge community support in the market as most of the AI companies are keen on using it.
- It supports multi backend, which means you can use any one of them among TensorFlow, CNTK, and Theano with Keras as a backend according to your requirement.
- Since it has an easy deployment, it also holds support for cross-platform. Following are the devices on which Keras can be deployed:
    1. iOS with CoreML
    2. Android with TensorFlow Android
    3. Web browser with .js support
    4. Cloud engine
    5. Raspberry pi
- It supports Data parallelism, which means Keras can be trained on multiple GPU's at an instance for speeding up the training time and processing a huge amount of data.

Disadvantages of Keras

- The only disadvantage is that Keras has its own pre-configured layers, and if you want to create an abstract layer, it won't let you because it cannot handle low-level APIs. It only supports high-level API running on the top of the backend engine (TensorFlow, Theano, and CNTK).

Prerequisite

This Keras tutorial is made for both beginners and professionals, to help them understand the fundamental concept of Keras. After the completion of this tutorial, you will find yourself at a moderate level of expertise from where you can take yourself to the next level.

Audience

Since Keras is a deep learning's high-level library, so you are required to have hands-on Python language as well as basic knowledge of the neural network.

Problem

We assure you that you will not find any difficulty in this tutorial. In case you come up with a query, or you find any mistake in this tutorial, do let us know by posting it in the contact form so that we can further improve it.