



KURUNJI VENKATRAMANA GOWDA POLYTECHNIC SULLIA-574327

5TH SEMESTER

AI/ML WEEK-10

WEEK – 10:

Limitations of Machine Learning:

The following factors serve to limit it:

1. Data Acquisition

Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

2. Time and Resources

ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

3. Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

4. High error-susceptibility

Machine Learning is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being displayed to customers. In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

5.Data

This is the most obvious limitation. If the model you feed is poor, it can only give you bad results. This can be manifested in two ways: lack of data and lack of **Good** data.

Missing data

Many machine learning algorithms require a lot of data before they begin to provide useful results. A good example is the neural network. A neural network is a data phagocytic machine that requires a lot of training data. The larger the architecture, the more data is needed to produce a viable outcome. Reusing data is a bad idea, and data growth is useful to some extent, but having more data is always the preferred solution.

Lack of good data

Despite its appearance, this is different from the above comments. Let's imagine that you think you can trick your neural network by generating 10,000 fake data points. What happens when you put it in?

It will train itself, and then when it's tested on an invisible data set, it won't work. You have data, but the quality of the data does not meet the standard.

6. Interpretability:

Interpretability is one of the main problems of machine learning. An artificial intelligence consulting firm is trying to invest in a company that only uses traditional statistical methods, and if they think the model is unexplained, they can stop. If you can't convince your customers to understand how algorithms make decisions, how likely are they to trust you and your expertise?

Just like " *Business data mining-machine learning perspective* "In the blunt words:

"If you interpret the results in business terms, business managers are more likely to accept [machine learning methods] advice"

Unless these models can be explained, these models can become powerless, and the process of human interpretation follows rules far beyond the technical strength. Therefore, interpretability is the highest quality that machine learning methods should achieve if they are to be applied in practice.

Deep Learning:

What is Deep Learning?

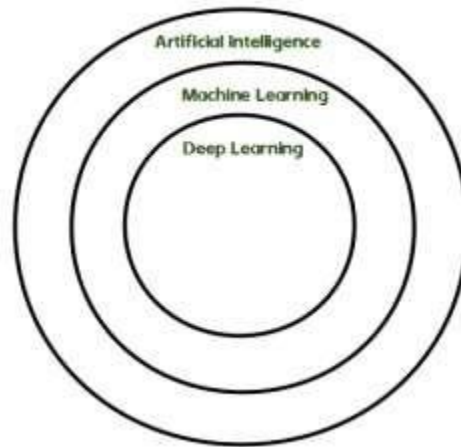
Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It's on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture. A formal definition of deep learning is- neurons.

Deep Learning Architectures:

1. **Deep Neural Network** – It is a neural network with a certain level of complexity (having multiple hidden layers in between input and output layers). They are capable of modeling and processing non-linear relationships.
2. **Deep Belief Network(DBN)** – It is a class of Deep Neural Network. It is multi-layer belief networks.

Steps for performing DBN :

- a. Learn a layer of features from visible units using Contrastive Divergence algorithm.
 - b. Treat activations of previously trained features as visible units and then learn features of features.
 - c. Finally, the whole DBN is trained when the learning for the final hidden layer is achieved.
3. **Recurrent** (perform same task for every element of a sequence) **Neural Network** – Allows for parallel and sequential computation. Similar to the human brain (large feedback network of connected neurons). They are able to remember important things about the input they received and hence enables them to be more precise.

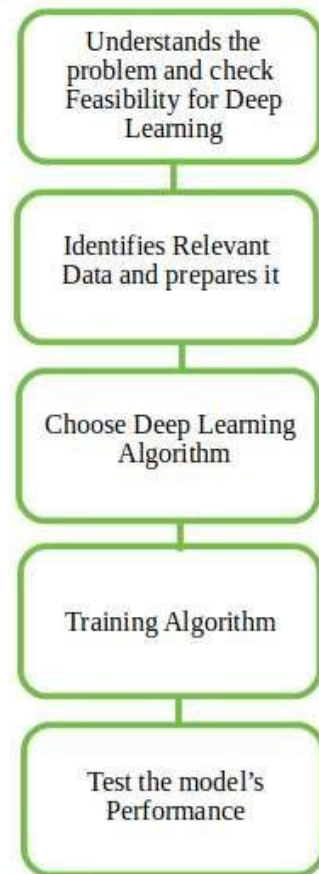


Difference between Machine Learning and Deep Learning:

Machine Learning	Deep Learning
Works on small amount of Dataset for accuracy.	Works on Large amount of Dataset.
Dependent on Low-end Machine.	Heavily dependent on High-end Machine.
Divides the tasks into sub-tasks, solves them individually and finally combine the results.	Solves problem end to end.
Takes less time to train.	Takes longer time to train.
Testing time may increase.	Less time to test the data.

Working:

First, we need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). Second, we need to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Third, Choose the Deep Learning Algorithm appropriately. Fourth, Algorithm should be used while training the dataset. Fifth, Final testing should be done on the dataset.

**Tools used :**

Anaconda, Jupyter, Pycharm, etc.

Languages used :

R, Python, Matlab, CPP, Java, Julia, Lisp, Java Script, etc.

Real Life Examples :

1. How to recognize square from other shapes?
 - ...a) Check the four lines!
 - ...b) Is it a closed figure?
 - ...c) Does the sides are perpendicular from each other?
 - ...d) Does all sides are equal?

So, Deep Learning is a complex task of identifying the shape and broken down into simpler tasks at a larger side

2.Recognizing an Animal! (Is it a Cat or Dog?)

Defining facial features which are important for classification and system will then identify this automatically.

(Whereas Machine Learning will manually give out those features for classification)

Advantages :

1. *Best in-class performance on problems.*
2. *Reduces need for feature engineering.*
3. *Eliminates unnecessary costs.*
4. *Identifies defects easily that are difficult to detect.*

Disadvantages :

1. *Large amount of data required.*
2. *Computationally expensive to train.*
3. *No strong theoretical foundation.*

Applications :

1. **Automatic Text Generation** – Corpus of text is learned and from this model new text is generated, word-by-word or character-by-character. Then this model is capable of learning how to spell, punctuate, form sentences, or it may even capture the style.
2. **Healthcare** – Helps in diagnosing various diseases and treating it.
3. **Automatic Machine Translation** – Certain words, sentences or phrases in one language is transformed into another language (Deep Learning is achieving top results in the areas of text, images).
4. **Image Recognition** – Recognizes and identifies peoples and objects in images as well as to understand content and context. This area is already being used in Gaming, Retail, Tourism, etc.
5. **Predicting Earthquakes** – Teaches a computer to perform viscoelastic computations which are used in predicting earthquakes.

Deep Learning Frameworks

Deep learning (DL) frameworks offer building blocks for designing, training, and validating deep neural networks through a high-level programming interface. Widely-used DL frameworks, such as PyTorch, TensorFlow, [PyTorch Geometric](#), [DGL](#), and others, rely on GPU-accelerated libraries, such as cuDNN, NCCL, and DALI to deliver high-performance, multi-GPU-accelerated training.

1. PyTorch

PyTorch is a Python package that provides two high-level features:

- Tensor computation (like numpy) with strong GPU acceleration.
- Deep Neural Networks (DNNs) built on a tape-based autograd system.

Reuse your favorite Python packages, such as numpy, scipy and Cython, to extend PyTorch when needed.

2. TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. For visualizing TensorFlow results, TensorFlow offers TensorBoard, a suite of visualization tools.

3. JAX

JAX is a Python library designed for high-performance numerical computing and machine learning research. JAX can automatically differentiate native Python and implement the NumPy API. With just a few lines of code change, JAX enables distributed training across multi-node, multi-GPU systems, with accelerated performance through an XLA-optimized kernel on NVIDIA GPUs. Both Python and NumPy are widely used and familiar, making JAX simple, flexible, and easy to adopt.

4. PaddlePaddle

PaddlePaddle provides an intuitive and flexible interface for loading data and specifying model structures. It supports CNN, RNN, and multiple variants, and easily configures complicated deep models.

PaddlePaddle also provides extremely optimized operations, memory recycling, and network communication, and makes it easy to scale heterogeneous computing resources and storage to accelerate the training process.

5. MXNet

MXNet is a DL framework designed for both efficiency and flexibility. It allows you to mix the flavors of symbolic programming and imperative programming to maximize efficiency and productivity.

At its core is a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on-the-fly. A graph optimization layer on top of that makes symbolic execution fast and memory efficient. The library is portable and lightweight, and it scales to multiple GPUs and machines.

Deep Learning Environment setup for windows:

PC Hardware Setup

First of all to perform machine learning and deep learning on any dataset, the software/program requires a computer system powerful enough to handle the computing power necessary. So the following is required:

1. **Central Processing Unit (CPU)** — Intel Core i5 6th Generation processor or higher. An AMD equivalent processor will also be optimal.
2. **RAM** — 8 GB minimum, 16 GB or higher is recommended.
3. **Graphics Processing Unit (GPU)** — NVIDIA GeForce GTX 960 or higher. AMD GPUs are not able to perform deep learning regardless. For more information on NVIDIA GPUs for deep learning please visit <https://developer.nvidia.com/cuda-gpus>.
4. **Operating System** — Ubuntu or Microsoft Windows 10. I recommend updating Windows 10 to the latest version before proceeding forward.

Table of Contents

In this tutorial, we will cover the following steps:

1. Download Anaconda
2. Install Anaconda & Python
3. Start and Update Anaconda
4. Install CUDA Toolkit & cuDNN
5. Create an Anaconda Environment
6. Install Deep Learning API's (TensorFlow & Keras)

Step 1: Download Anaconda

In this step, we will download the Anaconda Python package for your platform.

Anaconda is a free and easy-to-use environment for scientific Python.

- 1. Install Anaconda (Python 3.6 version) [Download](#)



I am using Windows you can choose according to your OS.

Step 2: Install Anaconda

In this step, we will install the Anaconda Python software on your system.

Installation is very easy and quick once you download the setup. Open the setup and follow the wizard instructions.

#Note: It will automatically install Python and some basic libraries with it.

It might take 5 to 10 minutes or some more time according to your system.



Step 3: Update Anaconda

Open Anaconda Prompt to type the following command(s). Don't worry Anaconda Prompt works the same as cmd.

```
conda update conda
conda update --all
```

Step 4: Install CUDA Toolkit & cuDNN

1. Install CUDA Toolkit 9.0 or 8.0 [Download](#)

Choose your version depending on your Operating System and GPU.

#Version Support: Here is a guide to check that if your [version supports](#) your Nvidia Graphic Card

CUDA Toolkit 9.0 Downloads

Select Target Platform ⓘ
Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows	Linux	Mac OSX		
Architecture ⓘ	x86_64				
Version	10	8.1	7	Server 2016	Server 2012 R2
Installer Type ⓘ	exe (network)	exe (local)			

Download Installers for Windows 10 x86_64

The base installer is available for download below.

There is 1 patch available. This patch requires the base installer to be installed first.

➤ Base Installer

Download (1.4 GB) ⬇

Installation Instructions:
1. Double click cuda_9.0.176_win10.exe
2. Follow on-screen prompts

➤ Patch 1 (Released Jan 25, 2018)

Download (54.1 MB) ⬇

cuBLAS Patch Update: This update to CUDA 9.0 includes new GEMM kernels optimized for the Volta architecture and improved heuristics to select GEMM kernels for given input sizes.

#Note: People with version 9.0 [Download](#) can also install the given patch in any case of error while proceeding.

2. Download cuDNN [Download](#)

Download the latest version of cuDNN. Choose your version depending on your Operating System and CUDA. Membership registration is required. Don't worry you can easily create an account using your email.

cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural networks.

☒ I Agree To the Terms of the [cuDNN Software License Agreement](#)

Note: Please refer to the [Installation Guide](#) for release prerequisites, including supported GPU architectures and compute capabilities, before downloading.

For more information, refer to the cuDNN Developer Guide, Installation Guide and Release Notes on the [Deep Learning SDK Documentation](#) web page.

[Download cuDNN v7.0.5 \(Dec 11, 2017\), for CUDA 9.1](#)

[Download cuDNN v7.0.5 \(Dec 5, 2017\) for CUDA 9.0](#)

[cuDNN Developer Guide](#)

[cuDNN Install Guide](#)

[cuDNN Release Notes](#)

[cuDNN v7.0.5 Library for Linux](#)

[cuDNN v7.0.5 Library for Linux \(Power8\)](#)

[cuDNN v7.0.5 Library for Windows 7](#)

[cuDNN v7.0.5 Library for Windows 10](#)

[cuDNN v7.0.5 Runtime Library for Ubuntu16.04 \(Deb\)](#)

[cuDNN v7.0.5 Developer Library for Ubuntu16.04 \(Deb\)](#)

Put your unzipped folder in C drive as follows:

C:\cudnn-9.0-windows10-x64-v7

Step 5: Add cuDNN into Environment Path

1. Open Run dialogue using (**Win + R**) and run the command **sysdm.cpl**
2. In Window-10 **System Properties**, please select the Tab **Advanced**.
3. Select Environment Variables
4. Add the following path to your Environment.

C:\cudnn-9.0-windows10-x64-v7\cuda\bin

Step 6: Create an Anaconda Environment

Here we will create a new anaconda environment for our specific usage so that it will not affect the root of Anaconda. Amazing!! isn't it? 🍕

Open Anaconda Prompt to type the following commands.

1. Create a conda environment named “tensorflow” (you can change the name) by invoking the following command:

conda create -n tensorflow pip python=3.6

2. Activate the conda environment by issuing the following command:

activate tensorflow

(tensorflow)C:> # Your prompt should change

Step 7: Install Deep Learning Libraries

In this step, we will install Python libraries used for deep learning, specifically: TensorFlow, and Keras.

1. TensorFlow

TensorFlow is a tool for machine learning. While it contains a wide range of functionality, TensorFlow is mainly designed for deep neural network models.

=> For installing TensorFlow, Open Anaconda Prompt to type the following commands.

To install the GPU version of TensorFlow:

C:\> **pip install tensorflow-gpu**

To install the CPU-only version of TensorFlow:

C:\> **pip install tensorflow**

If your machine or system is the only CPU supported you can install CPU version for basic learning and practice.

=> You can test the installation by running this program on shell:

```
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```

For getting started and documentation you can visit [TensorFlow](#) website.

2. Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of [TensorFlow](#), [CNTK](#), or [Theano](#).

=> For installing Keras Open Anaconda Prompt to type the following commands.

pip install keras

=> Let's try running [Mnist Mlp.Py](#) in your prompt. you can use other [examples](#) as well.

Open Anaconda Prompt to type the following commands.

activate tensorflow

python mnist_mlp.py

For getting started and documentation you can visit [Keras](#) website.

Deep Learning Cloud Environment:

Minimal Configuration Cloud-based Deep Learning Environments

Using a pre-configured cloud-based deep learning environment is the best. Typically, there are several cloud-based service providers for deep learning. The following options enable you to start working right away, with minimal setup and configuration. Do note this is my no means a comprehensive list but options I have experimented with or heard from fellow deep learning practitioners.

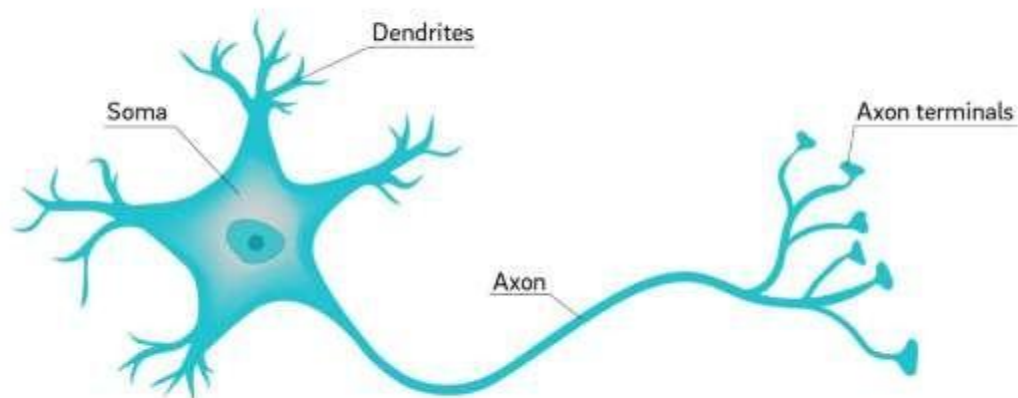
- [Google Colaboratory](#)
- [Paperspace Gradient°](#)
- [FloydHub Workspace](#)
- [Lambda GPU Cloud](#)
- [AWS Deep Learning AMIs](#)
- [GCP Deep Learning VM Images](#)

To get more information: <https://towardsdatascience.com/build-your-own-robust-deep-learning-environment-in-minutes-354cf140a5a6>

Biological Neurons:

Parts of a biological neural network

Neuron



[Image source](#)

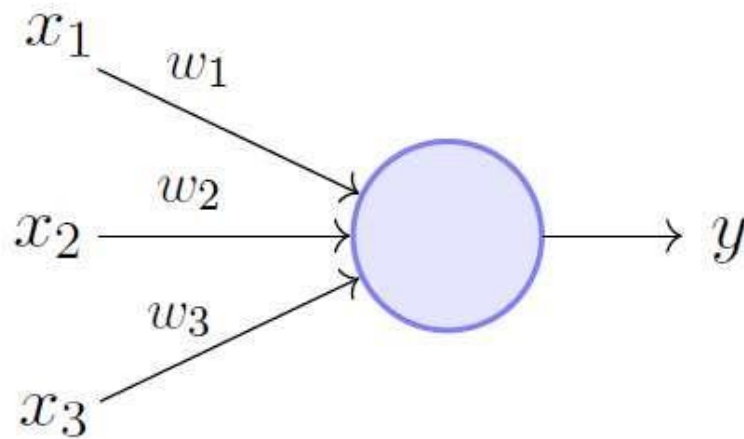
In living organisms, the brain is the control unit of the neural network, and it has different subunits that take care of vision, senses, movement, and hearing. The brain is connected with a dense network

of nerves to the rest of the body's sensors and actors. There are approximately 10^{11} neurons in the brain, and these are the building blocks of the complete central nervous system of the living body.

Why Understand Biological Neural Networks?

For creating mathematical models for artificial neural networks, theoretical analysis of biological neural networks is essential as they have a very close relationship. And this understanding of the brain's neural networks has opened horizons for the development of artificial neural network systems and adaptive systems designed to learn and adapt to the situations and inputs.

An artificial neuron

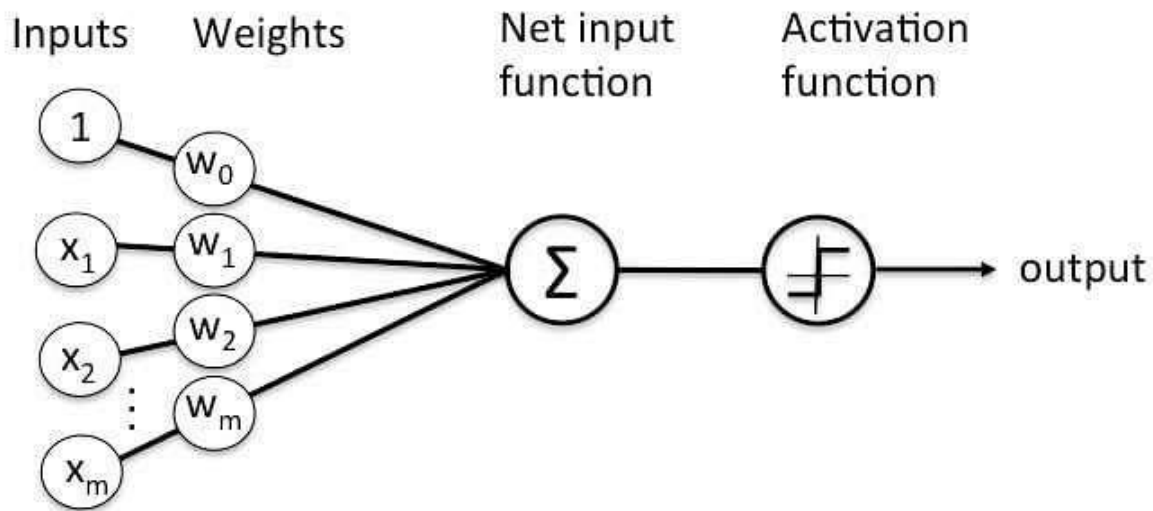


Perceptron Model (Minsky-Papert in 1969)



Perceptron

Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.

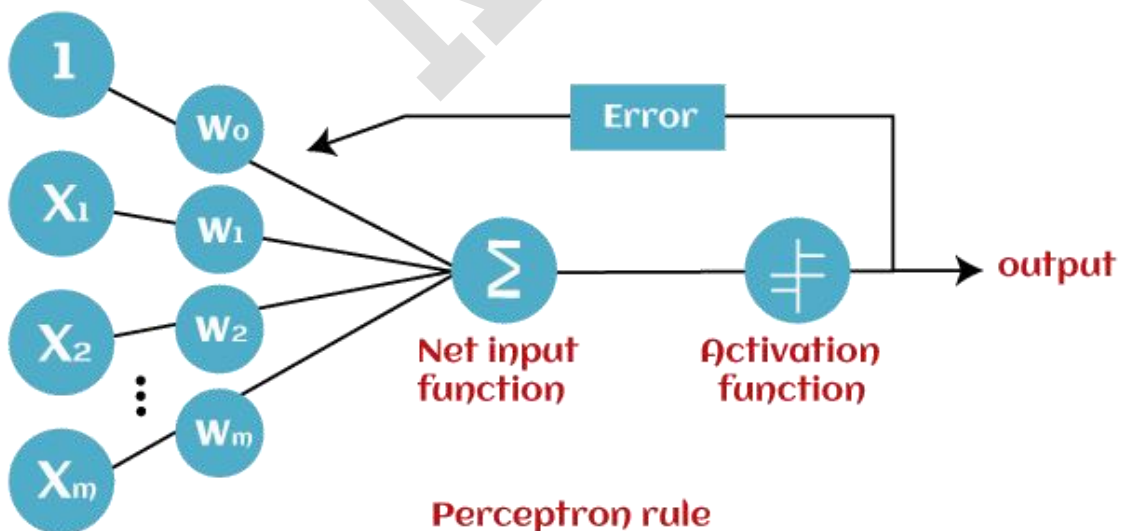


Types of Perceptron:

1. Single layer: Single layer perceptron can learn only linearly separable patterns.
2. Multilayer: Multilayer perceptrons can learn about two or more layers having a greater processing power.

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:

$$\sum w_i * x_i = x_1 * w_1 + x_2 * w_2 + \dots w_n * x_n$$

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

$$\sum w_i * x_i + b$$

Step-2

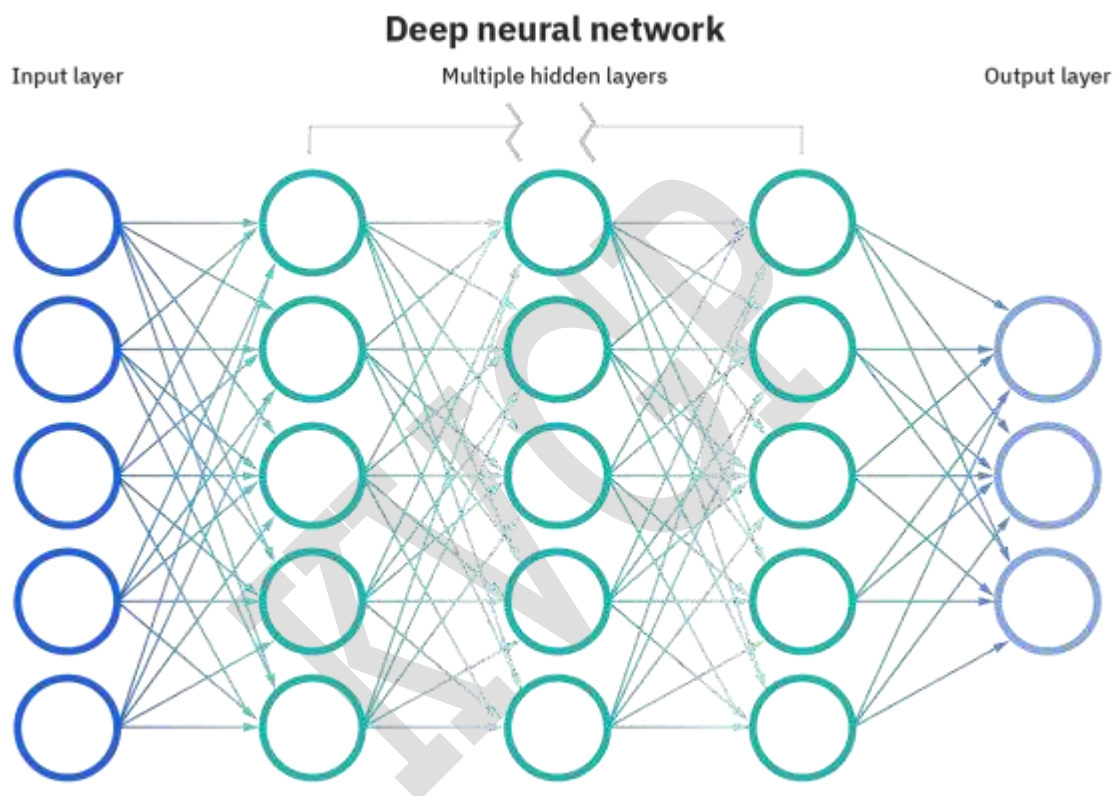
In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$$Y = f(\sum w_i * x_i + b)$$

Neural Networks:

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of [machine learning](#) and are at the heart of [deep learning](#) algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and 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.



Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and [artificial intelligence](#), allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

How do neural networks work?

Think of each individual node as its own [linear regression](#) model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like this:

$$\sum_{i=1}^m w_i x_i + \text{bias} = w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{bias}$$

$$\sum w_i x_i + \text{bias} = w_1 x_1 + w_2 x_2 + w_3 x_3 + \text{bias}$$

$$\text{output} = f(x) = \begin{cases} 1 & \text{if } \sum w_i x_i + b \geq 0 \\ 0 & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

$$\text{output} = f(x) = 1 \text{ if } \sum w_i x_i + b \geq 0; 0 \text{ if } \sum w_i x_i + b < 0$$

Forward and backward propagation in Neural Networks:

Forward Propagation is the way to move from the Input layer (left) to the Output layer (right) in the neural network.

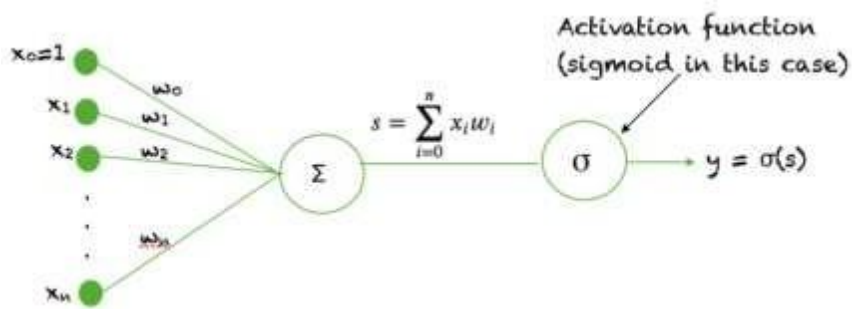
The process of moving from the right to left i.e backward from the Output to the Input layer is called the **Backward Propagation**.

Backward Propagation is the preferable method of adjusting or correcting the weights to reach the minimized loss function. In this article, we shall explore this second technique of Backward

Activation Functions:

1. Sigmoid As an Activation Function in Neural Networks

The sigmoid function is used as an activation function in neural networks. Just to review what is an activation function, the figure below shows the role of an activation function in one layer of a neural network. A weighted sum of inputs is passed through an activation function and this output serves as an input to the next layer.

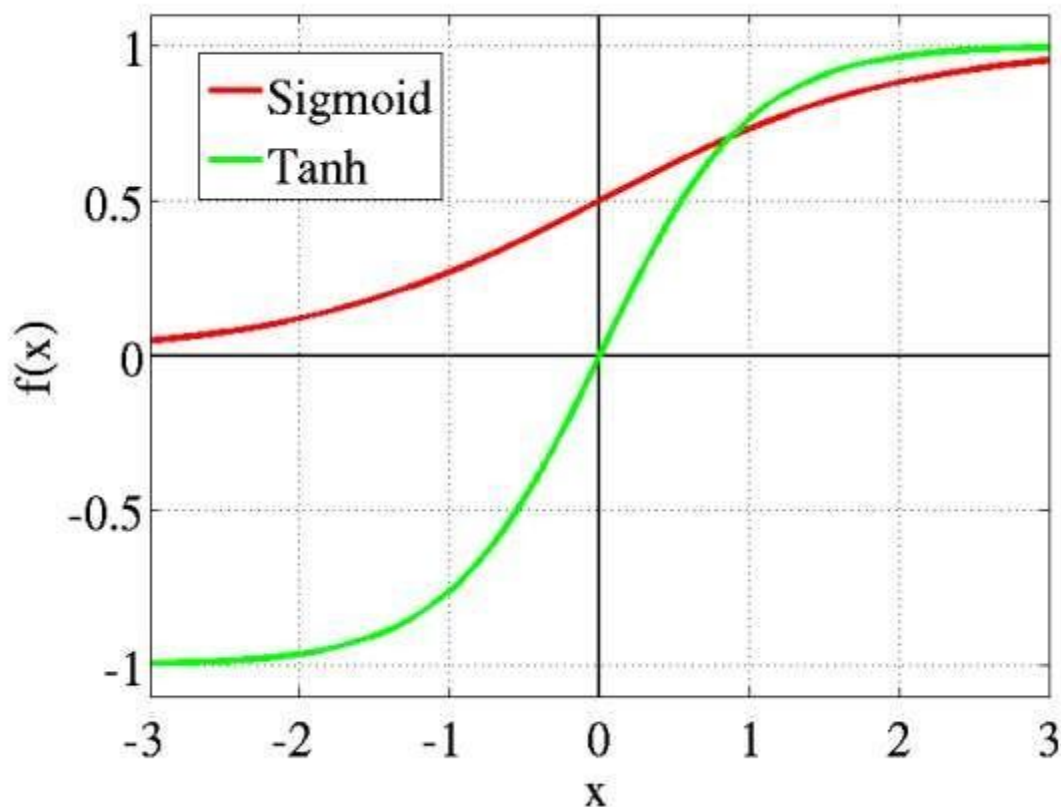


A sigmoid unit in a neural network

When the activation function for a neuron is a sigmoid function it is a guarantee that the output of this unit will always be between 0 and 1. Also, as the sigmoid is a non-linear function, the output of this unit would be a non-linear function of the weighted sum of inputs. Such a neuron that employs a sigmoid function as an activation function is termed as a sigmoid unit.

2. Tanh or hyperbolic tangent Activation Function

tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

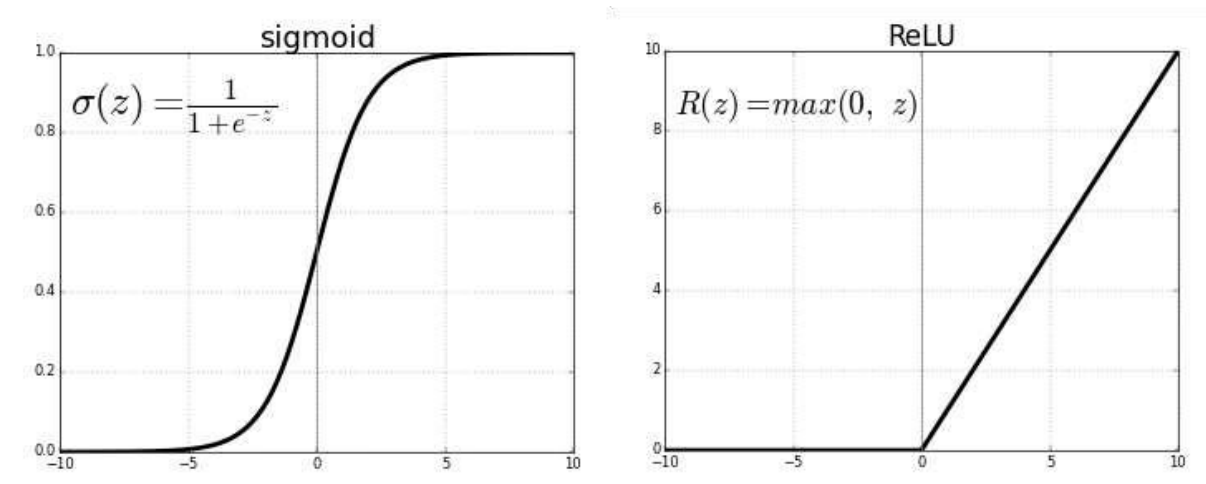


tanh v/s Logistic Sigmoid

- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.
- The function is **differentiable**.
- The function is **monotonic** while its **derivative is not monotonic**.
- The tanh function is mainly used classification between two classes.

3. ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning.



ReLU v/s Logistic Sigmoid

As you can see, the ReLU is half rectified (from bottom). $f(z)$ is zero when z is less than zero and $f(z)$ is equal to z when z is above or equal to zero.

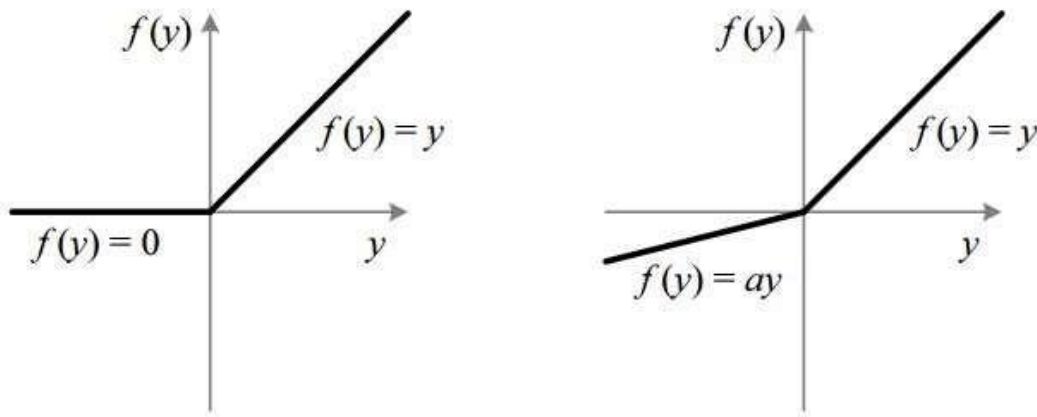
Range: [0 to infinity)

The function and its derivative **both are monotonic**.

But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

4. Leaky ReLU (Rectified Linear Unit)

It is an attempt to solve the dying ReLU problem



ReLU v/s Leaky ReLU

- The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.
- When a is not 0.01 then it is called **Randomized ReLU**.
- Therefore, the **range** of the Leaky ReLU is (-infinity to infinity).
- Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives also monotonic in nature.

Get more info: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

Cost Function:

What Is Cost Function in Machine Learning?

After training your model, you need to see how well your model is performing. While accuracy functions tell you how well the model is performing, they do not provide you with an insight on how to better improve them. Hence, you need a correctional function that can help you compute when the model is the most accurate, as you need to hit that small spot between an undertrained model and an overtrained model.

A Cost Function is used to measure just how wrong the model is in finding a relation between the input and output. It tells you how badly your model is behaving/predicting

Consider a robot trained to stack boxes in a factory. The robot might have to consider certain changeable parameters, called Variables, which influence how it performs. Let's say the robot comes across an obstacle, like a rock. The robot might bump into the rock and realize that it is not the correct action.

How to measure loss?

The formula for the loss is fairly straightforward. It is just the squared difference between the expected value and the predicted value.

$$L = (y_i - \hat{y}_i)^2$$

Suppose you have a model that helps you predict the price of oil per gallon. If the actual price of the house is \$2.89 and the model predicts \$3.07, you can calculate the error.

$$L = (2.89 - 3.07)^2 = 0.032$$

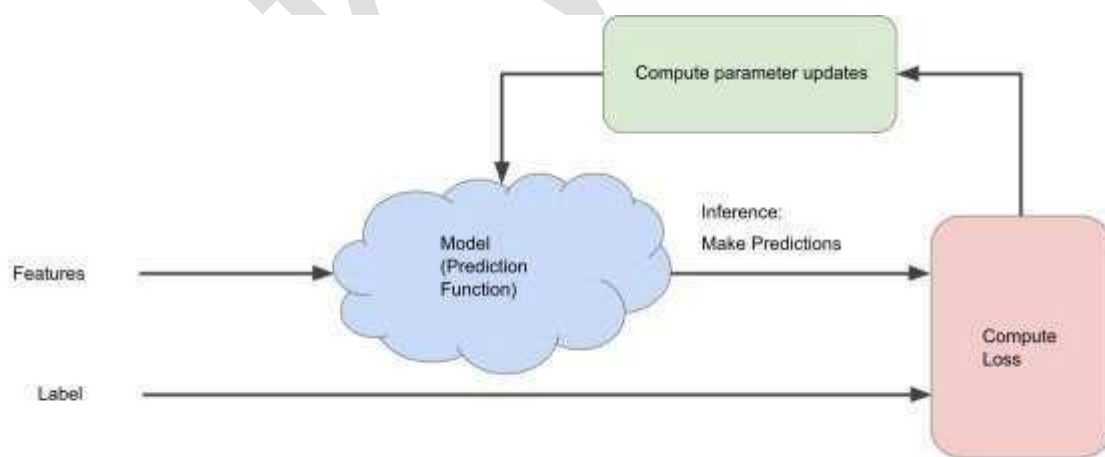
The cost is again calculated as the average overall losses for the individual examples.

$$C = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

How to reduce loss?

- Hyperparameters are the configuration settings used to tune how the model is trained.
- The derivative of $(y - y')^2$ with respect to the weights and biases tells us how loss changes for a given example
 - Simple to compute and convex
- So we repeatedly take small steps in the direction that minimizes loss
 - We call these **Gradient Steps** (But they're really negative Gradient Steps)
 - This strategy is called **Gradient Descent**

Block Diagram of Gradient Descent



So by choosing the correct optimizer (Adam Optimizer or SGD + Momentum Optimizer is recommended) and tuning the learning rate hyperparameter (Choose the small value and try training the model more and more).

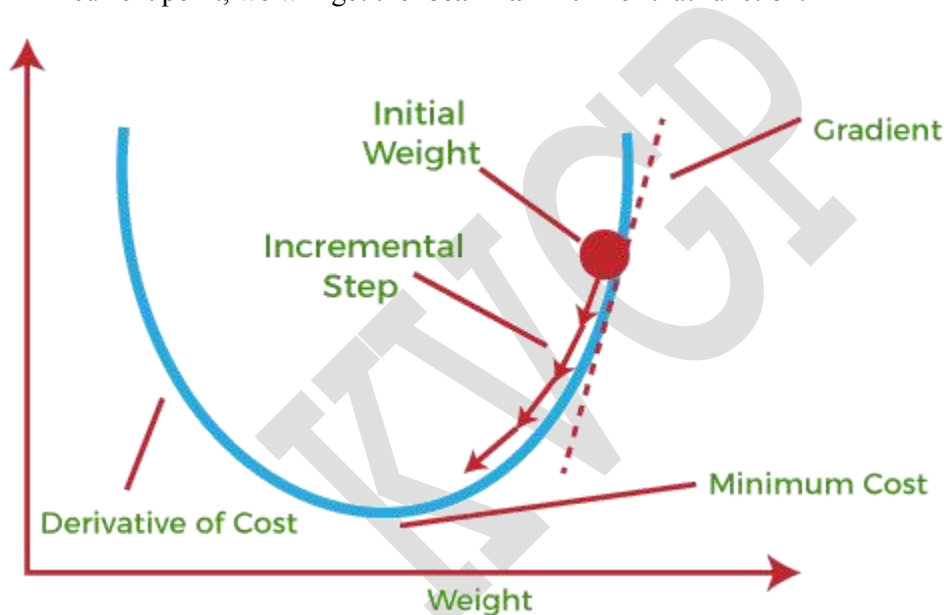
Gradient Descent or Steepest Descent:

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.

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.

TensorFlow:

What is TensorFlow?

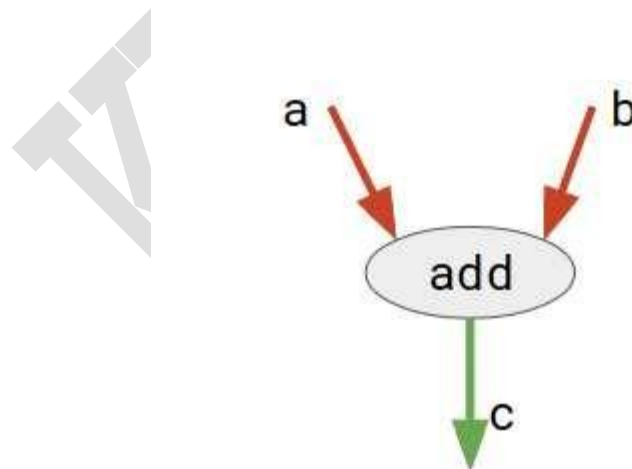
TensorFlow is a popular framework of **machine learning** and **deep learning**. It is a **free** and **open-source** library which is released on **9 November 2015** and developed by **Google Brain Team**. It is entirely based on Python programming language and use for numerical computation and data flow, which makes machine learning faster and easier.

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.

Introduction

TensorFlow is an open-source software library. **TensorFlow** was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well! Let us first try to understand what the word **TensorFlow** actually mean! **TensorFlow** is basically a software library for numerical computation using **data flow graphs** where:

- **nodes** in the graph represent mathematical operations.
- **edges** in the graph represent the multidimensional data arrays (called **tensors**) communicated between them. (Please note that **tensor** is the central unit of data in TensorFlow).



Consider the diagram given below:

Here, **add** is a node which represents addition operation. **a** and **b** are input tensors and **c** is the resultant tensor. This flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API!

TensorFlow APIs

TensorFlow provides multiple APIs (Application Programming Interfaces). These can be classified into 2 major categories:

1. Low level API:
 - complete programming control

- recommended for machine learning researchers
- provides fine levels of control over the models
- **TensorFlow Core** is the low level API of TensorFlow.

2. High level API:

- built on top of **TensorFlow Core**
- easier to learn and use than **TensorFlow Core**
- make repetitive tasks easier and more consistent between different users
- **tf.contrib.learn** is an example of a high level API.

In this article, we first discuss the basics of **TensorFlow Core** and then explore the higher level API, **tf.contrib.learn**.

TensorFlow Core

1. Installing TensorFlow

An easy to follow guide for **TensorFlow** installation is available here: [Installing TensorFlow](#). Once installed, you can ensure a successful installation by running this command in python interpreter:

```
import tensorflow as tf
```

2. The Computational Graph

Any **TensorFlow Core** program can be divided into two discrete sections:

- Building the computational graph. A **computational graph** is nothing but a series of TensorFlow operations arranged into a graph of nodes.
- Running the computational graph. To actually evaluate the nodes, we must run the computational graph within a **session**. A session encapsulates the control and state of the TensorFlow runtime.

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.

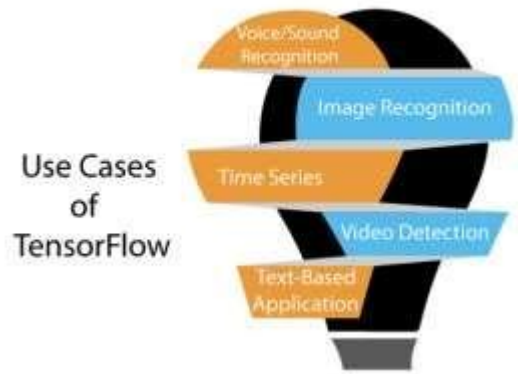
TensorFlow is used to define the flow of data in operation on a multidimensional array or Tensor.

Why tensorflow?

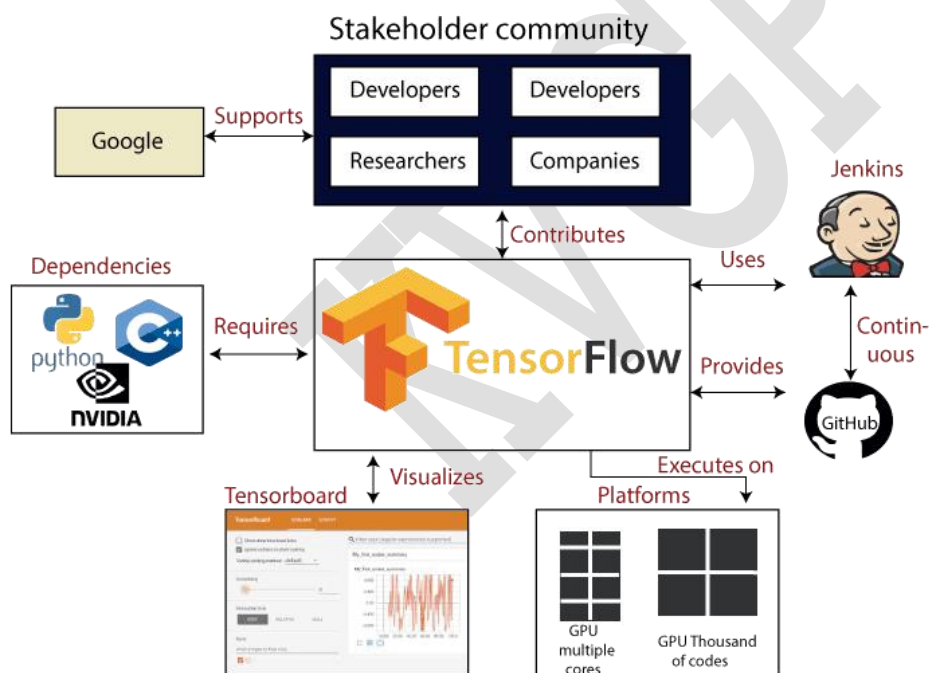
TensorFlow is the better library for all because it is accessible to everyone. TensorFlow library integrates different **API** to create a scale deep learning architecture like **CNN (Convolutional Neural Network)** or **RNN (Recurrent Neural Network)**.

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).

Use Cases/Applications of TensorFlow:



TensorFlow provides amazing functionalities and services when compared to other popular deep learning frameworks. TensorFlow is used to create a large-scale **neural network** with many layers.



It is mainly used for deep learning or machine learning problems such as **Classification, Perception, Understanding, Discovering Prediction, and Creation.**

1. Voice/Sound Recognition

Voice and sound recognition applications are the most-known use cases of deep-learning. If the neural networks have proper input data feed, neural networks are capable of understanding audio signals.

For example:

Voice recognition is used in the Internet of Things, automotive, security, and UX/UI.

Sentiment Analysis is mostly used in customer relationship management (**CRM**).

Flaw Detection (engine noise) is mostly used in automotive and Aviation.

Voice search is mostly used in customer relationship management (CRM)

2. Image Recognition

Image recognition is the first application that made deep learning and machine learning popular. Telecom, Social Media, and handset manufacturers mostly use image recognition. It is also used for face recognition, image search, motion detection, machine vision, and photo clustering.

For example, image recognition is used to recognize and identify people and objects in from of images. Image recognition is used to understand the context and content of any image.

For object recognition, TensorFlow helps to classify and identify arbitrary objects within larger images.

This is also used in engineering application to identify shape for 3D modelling purpose (3D reconstruction from 2D image) and by Facebook for photo tagging.

For example, deep learning uses TensorFlow for analyzing thousands of photos of cats. So a deep learning algorithm can learn to identify a cat because this algorithm is used to find general features of objects, animals, or people.

3. Time Series

Deep learning is using Time Series algorithms for examining the time series data to extract meaningful statistics. For example, it has used the time series to predict the stock market.

A recommendation is the most common use case for Time Series. **Amazon, Google, Facebook,** and **Netflix** are using deep learning for the suggestion. So, the deep learning algorithm is used to analyze customer activity and compare it to millions of other users to determine what the customer may like to purchase or watch.

For example, it can be used to recommend us TV shows or movies that people like based on TV shows or movies we already watched.

4. Video Detection

The deep learning algorithm is used for video detection. It is used for motion detection, real-time threat detection in gaming, security, airports, and UI/UX field.

For example, NASA is developing a deep learning network for object clustering of asteroids and orbit classification. So, it can classify and predict NEOs (**Near Earth Objects**).

5. Text-Based Applications

Text-based application is also a popular deep learning algorithm. Sentimental analysis, social media, threat detection, and fraud detection, are the example of Text-based applications.

For example, Google Translate supports over 100 languages.

Some **companies** who are *currently using TensorFlow* are Google, AirBnb, eBay, Intel, DropBox, Deep Mind, Airbus, CEVA, Snapchat, SAP, Uber, Twitter, Coca-Cola, and IBM.

TensorFlow Ecosystem:

To Build and fine-tune models with the TensorFlow ecosystem

Explore an entire ecosystem built on the [Core framework](#) that streamlines model construction, training, and export. TensorFlow supports distributed training, immediate model iteration and easy debugging with [Keras](#), and much more. Tools like [Model Analysis](#) and [TensorBoard](#) help you track development and improvement through your model's lifecycle.

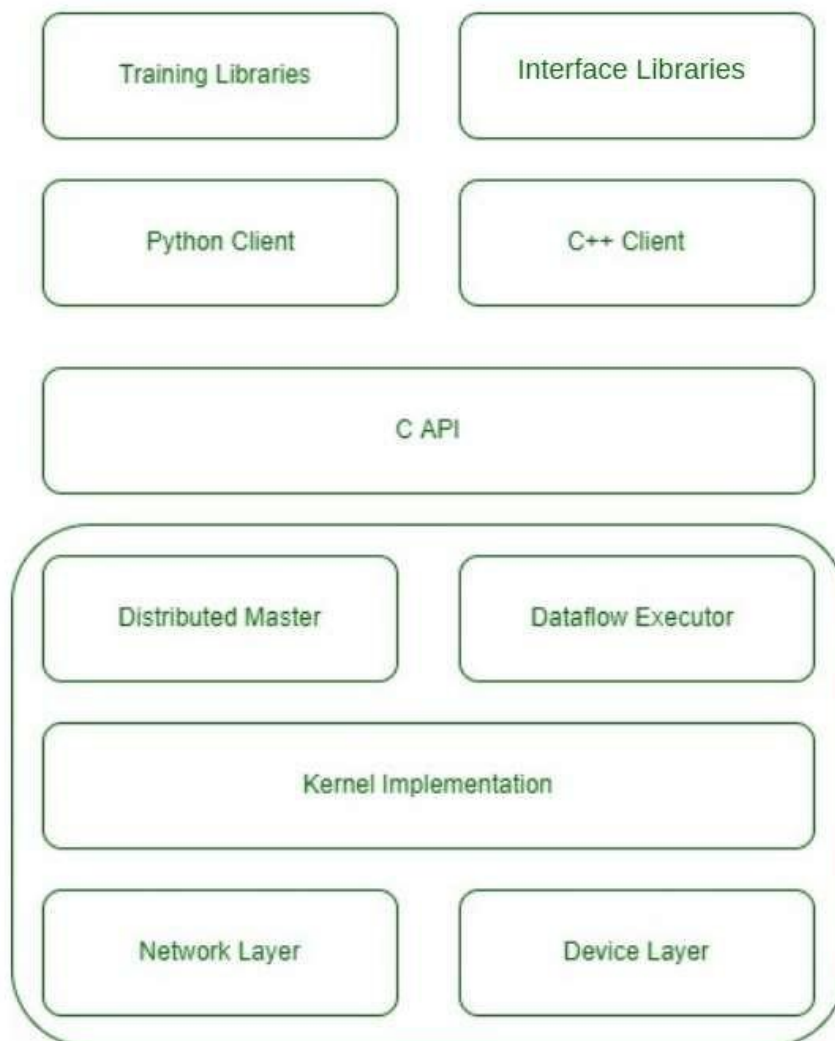
To help you get started, find collections of pre-trained models at [TensorFlow Hub](#) from Google and the community, or implementations of state-of-the-art research models in the [Model Garden](#). These libraries of high-level components allow you to take powerful models, and fine-tune them on new data or customize them to perform new tasks.

Visit: [TensorFlow](#)

KVGP

Architecture of TensorFlow

High-level Architecture



Architecture of TensorFlow

- The first layer of TensorFlow consists of the device layer and the network layer. The device layer contains the implementation to communicate to the various devices like GPU, CPU, TPU in the operating system where TensorFlow will run. Whereas the network layer has implementations to communicate with different machines using different networking protocols in the Distributable Trainable setting.
- The second layer of TensorFlow contains kernel implementations for applications mostly used in machine learning.
- The third layer of TensorFlow consists of distributed master and dataflow executors. Distributed Master has the ability to distribute workloads to different devices on the system. Whereas data flow executor performs the data flow graph optimally.

- The next layer exposes all the functionalities in the form of API which is implemented in C language. C language is chosen because it is fast, reliable, and can run on any operating system.
- The fifth layer provides support for Python and C++ clients.
- And the last layer of TensorFlow contains training and inference libraries implemented in python and C++.

TensorFlow's Basic Programming Elements

TensorFlow allows us to assign data to three kinds of data elements: constants, variables, and placeholders.

Let's take a closer look at what each of these data components represents.

1. Constants

Constants, as the name implies, are parameters with fixed values. A constant in TensorFlow is defined using the command `tf.constant()`. Constant values cannot be altered during computation.

Here's an example:

```
c = tf.constant(2.0,tf.float32)
```

```
d = tf.constant(3.0)
```

```
Print (c,d)
```

2. Variables

Variables allow new parameters to be added to the graph. The `tf.variable()` command defines a variable that must be initialized before the graph can be run in a session.

Here's an example:

```
Y = tf.Variable([.4],dtype=tf.float32)
```

```
a = tf.Variable([-4],dtype=tf.float32)
```

```
b = tf.placeholder(tf.float32)
```

```
linear_model = Y*b+a
```

3. Placeholders

Data can be fed into a model from the outside using placeholders. It later permits the assignment of values. A placeholder is defined using the command `tf.placeholder()`.

Here's an example:

```
c = tf.placeholder(tf.float32)
d = c*2
result = sess.run(d,feed_out={c:3.0})
```

The placeholder is mostly used to input data into a model. Data from the outside are fed into a graph via a variable name (in the above example, the variable name is fed out). Following that, we specify how we want to feed the data to the model while executing the session.

Example of a session:

The graph is executed by launching a session. The TensorFlow runtime is used to evaluate the nodes in the graph. The command `sess = tf.Session()` begins a session.

Example:

```
x = tf.constant(3.0)
y = tf.constant(4.0)
z = x+y
sess = tf.Session() #Launching Session
print(sess.run(z)) #Evaluating the Tensor z
```

There are three nodes in the preceding example: x, y, and z. The mathematical operation is performed at node 'z,' and the result is received afterward. When you start a session and run the node z, the nodes x and y are produced first. The addition operation will then occur at node z. As a result, we will get the answer '7'.

Keras:



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

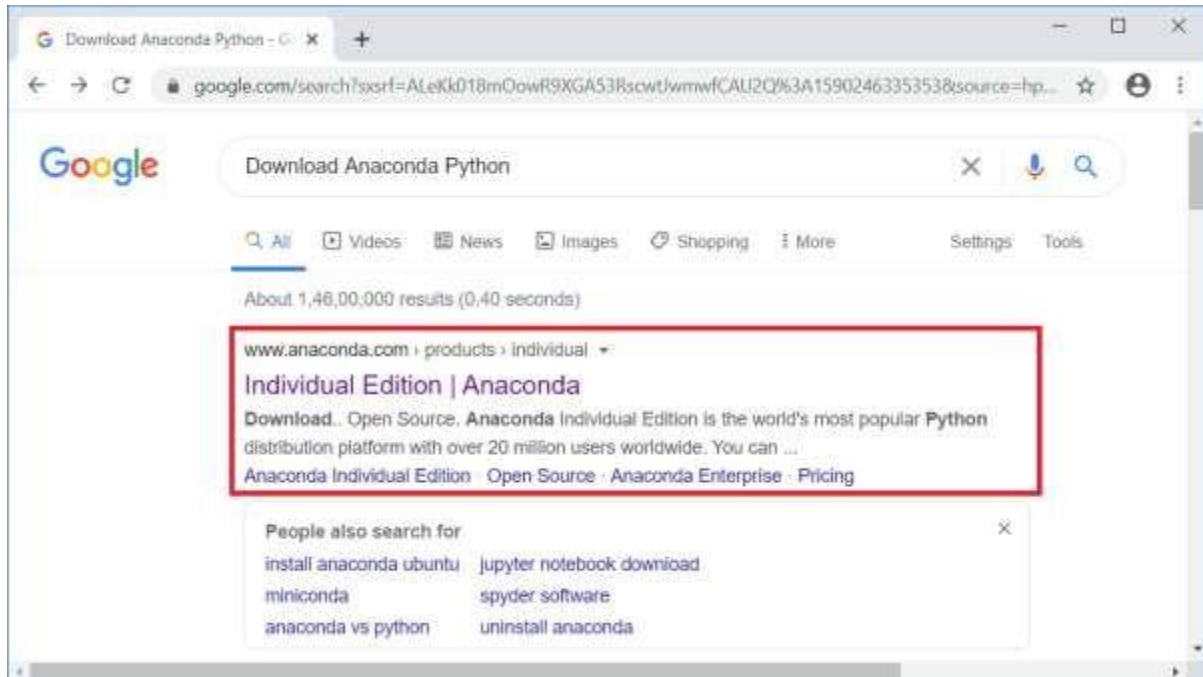
Keras provide high flexibility to all of its developers by integrating low-level deep learning languages such as TensorFlow or Theano, which ensures that anything written in the base language can be implemented in Keras.

Following are the steps that illustrate Keras installation:

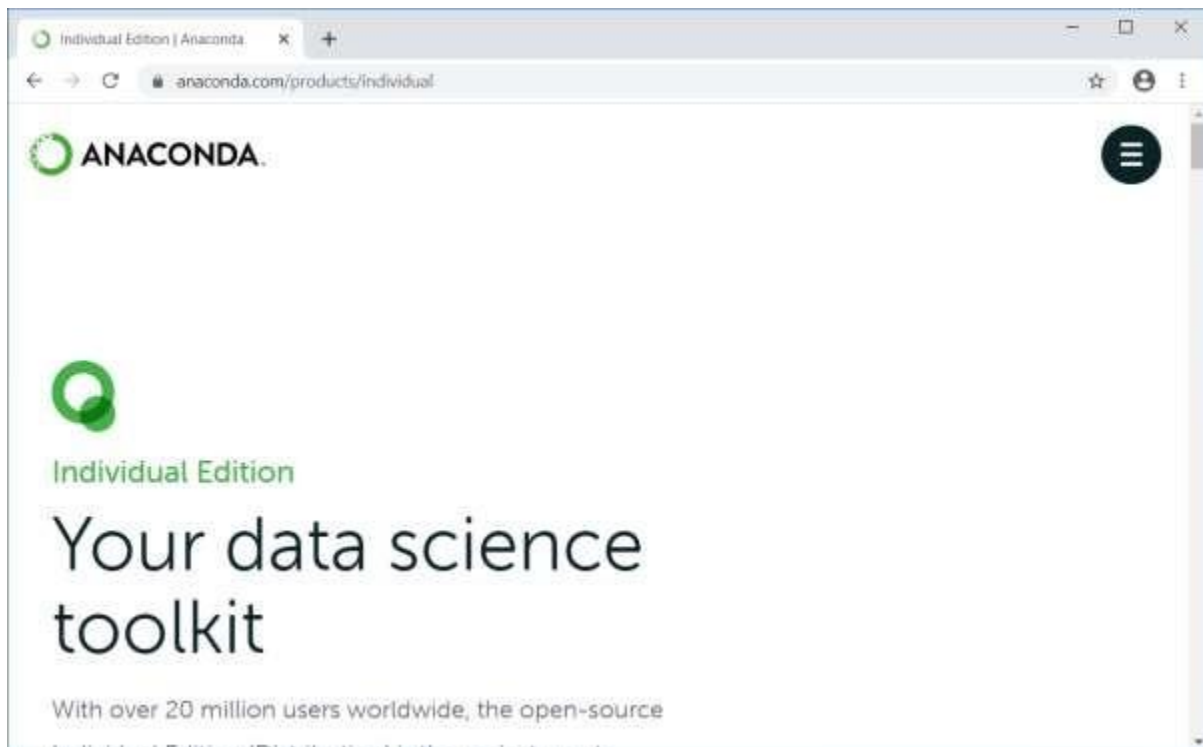
Step1: Download Anaconda Python

To download Anaconda, you can either go to one of your favorite browser and type **Download Anaconda Python** in the search bar or, simply follow the link given below.

<https://www.anaconda.com/distribution/#download-section>



Click on the very first link, and you will get directed to the Anaconda's download page, as shown below:

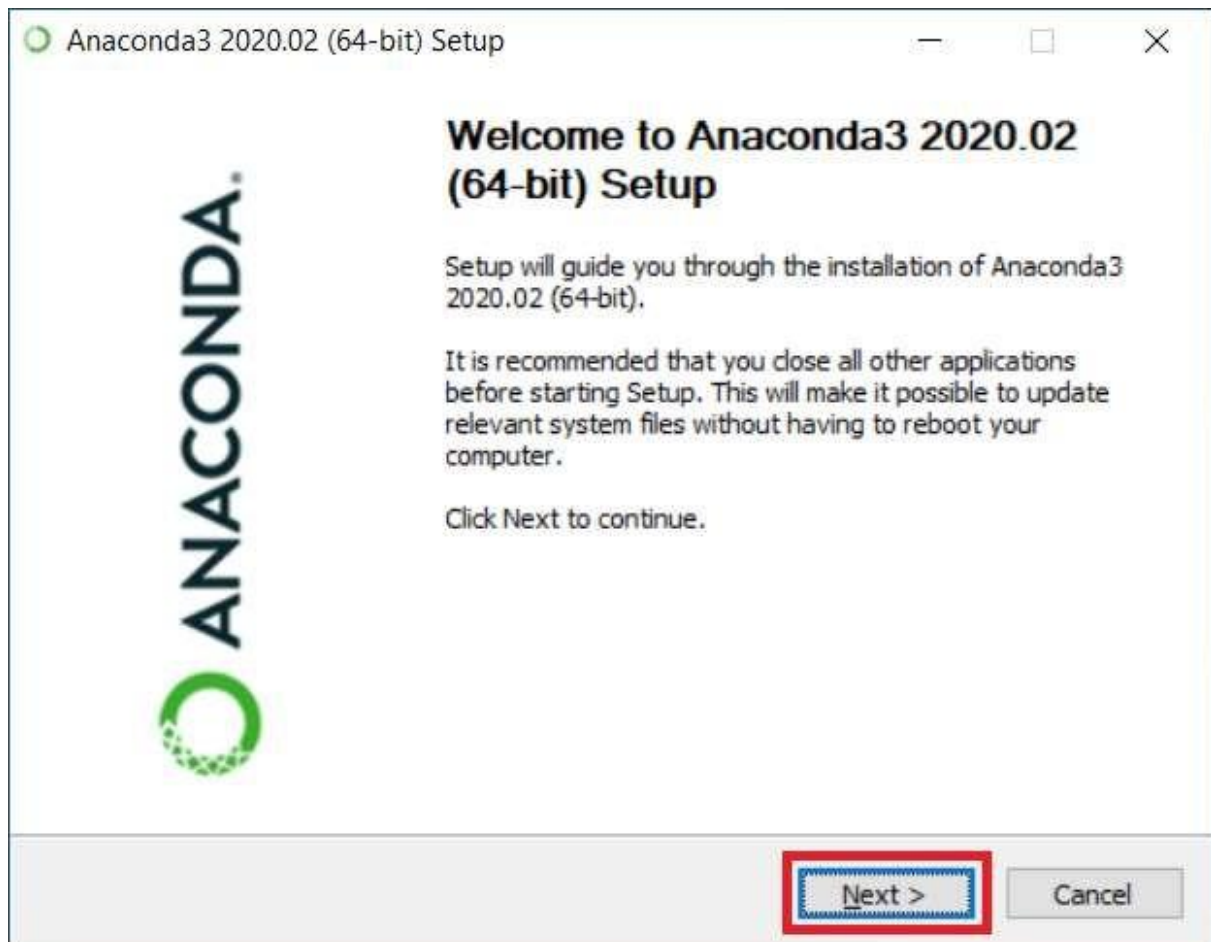


You will notice that Anaconda is available for various operating systems such as **Windows**, **MAC OS**, and **Linux**. You can download it by clicking on the available options as per your OS. It will offer you Python 2.7 and Python 3.7 version. Since the latest version is **Python 3.7**, so download it by clicking on the download option. The downloading will automatically start after you hit the download option.

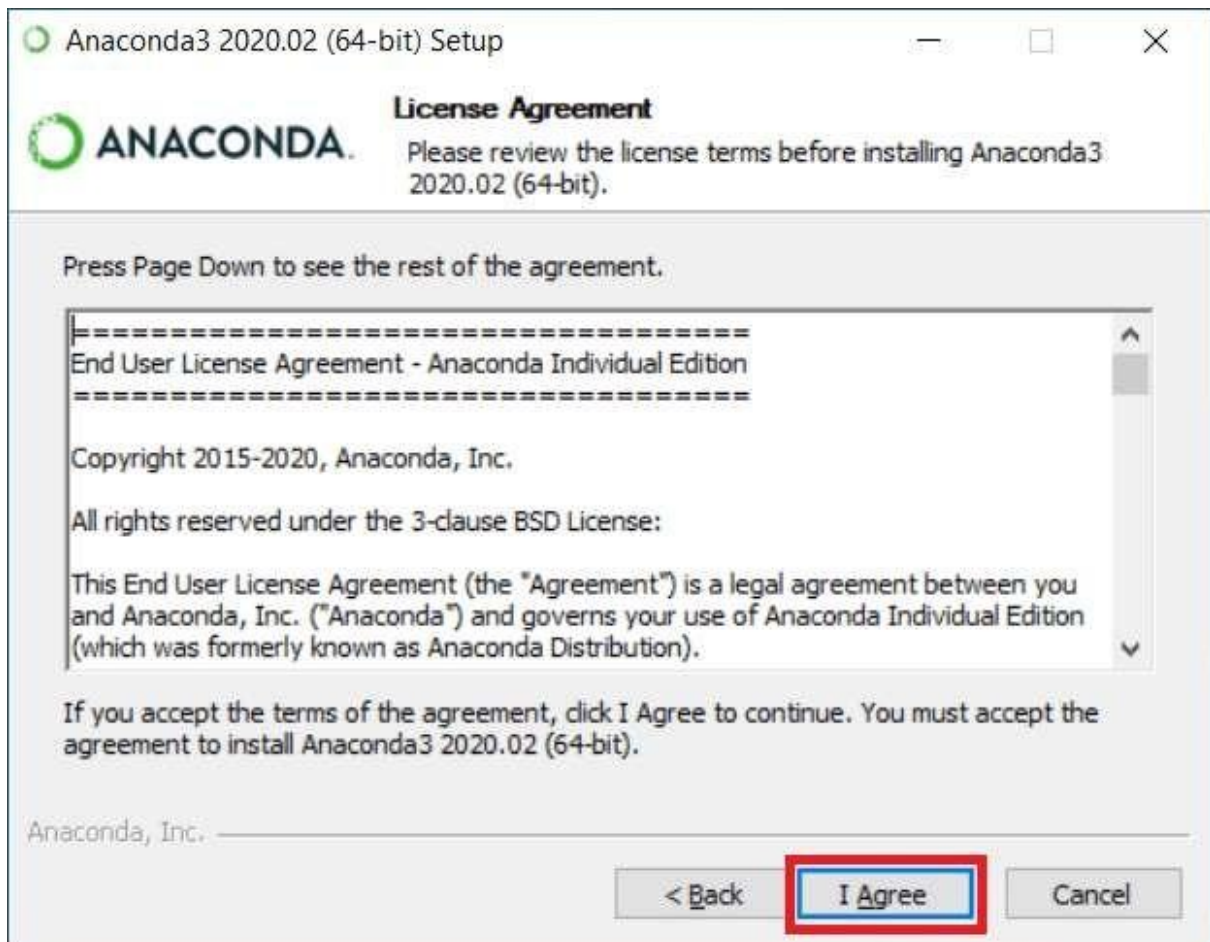


Step2: Install Anaconda Python

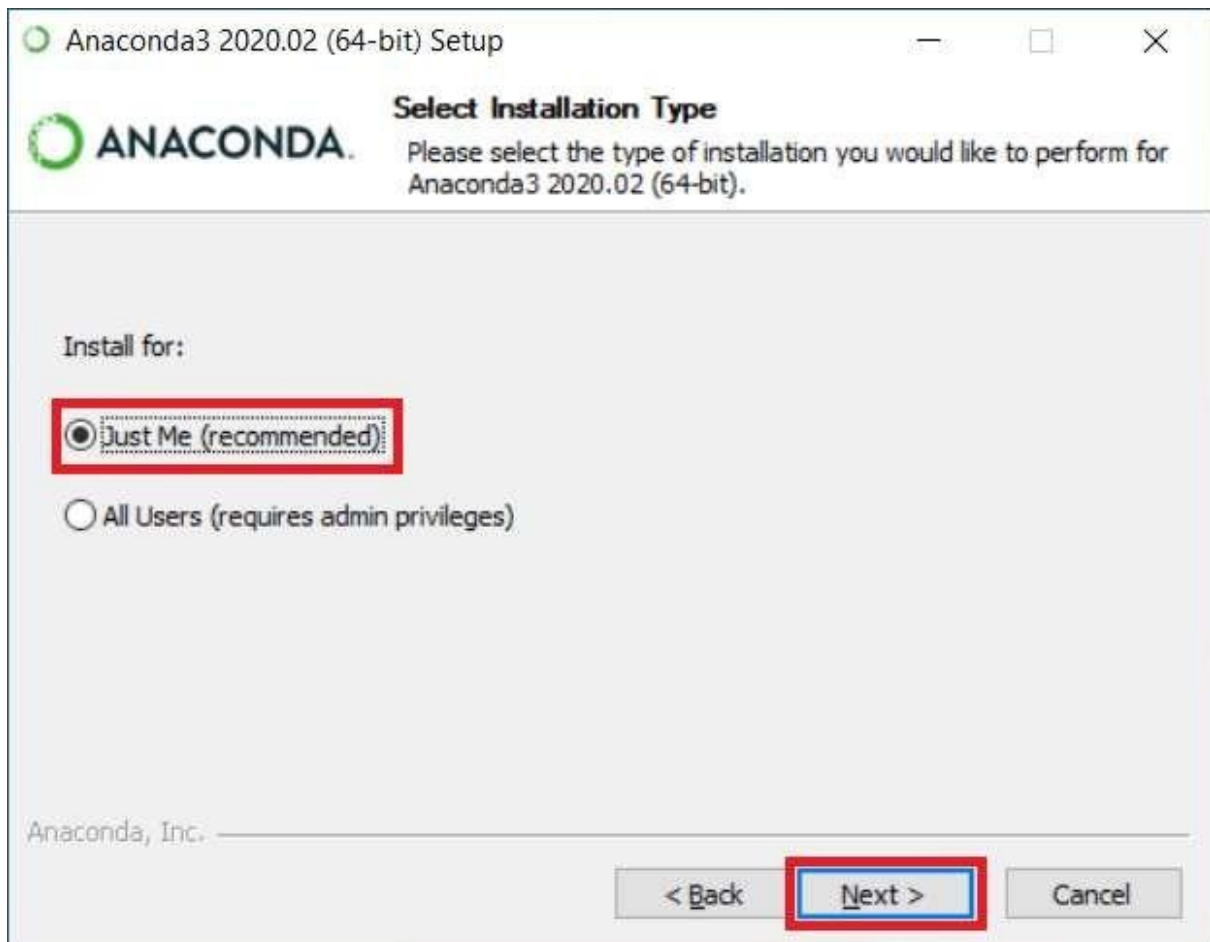
After the download is finished, go to the download folder and click on the Anaconda's **.exe** file (Anaconda3-2019.03-Windows-x86_64.exe). The setup window for the installation of Anaconda will get open up where you have to click on **Next**, as shown below:



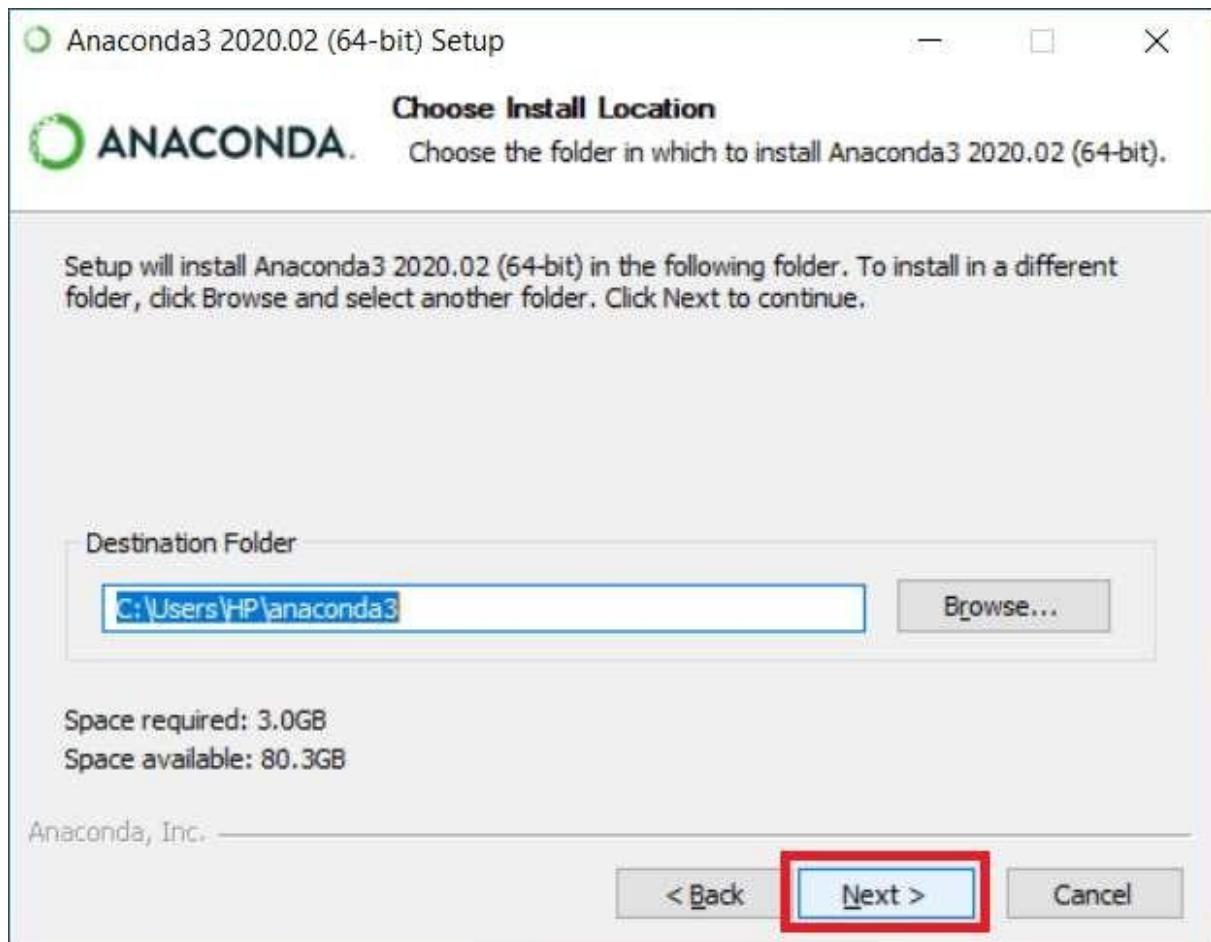
After clicking on the Next, it will open a License Agreement window, click on I Agree to move ahead with the installation.



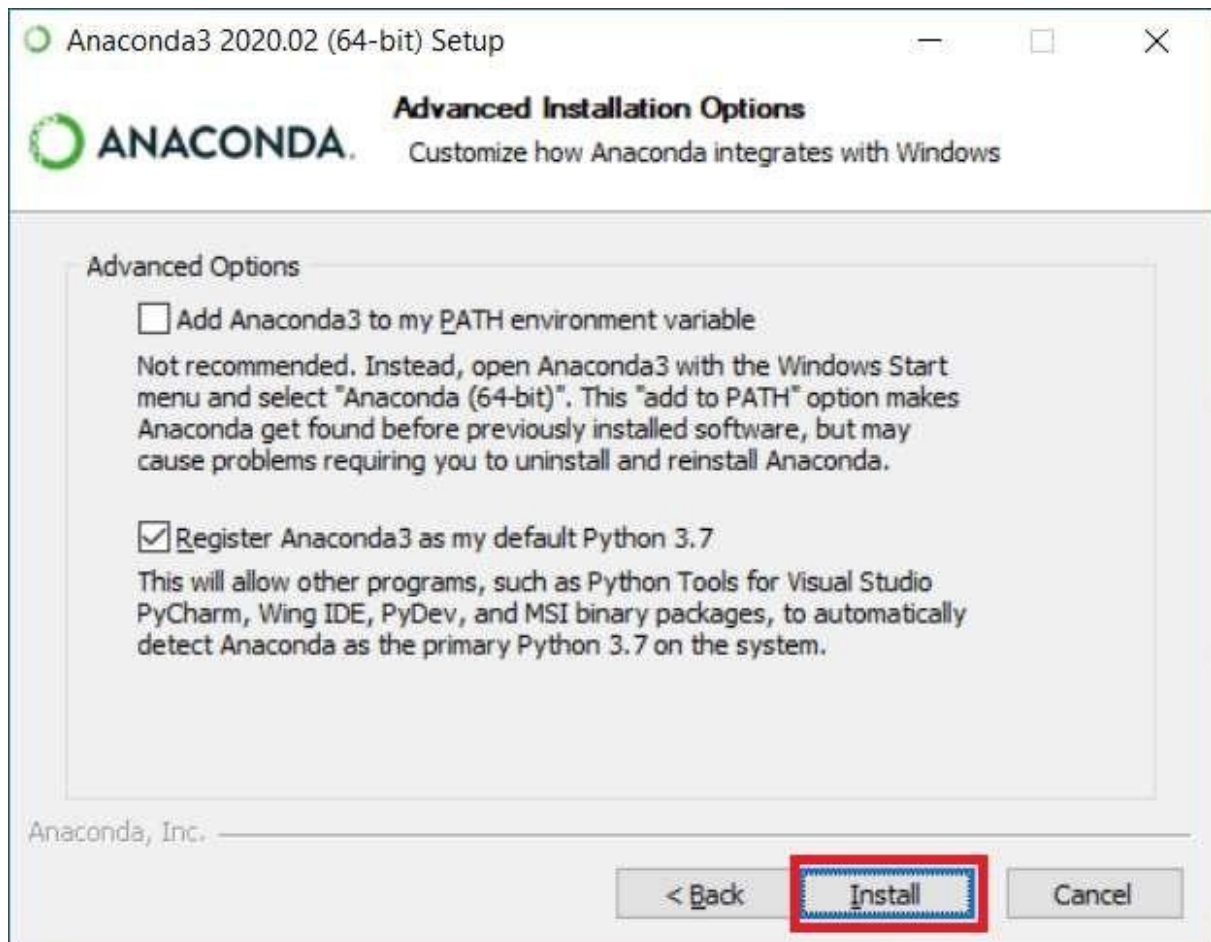
Next, you will get two options in the window; click on the first option, followed by clicking on **Next**.



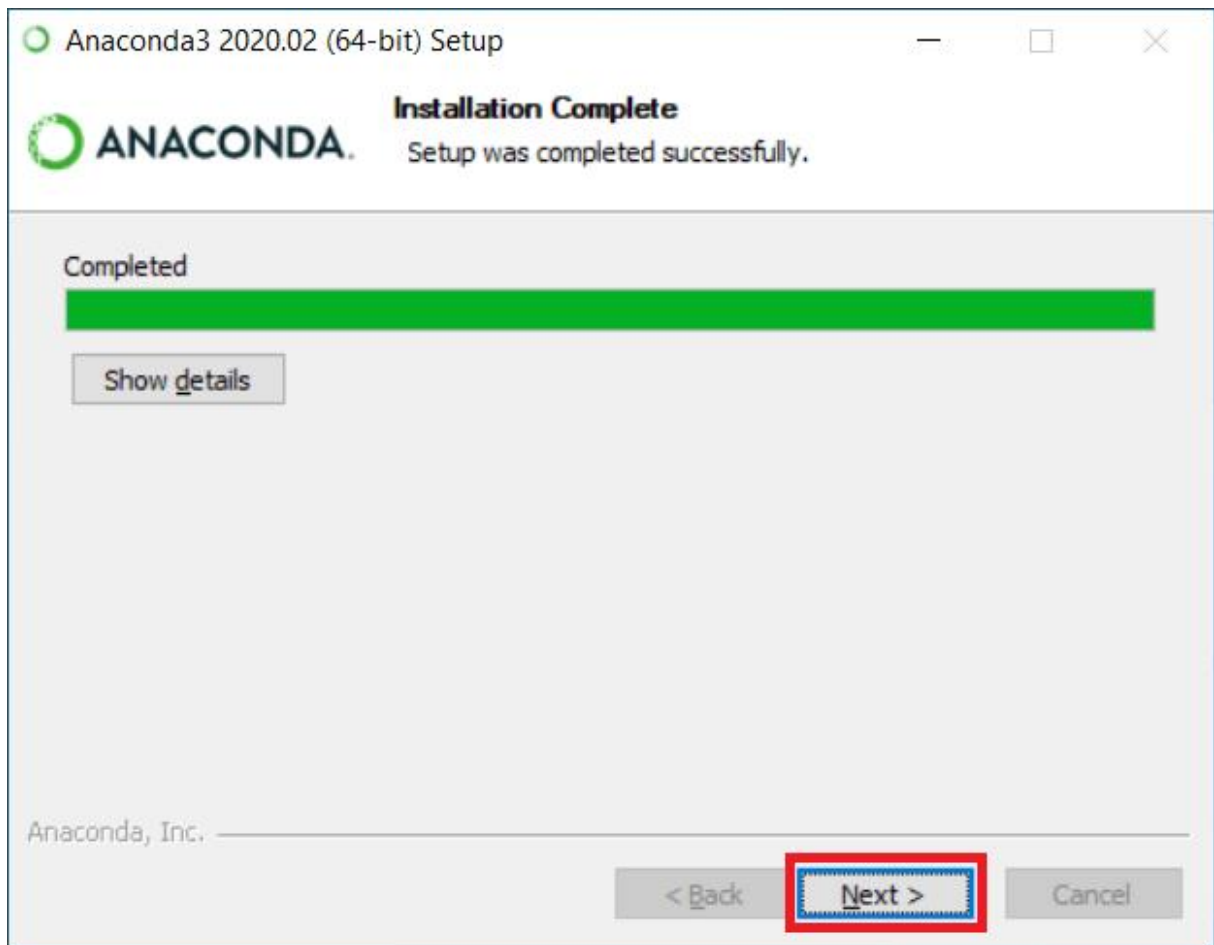
Thereafter you will be directed to the window where it will ask you for the installation location, and it's your choice to either keep it as default or change the location by browsing a location and then click on **Next**, as shown below:



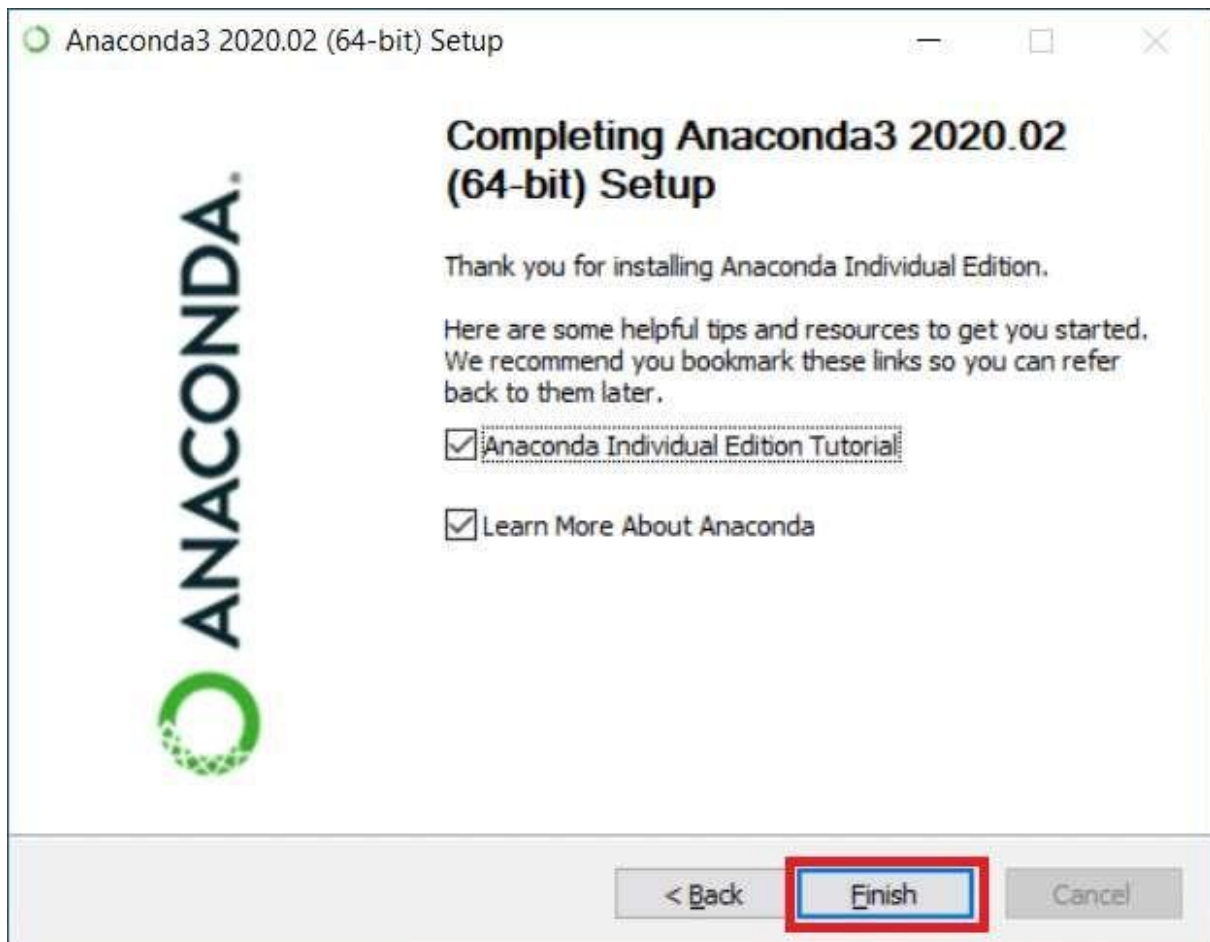
Click on Install.



Once you are done with the installation, click on **Next**.



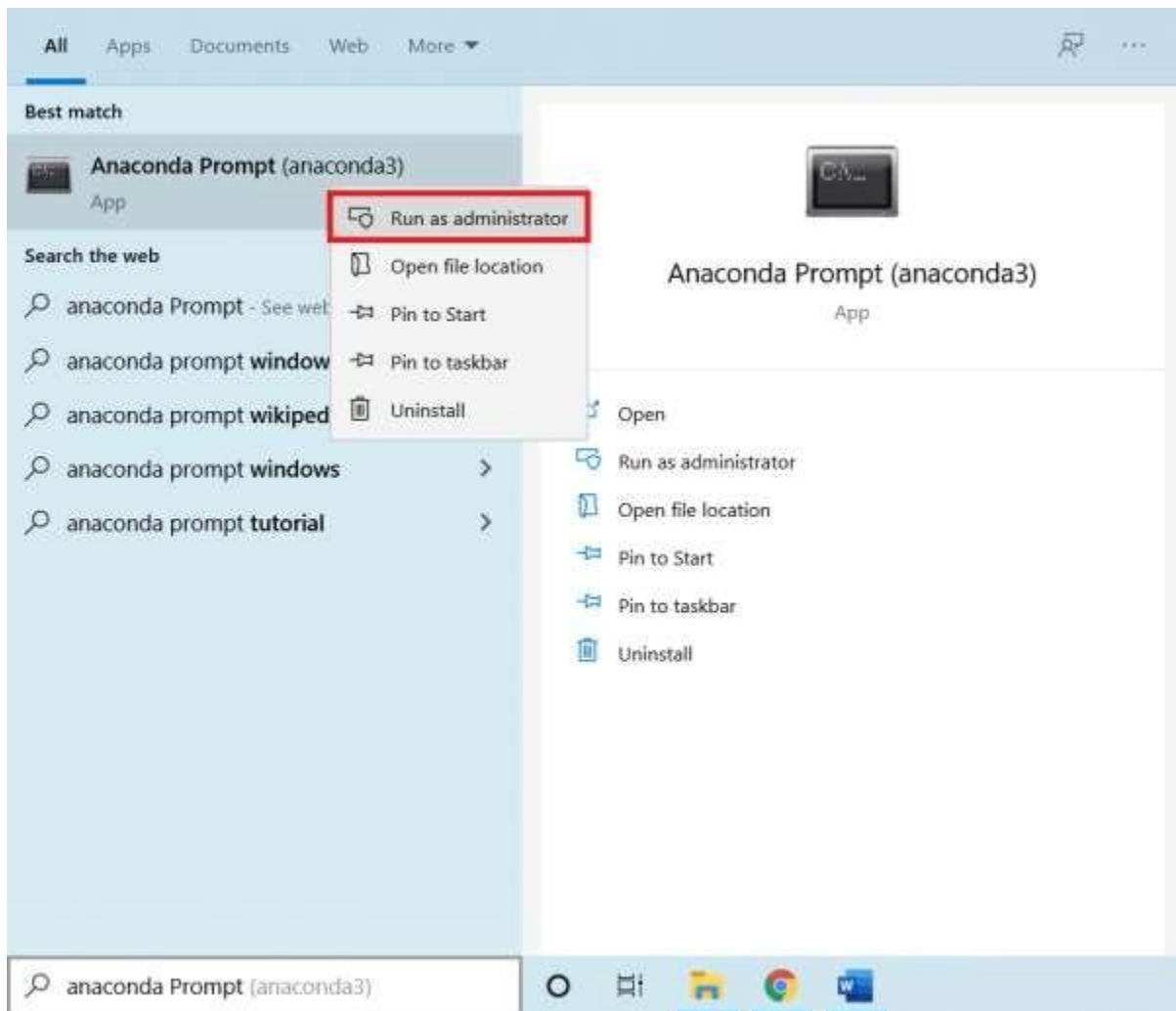
Click on Finish after the installation is completed to end the process.



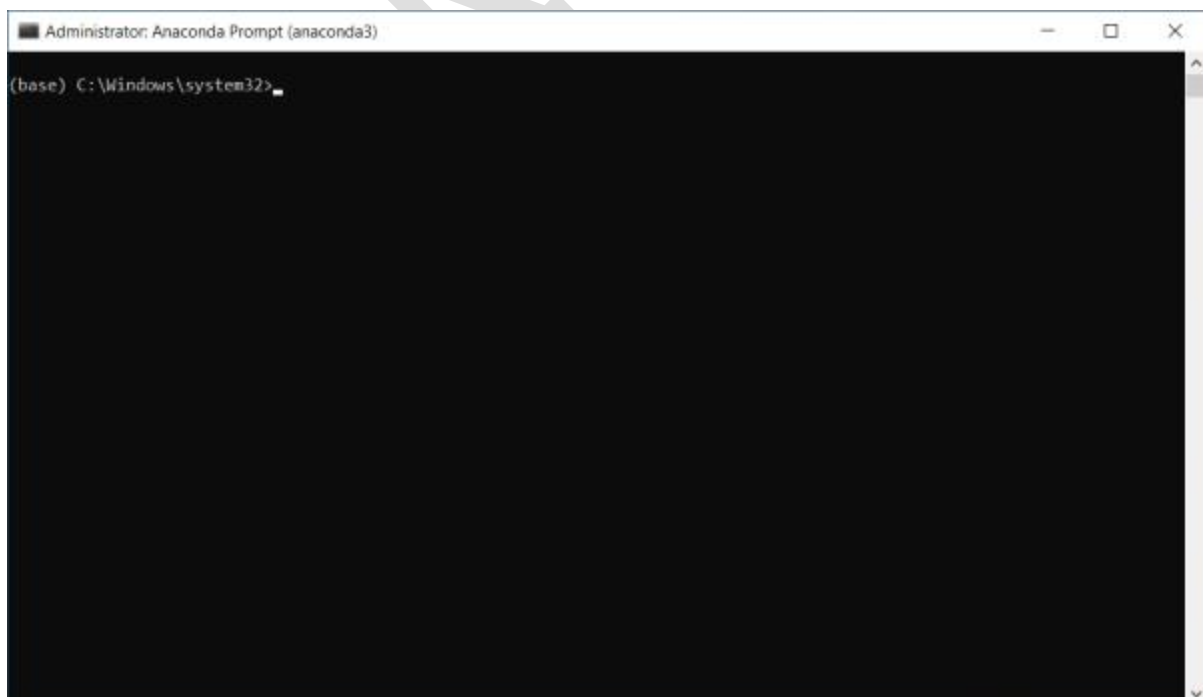
Step3: Create Environment

Now that you are done with installing Anaconda, you have to create a new conda environment where you will be installing all your modules to build your models.

You can run Anaconda prompt as an Administrator, which you can do by searching the Anaconda prompt in the search bar and then click right on it, followed by selecting the first option, which says **Run as administrator**.

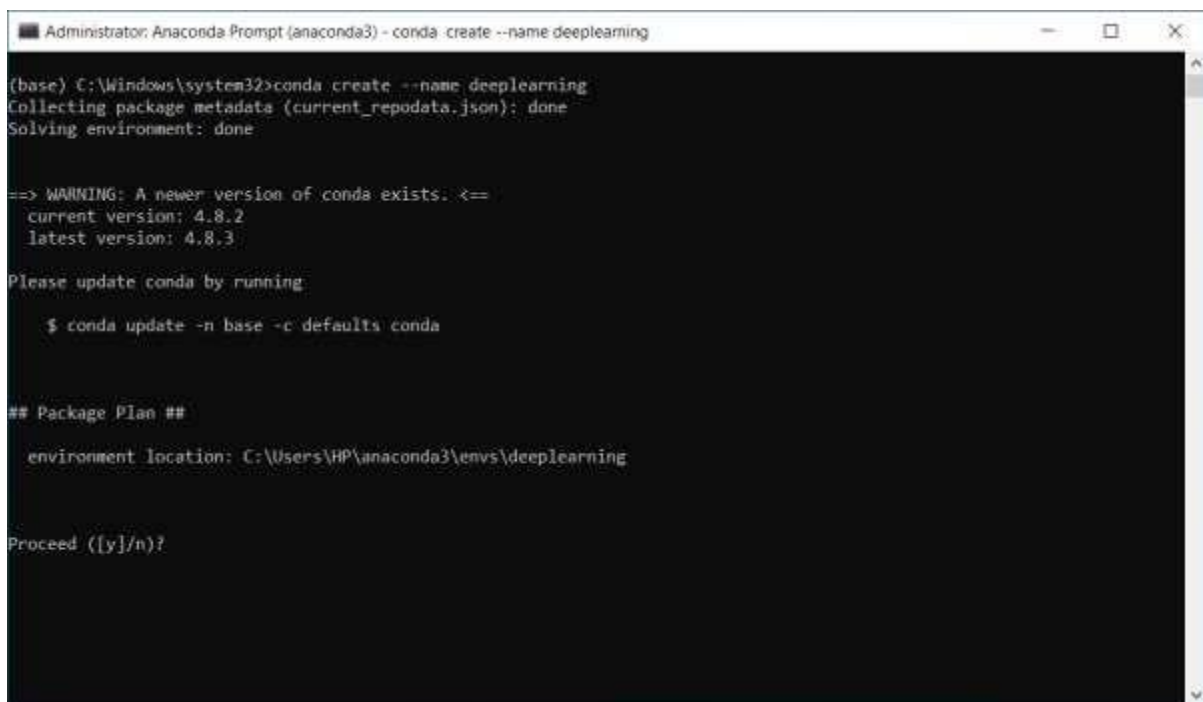


After you click on it, you will see that your anaconda prompt has opened, and it will look like the image given below.



Next, you will need to create an environment. For which you have to write the following command on the anaconda prompt and press enter. Here deeplearning specifies to the name of the environment, but you can write anything as per your choice.

1. `conda create --name deeplearning`



```
Administrator: Anaconda Prompt (anaconda3) - conda: create --name deeplearning
(base) C:\Windows\system32>conda create --name deeplearning
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.8.2
  latest version: 4.8.3

Please update conda by running

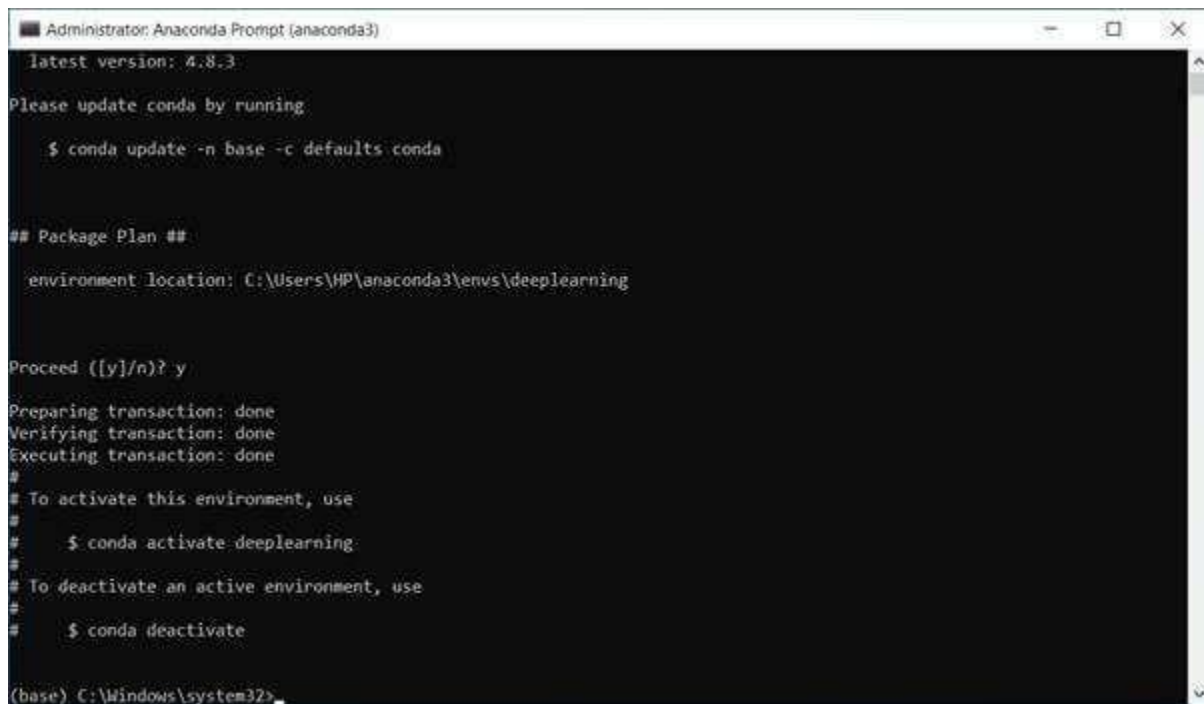
    $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\HP\anaconda3\envs\deeplearning

Proceed ([y]/n)?
```

From the image given above, you can see that it is asking you for the package plan environment location, click on y and press enter.



```
Administrator: Anaconda Prompt (anaconda3)
latest version: 4.8.3
Please update conda by running
$ conda update -n base -c defaults conda

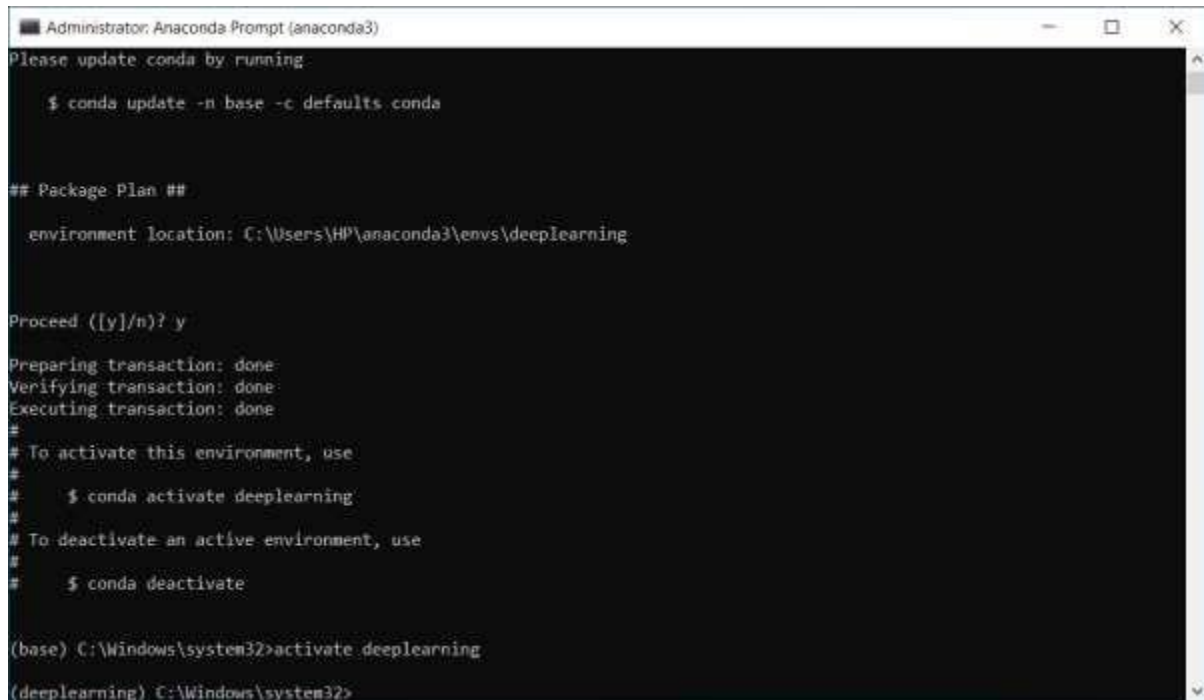
## Package Plan ##
environment location: C:\Users\HP\anaconda3\envs\deeplearning

Proceed ([y]/n)? y
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
# $ conda activate deeplearning
#
# To deactivate an active environment, use
#
# $ conda deactivate

(base) C:\Windows\system32>
```

So, you can see in the above image that you have successfully created an environment. Now the next step is to activate the environment that you created earlier. To activate the environment, write the following;

1. activate deeplearning



```
Administrator: Anaconda Prompt (anaconda3)
Please update conda by running

$ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\HP\anaconda3\envs\deeplearning

Proceed ([y]/n)? y
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
# $ conda activate deeplearning
#
# To deactivate an active environment, use
#
# $ conda deactivate

(base) C:\Windows\system32>activate deeplearning

(deeplearning) C:\Windows\system32>
```

From the above image, you can see that you are in this environment.

Next, you have to install the Keras, which you can simply do by using the below-given command.

1. `conda install -c anaconda keras`

```

Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras
pyopenssl          anaconda/win-64::pyopenssl-19.1.0-py37_0
pyreadline         anaconda/win-64::pyreadline-2.1-py37_1
pysocks            anaconda/win-64::pysocks-1.7.1-py37_0
python             anaconda/win-64::python-3.7.7-h81c818b_4
pyyaml             anaconda/win-64::pyyaml-5.3.1-py37he774522_0
requests           anaconda/win-64::requests-2.23.0-py37_0
requests-oauthlib  anaconda/noarch::requests-oauthlib-1.3.0-py_0
rsa                anaconda/noarch::rsa-4.0-py_0
scipy              anaconda/win-64::scipy-1.4.1-py37h9439919_0
setuptools         anaconda/win-64::setuptools-46.4.0-py37_0
six                anaconda/win-64::six-1.14.0-py37_0
sqlite             anaconda/win-64::sqlite-3.31.1-h2a8f88b_1
tensorboard        anaconda/noarch::tensorboard-2.1.0-py3_0
tensorflow         anaconda/win-64::tensorflow-2.1.0-eigen_py37hd727fc0_0
tensorflow-base    anaconda/win-64::tensorflow-base-2.1.0-eigen_py37h49b2757_0
tensorflow-estima anaconda/noarch::tensorflow-estimator-2.1.0-pyhd54b08b_0
termcolor          anaconda/win-64::termcolor-1.1.0-py37_1
urllib3            anaconda/win-64::urllib3-1.25.8-py37_0
vc                 anaconda/win-64::vc-14.1-h0510ff6_4
vs2015_runtime     anaconda/win-64::vs2015_runtime-14.16.27012-hf0eaf9b_2
werkzeug           anaconda/win-64::werkzeug-0.14.1-py37_0
wheel              anaconda/win-64::wheel-0.34.2-py37_0
win_inet_pton      anaconda/win-64::win_inet_pton-1.1.0-py37_0
wincertstore       anaconda/win-64::wincertstore-0.2-py37_0
wrapt              anaconda/win-64::wrapt-1.12.1-py37he774522_1
yaml               anaconda/win-64::yaml-0.1.7-vc14h4cb57cf_1
zlib               anaconda/win-64::zlib-1.2.11-vc14h1cdd9ab_1

Proceed ([y]/n)?

```

You can see that it is asking you to install the following packages, so proceed with typing y.

```

Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras
oauthlib-3.1.0      88 KB      ##### 100%
win_inet_pton-1.1.0 9 KB       ##### 100%
vs2015_runtime-14.16 2.4 MB     ##### 100%
pyyaml-5.3.1       165 KB     ##### 100%
wheel-0.34.2       67 KB      ##### 100%
hdf5-1.10.4        19.2 MB    ##### 100%
openssl-1.1.1g     5.8 MB     ##### 100%
abs1-py-0.9.0      166 KB     ##### 100%
zlib-1.2.11        117 KB     ##### 100%
astor-0.8.0        45 KB      ##### 100%
tensorboard-2.1.0  3.4 MB     ##### 100%
markdown-3.1.1     132 KB     ##### 100%
google-auth-oauthlib 21 KB      ##### 100%
vc-14.1            6 KB       ##### 100%
h5py-2.10.0        988 KB     ##### 100%
mkl_fft-1.0.15     137 KB     ##### 100%
tensorflow-2.1.0   4 KB       ##### 100%
numpy-1.18.1       5 KB       ##### 100%
setuptools-46.4.0  684 KB     ##### 100%
yaml-0.1.7         103 KB     ##### 100%
mkl-2019.4         157.5 MB   ##### 100%
wincertstore-0.2   13 KB      ##### 100%
cryptography-2.9.2 579 KB     ##### 100%
werkzeug-0.14.1    422 KB     ##### 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(deeplearning) C:\Windows\system32>

```

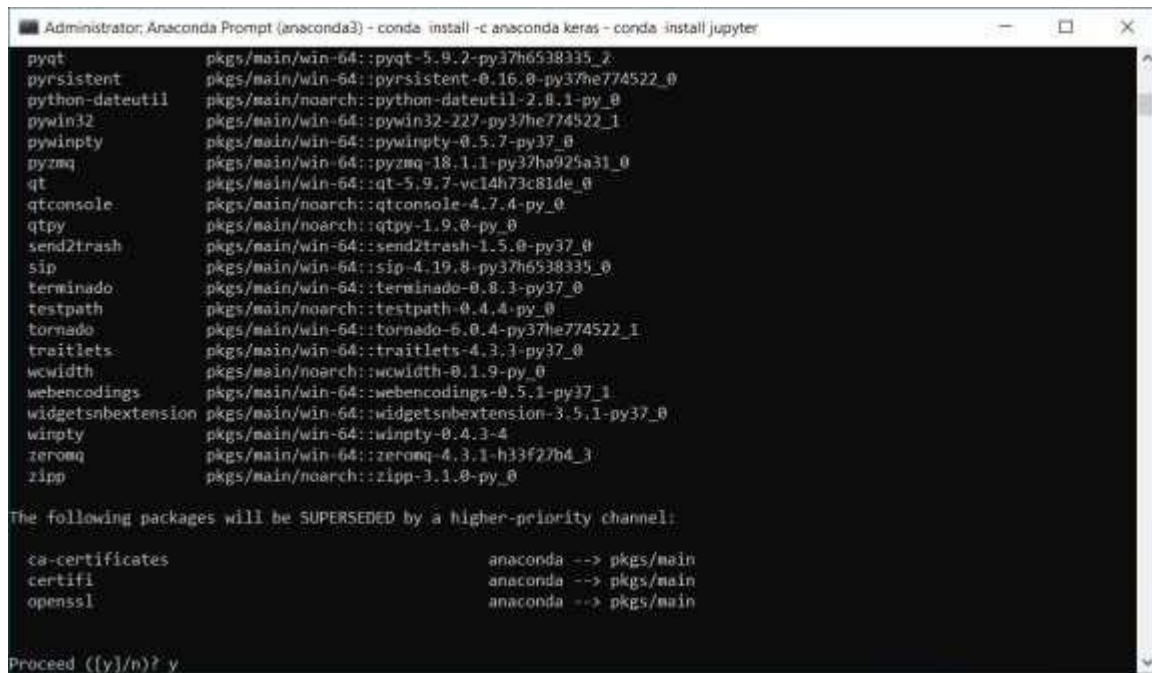
From the above image, you can see that you are done with the installation successfully.

Since this is a new environment so, you need to do a few installations again so as to avoid the occurrence of error: **ModuleNotFoundError: No module named 'keras'** while importing [Keras](#)

So, you have to run two of the most important commands because when you create an environment, **jupyter** and **spyder** are not preinstalled, that is why you have to run them.

First, you will run the command for jupyter, which is as follow:

1. conda install jupyter



```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras -c conda install jupyter

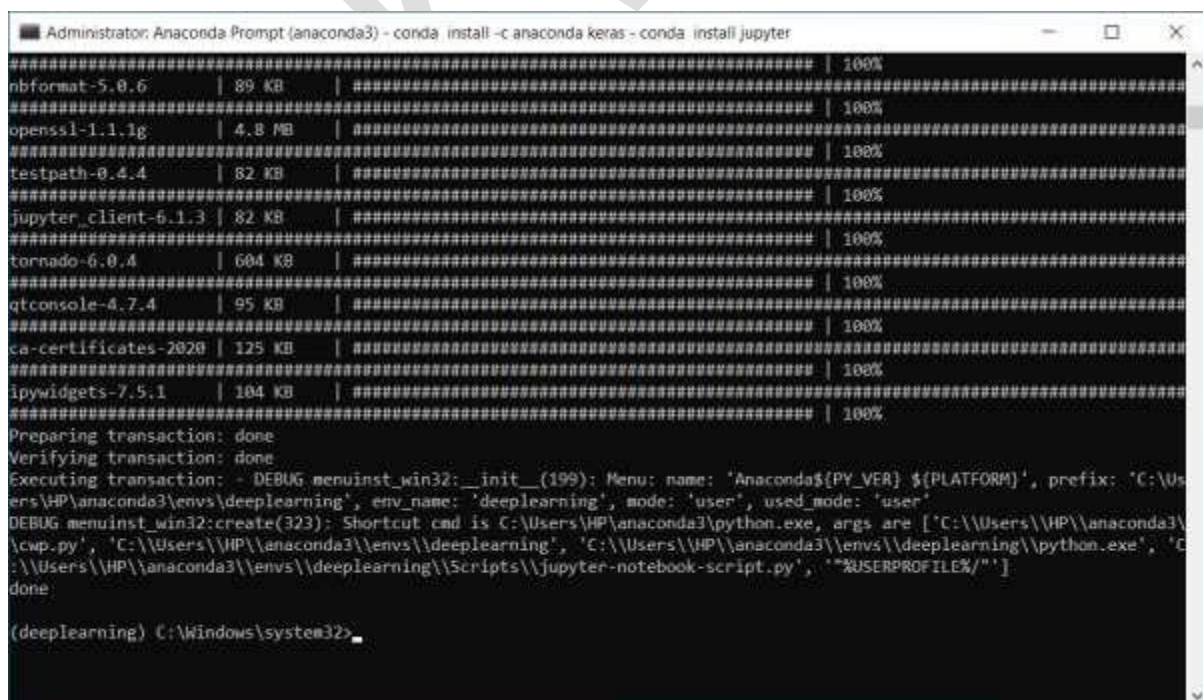
pyqt                pkgs/main/win-64::pyqt-5.9.2-py37h6538335_2
pyrsistent          pkgs/main/win-64::pyrsistent-0.16.0-py37he774522_0
python-dateutil      pkgs/main/noarch::python-dateutil-2.8.1-py_0
pywin32              pkgs/main/win-64::pywin32-227-py37he774522_1
pywinpty             pkgs/main/win-64::pywinpty-0.5.7-py37_0
pyzmq                pkgs/main/win-64::pyzmq-18.1.1-py37h925a31_0
qt                   pkgs/main/win-64::qt-5.9.7-vc14h73c81de_0
qtconsole            pkgs/main/noarch::qtconsole-4.7.4-py_0
qtpy                  pkgs/main/noarch::qtpy-1.9.0-py_0
send2trash           pkgs/main/win-64::send2trash-1.5.0-py37_0
sip                   pkgs/main/win-64::sip-4.19.8-py37h6538335_0
terminado            pkgs/main/win-64::terminado-0.8.3-py37_0
testpath             pkgs/main/noarch::testpath-0.4.4-py_0
tornado              pkgs/main/win-64::tornado-6.0.4-py37he774522_1
traitlets            pkgs/main/win-64::traitlets-4.3.3-py37_0
wcwidth              pkgs/main/noarch::wcwidth-0.1.9-py_0
webencodings         pkgs/main/win-64::webencodings-0.5.1-py37_1
widgetsnbextension  pkgs/main/win-64::widgetsnbextension-3.5.1-py37_0
winpty               pkgs/main/win-64::winpty-0.4.3-4
zmq                  pkgs/main/win-64::zmq-4.3.1-h33f27b4_3
zipp                  pkgs/main/noarch::zipp-3.1.0-py_0

The following packages will be SUPERSEDED by a higher-priority channel:

ca-certificates      anaconda --> pkgs/main
certifi               anaconda --> pkgs/main
openssl               anaconda --> pkgs/main

Proceed ([y]/n)? y
```

Again, it will ask you to install the following packages, so proceed with typing y.



```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras -c conda install jupyter

##### | 100%
nbformat-5.0.6      | 89 KB | #####
##### | 100%
openssl-1.1.1g      | 4.8 MB | #####
##### | 100%
testpath-0.4.4      | 82 KB | #####
##### | 100%
jupyter_client-6.1.3 | 82 KB | #####
##### | 100%
tornado-6.0.4       | 604 KB | #####
##### | 100%
qtconsole-4.7.4     | 95 KB | #####
##### | 100%
ca-certificates-2020 | 125 KB | #####
##### | 100%
ipywidgets-7.5.1    | 104 KB | #####
##### | 100%

Preparing transaction: done
Verifying transaction: done
Executing transaction: - DEBUG menuinst_win32: _init_(199): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\HP\anaconda3\envs\deeplearning', env_name: 'deeplearning', mode: 'user', used_mode: 'user'
DEBUG menuinst_win32:create(323): Shortcut cmd is C:\Users\HP\anaconda3\python.exe, args are ['C:\Users\HP\anaconda3\python.exe', 'C:\Users\HP\anaconda3\envs\deeplearning\Scripts\jupyter-notebook-script.py', "%USERPROFILE%\"]
done

(deeplearning) C:\Windows\system32>
```

You can see in the above image that it has been successfully installed.

Next, you will do the same for spyder.

1. conda install spyder

```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras - conda install jupyter - conda install spyder

python-jsonrpc-server pkgs/main/noarch::python-jsonrpc-server-0.3.4-py_0
python-language-server pkgs/main/win-64::python-language-server-0.31.10-py37_0
pytz pkgs/main/noarch::pytz-2020.1-py_0
pywin32-ctypes pkgs/main/win-64::pywin32-ctypes-0.2.0-py37_1000
qdarkstyle pkgs/main/noarch::qdarkstyle-2.8.1-py_0
qtawesome pkgs/main/noarch::qtawesome-0.7.0-py_0
rope pkgs/main/noarch::rope-0.17.0-py_0
rtree pkgs/main/win-64::rtree-0.9.4-py37h21ff451_1
snowballstemmer pkgs/main/noarch::snowballstemmer-2.0.0-py_0
sortedcontainers pkgs/main/win-64::sortedcontainers-2.1.0-py37_0
sphinx pkgs/main/noarch::sphinx-3.0.3-py_0
sphinxcontrib-applehelp pkgs/main/noarch::sphinxcontrib-applehelp-1.0.2-py_0
sphinxcontrib-devhelp pkgs/main/noarch::sphinxcontrib-devhelp-1.0.2-py_0
sphinxcontrib-htmlhelp pkgs/main/noarch::sphinxcontrib-htmlhelp-1.0.3-py_0
sphinxcontrib-jsmath pkgs/main/noarch::sphinxcontrib-jsmath-1.0.1-py_0
sphinxcontrib-qthelp pkgs/main/noarch::sphinxcontrib-qthelp-1.0.3-py_0
sphinxcontrib-serializinghtml pkgs/main/noarch::sphinxcontrib-serializinghtml-1.1.4-py_0
spyder pkgs/main/win-64::spyder-4.1.3-py37_0
spyder-kernels pkgs/main/win-64::spyder-kernels-1.9.1-py37_0
ujson pkgs/main/win-64::ujson-1.35-py37hf6e2cd_0
watchdog pkgs/main/win-64::watchdog-0.10.2-py37_0
yapf pkgs/main/noarch::yapf-0.28.0-py_0

The following packages will be DOWNGRADED:

jedi 0.17.0-py37_0 --> 0.15.2-py37_0
parso 0.7.0-py_0 --> 0.5.2-py_0

Proceed ([y]/n)?
```

Since you are doing for the very first time, so it will again ask you for y/n, so you just simply proceed by clicking on y as you did before.

```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras - conda install jupyter - conda install spyder

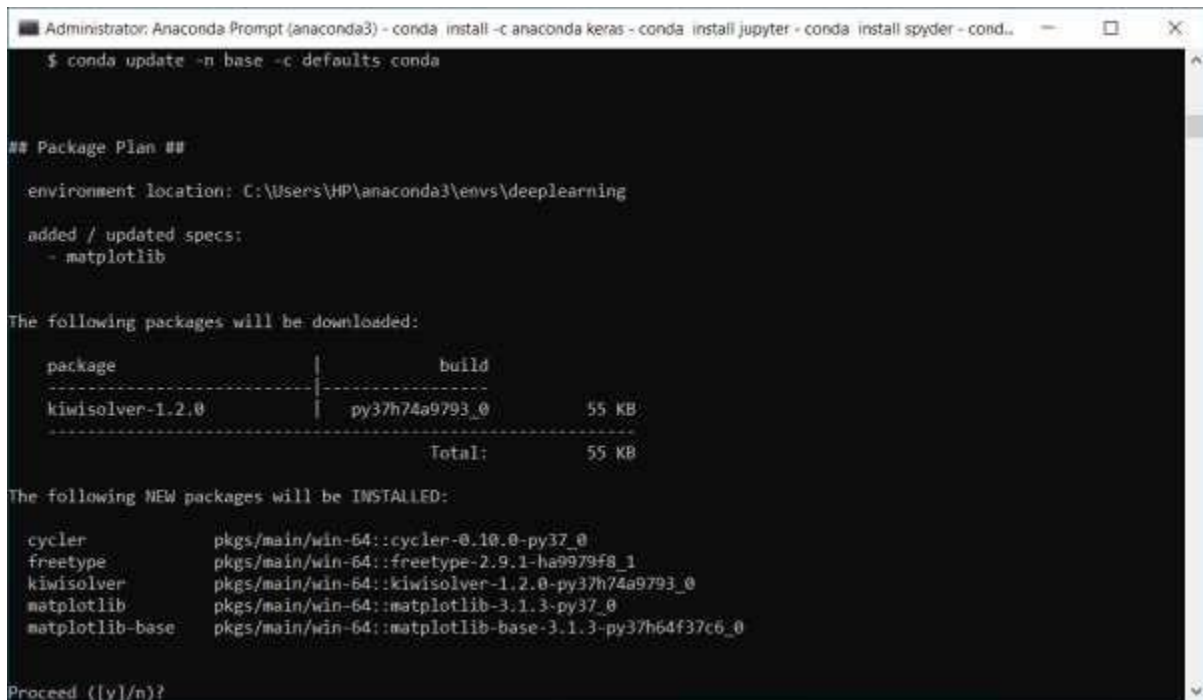
sphinxcontrib-htmlhelp 27 KB |#####| 100%
sphinxcontrib-serial 24 KB |#####| 100%
qtawesome-0.7.0 726 KB |#####| 100%
sphinx-3.0.3 1.1 MB |#####| 100%
packaging-20.3 36 KB |#####| 100%
sphinxcontrib-devhel 22 KB |#####| 100%
sphinxcontrib-jsmath 9 KB |#####| 100%
intervaltree-3.0.2 25 KB |#####| 100%
qdarkstyle-2.8.1 176 KB |#####| 100%
pytz-2020.1 184 KB |#####| 100%
sphinxcontrib-qthelp 25 KB |#####| 100%
jedi-0.15.2 738 KB |#####| 100%
cloudpickle-1.4.1 30 KB |#####| 100%
snowballstemmer-2.0 62 KB |#####| 100%
spyder-kernels-1.9.1 96 KB |#####| 100%
pyparsing-2.4.7 65 KB |#####| 100%
pydocstyle-4.0.1 35 KB |#####| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: \ DEBUG menuinst_win32: __init__(199): Menu: name: 'Anaconda${PY_VER} ${PLATFORM}', prefix: 'C:\Users\HP\anaconda3\envs\deeplearning', env_name: 'deeplearning', mode: 'user', used_mode: 'user'
DEBUG menuinst_win32:create(323): Shortcut cmd is C:\Users\HP\anaconda3\pythonw.exe, args are ['C:\Users\HP\anaconda3\envs\deeplearning\pythonw.exe', '\\cwp.py', 'C:\Users\HP\anaconda3\envs\deeplearning', 'C:\Users\HP\anaconda3\envs\deeplearning\Scripts\spyder-script.py']
| DEBUG menuinst_win32:create(323): Shortcut cmd is C:\Users\HP\anaconda3\python.exe, args are ['C:\Users\HP\anaconda3\envs\deeplearning\python.exe', 'C:\Users\HP\anaconda3\envs\deeplearning\Scripts\spyder-script.py', '--reset']
done

(deeplearning) C:\Windows\system32>
```


You can see that your installation is successfully completed.

You would require to install matplotlib for visualization. Again, the same procedure will be carried out.

1. conda install matplotlib



```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras - conda install jupyter - conda install spyder - conda...
$ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\HP\anaconda3\envs\deeplearning

  added / updated specs:
    - matplotlib

The following packages will be downloaded:



| package          | build          | size  |
|------------------|----------------|-------|
| kiwisolver-1.2.0 | py37h74a9793_0 | 55 KB |
| Total:           |                | 55 KB |



The following NEW packages will be INSTALLED:

cycler                pkgs/main/win-64::cycler-0.10.0-py37_0
freetype              pkgs/main/win-64::freetype-2.9.1-ha9979f8_1
kiwisolver            pkgs/main/win-64::kiwisolver-1.2.0-py37h74a9793_0
matplotlib            pkgs/main/win-64::matplotlib-3.1.3-py37_0
matplotlib-base       pkgs/main/win-64::matplotlib-base-3.1.3-py37h64f37c6_0

Proceed ([y]/n)?
```

It will ask you for y/n, click on y to proceed further.

```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras - conda install jupyter - conda install spyder - cond..
- matplotlib

The following packages will be downloaded:

package | build
-----|-----
kiwisolver-1.2.0 | py37h74a9793_0 55 KB
-----|-----
Total: 55 KB

The following NEW packages will be INSTALLED:

cycler pkgs/main/win-64::cycler-0.10.0-py37_0
freetype pkgs/main/win-64::freetype-2.9.1-ha9979f8_1
kiwisolver pkgs/main/win-64::kiwisolver-1.2.0-py37h74a9793_0
matplotlib pkgs/main/win-64::matplotlib-3.1.3-py37_0
matplotlib-base pkgs/main/win-64::matplotlib-base-3.1.3-py37h64f37c6_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
kiwisolver-1.2.0 | 55 KB | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(deeplearning) C:\Windows\system32>
```

You can see that you have successfully installed matplotlib.

Lastly, you will be installing pandas, and again the procedure is the same.

1. conda install pandas

```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras - conda install jupyter - conda install spyder - cond..
latest version: 4.8.3
Please update conda by running

$ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\HP\anaconda3\envs\deeplearning

added / updated specs:
- pandas

The following packages will be downloaded:

package | build
-----|-----
pandas-1.0.3 | py37h47e9c7a_0 7.4 MB
-----|-----
Total: 7.4 MB

The following NEW packages will be INSTALLED:

pandas pkgs/main/win-64::pandas-1.0.3-py37h47e9c7a_0

Proceed ([y]/n)?
```

Proceed with clicking on y.

```
Administrator: Anaconda Prompt (anaconda3) - conda install -c anaconda keras - conda install jupyter - conda install spyder - cond...
environment location: C:\Users\HP\anaconda3\envs\deeplearning
added / updated specs:
- pandas

The following packages will be downloaded:

package | build | size
-----|-----|-----
pandas-1.0.3 | py37h47e9c7a_0 | 7.4 MB
Total: 7.4 MB

The following NEW packages will be INSTALLED:

pandas pkgs/main/win-64::pandas-1.0.3-py37h47e9c7a_0

Proceed ([y]/n)? y

Downloading and Extracting Packages
pandas-1.0.3 | 7.4 MB | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(deeplearning) C:\Windows\system32>
```

From the image given above, you can see that it also has been installed successfully.

Keras APIs – three programming models

Sequential model:

The core idea of **Sequential API** is simply arranging the Keras layers in a sequential order and so, it is called **Sequential API**. Most of the ANN also has layers in sequential order and the data flows from one layer to another layer in the given order until the data finally reaches the output layer.

A ANN model can be created by simply calling **Sequential()** API as specified below –

```
from keras.models import Sequential
model = Sequential()
```

Add layers

To add a layer, simply create a layer using Keras layer API and then pass the layer through add() function as specified below –

```
from keras.models import Sequential

model = Sequential()
input_layer = Dense(32, input_shape=(8,)) model.add(input_layer)
hidden_layer = Dense(64, activation='relu'); model.add(hidden_layer)
output_layer = Dense(8)
model.add(output_layer)
```

Here, we have created one input layer, one hidden layer and one output layer.

Access the model

Keras provides few methods to get the model information like layers, input data and output data. They are as follows –

- ***model.layers*** – Returns all the layers of the model as list.

```
>>> layers = model.layers
>>> layers
[
  <keras.layers.core.Dense object at 0x000002C8C888B8D0>,
  <keras.layers.core.Dense object at 0x000002C8C888B7B8>,
  <keras.layers.core.Dense object at 0x000002C8C888B898>
]
```

- ***model.inputs*** – Returns all the input tensors of the model as list.

```
>>> inputs = model.inputs
>>> inputs
[<tf.Tensor 'dense_13_input:0' shape=(?, 8) dtype=float32>]
```

- ***model.outputs*** – Returns all the output tensors of the model as list.

```
>>> outputs = model.outputs
>>> outputs
<tf.Tensor 'dense_15/BiasAdd:0' shape=(?, 8) dtype=float32>]
```

- ***model.get_weights*** – Returns all the weights as NumPy arrays.
- ***model.set_weights(weight_numpy_array)*** – Set the weights of the model.

Functional API

Sequential API is used to create models layer-by-layer. Functional API is an alternative approach of creating more complex models. Functional model, you can define multiple input or output that share layers. First, we create an instance for model and connecting to the layers to access input and output to the model. This section explains about functional model in brief.

Create a model

Import an input layer using the below module –

```
>>> from keras.layers import Input
```

Now, create an input layer specifying input dimension shape for the model using the below code –

```
>>> data = Input(shape=(2,3))
```

Define layer for the input using the below module –

```
>>> from keras.layers import Dense
```

Add Dense layer for the input using the below line of code –

```
>>> layer = Dense(2)(data)
>>> print(layer)
Tensor("dense_1/add:0", shape=(?, 2, 2), dtype = float32)
```

Define model using the below module –

```
from keras.models import Model
```

Create a model in functional way by specifying both input and output layer –

```
model = Model(inputs = data, outputs = layer)
```

The complete code to create a simple model is shown below –

```
from keras.layers import Input
from keras.models import Model
from keras.layers import Dense

data = Input(shape=(2,3))
layer = Dense(2)(data) model =
Model(inputs=data,outputs=layer) model.summary()
```

Layer (type)	Output Shape	Param #
=====		
input_2 (InputLayer)	(None, 2, 3)	0

dense_2 (Dense)	(None, 2, 2)	8
=====		
Total params: 8		
Trainable params: 8		
Non-trainable params: 0		

Model Sub-Classing:

In **Model Sub-Classing** there are two most important functions `__init__` and `call`. Basically, we will define all the trainable *tf.keras* layers or custom implemented layers inside the `__init__` method and call those layers based on our network design inside the `call` method which is used to perform a forward propagation. (It's quite the same as the *forward* method that is used to build the model in PyTorch anyway.)

```
class ModelSubClassing(tf.keras.Model):
    def __init__(self, num_classes):
        super(ModelSubClassing, self).__init__()
        # define all layers in init
        # Layer of Block 1
        self.conv1 = tf.keras.layers.Conv2D(32, 3, strides=2, activation="relu")
        self.max1 = tf.keras.layers.MaxPooling2D(3)
        self.bn1 = tf.keras.layers.BatchNormalization()

        # Layer of Block 2
        self.conv2 = tf.keras.layers.Conv2D(64, 3, activation="relu")
        self.bn2 = tf.keras.layers.BatchNormalization()
        self.drop = tf.keras.layers.Dropout(0.3)

        # GAP, followed by Classifier
        self.gap = tf.keras.layers.GlobalAveragePooling2D()
        self.dense = tf.keras.layers.Dense(num_classes)

    def call(self, input_tensor, training=False):
        # forward pass: block 1
        x = self.conv1(input_tensor)
        x = self.max1(x)
        x = self.bn1(x)

        # forward pass: block 2
        x = self.conv2(x)
        x = self.bn2(x)

        # dropout followed by gap and classifier
        x = self.drop(x)
        x = self.gap(x)
        return self.dense(x)
```

Keras Layers:

Introduction

A Keras layer requires *shape of the input (input_shape)* to understand the structure of the input data, *initializer* to set the weight for each input and finally activators to transform the output to make it non-linear. In between, constraints restricts and specify the range in which the weight of input data to be generated and regularizer will try to optimize the layer (and the model) by dynamically applying the penalties on the weights during optimization process.

To summarise, Keras layer requires below minimum details to create a complete layer.

- Shape of the input data
- Number of neurons / units in the layer
- Initializers
- Regularizers
- Constraints
- Activations

Let us understand the basic concept in the next chapter. Before understanding the basic concept, let us create a simple Keras layer using Sequential model API to get the idea of how Keras model and layer works.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers
from keras import regularizers
from keras import constraints

model = Sequential()

model.add(Dense(32, input_shape=(16,), kernel_initializer = 'he_uniform',
    kernel_regularizer = None, kernel_constraint = 'MaxNorm', activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(8))
```

where,

- **Line 1-5** imports the necessary modules.
- **Line 7** creates a new model using Sequential API.
- **Line 9** creates a new **Dense** layer and add it into the model. **Dense** is an entry level layer provided by Keras, which accepts the number of neurons or units (32) as its required parameter. If the layer is first layer, then we need to provide **Input Shape, (16,)** as well. Otherwise, the output of the previous layer will be used as input of the next layer. All other parameters are optional.
 - First parameter represents the number of units (neurons).
 - **input_shape** represent the shape of input data.
 - **kernel_initializer** represent initializer to be used. **he_uniform** function is set as value.
 - **kernel_regularizer** represent **regularizer** to be used. None is set as value.

- **kernel_constraint** represent constraint to be used. **MaxNorm** function is set as value.
 - **activation** represent activation to be used. **relu** function is set as value.
- **Line 10** creates second **Dense** layer with 16 units and set **relu** as the activation function.
- **Line 11** creates final Dense layer with 8 units.

Basic Concept of Layers

Let us understand the basic concept of layer as well as how Keras supports each concept.

Input shape

In machine learning, all type of input data like text, images or videos will be first converted into array of numbers and then feed into the algorithm. Input numbers may be single dimensional array, two dimensional array (matrix) or multi-dimensional array. We can specify the dimensional information using **shape**, a tuple of integers. For example, **(4,2)** represent matrix with four rows and two columns.

```
>>> import numpy as np
>>> shape = (4, 2)
>>> input = np.zeros(shape)
>>> print(input)
[
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
]
```

Similarly, **(3,4,2)** three dimensional matrix having three collections of 4x2 matrix (two rows and four columns).

```
>>> import numpy as np
>>> shape = (3, 4, 2)
>>> input = np.zeros(shape)
>>> print(input)
[
 [[0. 0.] [0. 0.] [0. 0.] [0. 0.]]
 [[0. 0.] [0. 0.] [0. 0.] [0. 0.]]
 [[0. 0.] [0. 0.] [0. 0.] [0. 0.]]
]
```

To create the first layer of the model (or input layer of the model), shape of the input data should be specified.

Initializers

In Machine Learning, weight will be assigned to all input data. *Initializers* module provides different functions to set these initial weight. Some of the *Keras Initializer* function are as follows –

Zeros

Generates **0** for all input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.Zeros()
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

Where, **kernel_initializer** represent the initializer for kernel of the model.

Ones

Generates **1** for all input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.Ones()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

Constant

Generates a constant value (say, **5**) specified by the user for all input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.Constant(value = 0) model.add(
    Dense(512, activation = 'relu', input_shape = (784,), kernel_initializer = my_init)
)
```

where, **value** represent the constant value

RandomNormal

Generates value using normal distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
from keras import initializers

my_init = initializers.RandomNormal(mean=0.0,
stddev = 0.05, seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
kernel_initializer = my_init))
```

where,

- **mean** represent the mean of the random values to generate
- **stddev** represent the standard deviation of the random values to generate
- **seed** represent the values to generate random number

RandomUniform

Generates value using uniform distribution of input data.

```
from keras import initializers

my_init = initializers.RandomUniform(minval = -0.05, maxval = 0.05, seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
kernel_initializer = my_init))
```

where,

- **minval** represent the lower bound of the random values to generate
- **maxval** represent the upper bound of the random values to generate

TruncatedNormal

Generates value using truncated normal distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.TruncatedNormal(mean = 0.0, stddev = 0.05, seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
kernel_initializer = my_init))
```

VarianceScaling

Generates value based on the input shape and output shape of the layer along with the specified scale.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.VarianceScaling(
scale = 1.0, mode = 'fan_in', distribution = 'normal', seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
```

```
kernel_initializer = my_init))
```

where,

- **scale** represent the scaling factor
- **mode** represent any one of **fan_in**, **fan_out** and **fan_avg** values
- **distribution** represent either of **normal** or **uniform**

VarianceScaling

It finds the **stddev** value for normal distribution using below formula and then find the weights using normal distribution,

```
stddev = sqrt(scale / n)
```

where **n** represent,

- number of input units for mode = fan_in
- number of out units for mode = fan_out
- average number of input and output units for mode = fan_avg

Similarly, it finds the *limit* for uniform distribution using below formula and then find the weights using uniform distribution,

```
limit = sqrt(3 * scale / n)
```

lecun_normal

Generates value using lecu normal distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.RandomUniform(minval = -0.05, maxval = 0.05, seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

It finds the **stddev** using the below formula and then apply normal distribution

```
stddev = sqrt(1 / fan_in)
```

where, **fan_in** represent the number of input units.

lecun_uniform

Generates value using lecu uniform distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.lecun_uniform(seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

It finds the **limit** using the below formula and then apply uniform distribution

```
limit = sqrt(3 / fan_in)
```

where,

- **fan_in** represents the number of input units
- **fan_out** represents the number of output units

glorot_normal

Generates value using glorot normal distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.glorot_normal(seed=None) model.add(
    Dense(512, activation = 'relu', input_shape = (784,), kernel_initializer = my_init)
)
```

It finds the **stddev** using the below formula and then apply normal distribution

```
stddev = sqrt(2 / (fan_in + fan_out))
```

where,

- **fan_in** represents the number of input units
- **fan_out** represents the number of output units

glorot_uniform

Generates value using glorot uniform distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.glorot_uniform(seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

It finds the **limit** using the below formula and then apply uniform distribution

```
limit = sqrt(6 / (fan_in + fan_out))
```

where,

- **fan_in** represent the number of input units.
- **fan_out** represents the number of output units

he_normal

Generates value using he normal distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
from keras import initializers

my_init = initializers.RandomUniform(minval = -0.05, maxval = 0.05, seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

It finds the *stddev* using the below formula and then apply normal distribution.

```
stddev = sqrt(2 / fan_in)
```

where, **fan_in** represent the number of input units.

he_uniform

Generates value using he uniform distribution of input data.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.he_normal(seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

It finds the *limit* using the below formula and then apply uniform distribution.

```
limit = sqrt(6 / fan_in)
```

where, **fan_in** represent the number of input units.

Orthogonal

Generates a random orthogonal matrix.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.Orthogonal(gain = 1.0, seed = None)
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_initializer = my_init))
```

where, **gain** represent the multiplication factor of the matrix.

Identity

Generates identity matrix.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.Identity(gain = 1.0) model.add(
    Dense(512, activation = 'relu', input_shape = (784,), kernel_initializer = my_init))
```

```
)
```

Constraints

In machine learning, a constraint will be set on the parameter (weight) during optimization phase. <>Constraints module provides different functions to set the constraint on the layer. Some of the constraint functions are as follows.

NonNeg

Constrains weights to be non-negative.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import initializers

my_init = initializers.Identity(gain = 1.0)
model.add(Dense(512, activation = 'relu',
input_shape = (784,)), kernel_initializer = my_init)
)
```

where, **kernel_constraint** represent the constraint to be used in the layer.

UnitNorm

Constrains weights to be unit norm.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import constraints

my_constrain = constraints.UnitNorm(axis = 0)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
kernel_constraint = my_constrain))
```

MaxNorm

Constrains weight to norm less than or equals to the given value.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import constraints

my_constrain = constraints.MaxNorm(max_value = 2, axis = 0)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
kernel_constraint = my_constrain))
```

where,

- **max_value** represent the upper bound

- *axis* represent the dimension in which the constraint to be applied. e.g. in Shape (2,3,4) axis 0 denotes first dimension, 1 denotes second dimension and 2 denotes third dimension

MinMaxNorm

Constrains weights to be norm between specified minimum and maximum values.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import constraints

my_constraint = constraints.MinMaxNorm(min_value = 0.0, max_value = 1.0, rate = 1.0, axis = 0)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_constraint = my_constraint))
```

where, **rate** represent the rate at which the weight constrain is applied.

Regularizers

In machine learning, regularizers are used in the optimization phase. It applies some penalties on the layer parameter during optimization. Keras regularization module provides below functions to set penalties on the layer. Regularization applies per-layer basis only.

L1 Regularizer

It provides L1 based regularization.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import regularizers

my_regularizer = regularizers.l1(0.)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_regularizer = my_regularizer))
```

where, **kernel_regularizer** represent the rate at which the weight constrain is applied.

L2 Regularizer

It provides L2 based regularization.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import regularizers
```

```
my_regularizer = regularizers.l2(0.)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_regularizer = my_regularizer))
```

L1 and L2 Regularizer

It provides both L1 and L2 based regularization.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import regularizers

my_regularizer = regularizers.l2(0.)
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,),
    kernel_regularizer = my_regularizer))
```

Activations

In machine learning, activation function is a special function used to find whether a specific neuron is activated or not. Basically, the activation function does a nonlinear transformation of the input data and thus enable the neurons to learn better. Output of a neuron depends on the activation function.

As you recall the concept of single perceptron, the output of a perceptron (neuron) is simply the result of the activation function, which accepts the summation of all input multiplied with its corresponding weight plus overall bias, if any available.

```
result = Activation(SUMOF(input * weight) + bias)
```

So, activation function plays an important role in the successful learning of the model. Keras provides a lot of activation function in the activations module. Let us learn all the activations available in the module.

linear

Applies Linear function. Does nothing.

```
from keras.models import Sequential
from keras.layers import Activation, Dense

model = Sequential()
model.add(Dense(512, activation = 'linear', input_shape = (784,)))
```

Where, **activation** refers the activation function of the layer. It can be specified simply by the name of the function and the layer will use corresponding activators.

elu

Applies Exponential linear unit.


```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
model = Sequential()
model.add(Dense(512, activation = 'elu', input_shape = (784,)))
```

selu

Applies Scaled exponential linear unit.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
model = Sequential()
model.add(Dense(512, activation = 'selu', input_shape = (784,)))
```

relu

Applies Rectified Linear Unit.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
model = Sequential()
model.add(Dense(512, activation = 'relu', input_shape = (784,)))
```

softmax

Applies Softmax function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
model = Sequential()
model.add(Dense(512, activation = 'softmax', input_shape = (784,)))
```

softplus

Applies Softplus function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
model = Sequential()
model.add(Dense(512, activation = 'softplus', input_shape = (784,)))
```

softsign

Applies Softsign function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
```

```
model = Sequential()
model.add(Dense(512, activation = 'softsign', input_shape = (784,)))
```

tanh

Applies Hyperbolic tangent function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense
model = Sequential()
model.add(Dense(512, activation = 'tanh', input_shape = (784,)))
```

sigmoid

Applies Sigmoid function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense

model = Sequential()
model.add(Dense(512, activation = 'sigmoid', input_shape = (784,)))
```

hard_sigmoid

Applies Hard Sigmoid function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense

model = Sequential()
model.add(Dense(512, activation = 'hard_sigmoid', input_shape = (784,)))
```

exponential

Applies exponential function.

```
from keras.models import Sequential
from keras.layers import Activation, Dense

model = Sequential()
model.add(Dense(512, activation = 'exponential', input_shape = (784,)))
```

Sr.No	Layers & Description
1	Dense Layer Dense layer is the regular deeply connected neural network layer.
2	Dropout Layers Dropout is one of the important concept in the machine learning.

3	<p>Flatten Layers</p> <p>Flatten is used to flatten the input.</p>
4	<p>Reshape Layers</p> <p>Reshape is used to change the shape of the input.</p>
5	<p>Permute Layers</p> <p>Permute is also used to change the shape of the input using pattern.</p>
6	<p>RepeatVector Layers</p> <p>RepeatVector is used to repeat the input for set number, n of times.</p>
7	<p>Lambda Layers</p> <p>Lambda is used to transform the input data using an expression or function.</p>
8	<p>Convolution Layers</p> <p>Keras contains a lot of layers for creating Convolution based ANN, popularly called as <i>Convolution Neural Network (CNN)</i>.</p>
9	<p>Pooling Layer</p> <p>It is used to perform max pooling operations on temporal data.</p>
10	<p>Locally connected layer</p> <p>Locally connected layers are similar to Conv1D layer but the difference is Conv1D layer weights are shared but here weights are unshared.</p>
11	<p>Merge Layer</p> <p>It is used to merge a list of inputs.</p>

12	<p>Embedding Layer</p> <p>It performs embedding operations in input layer.</p>
----	--

Visit:

https://infyspringboard.onwingspan.com/en/app/toc/lex_auth_01330395037450240034294_share_d/contents

KNUGP

Dealing with Data:

Missing values:

Figure Out How To Handle The Missing Data

Analyze each column with missing values carefully to understand the reasons behind the missing values as it is crucial to find out the strategy for handling the missing values.

There are 2 primary ways of handling missing values:

- 1. Deleting the Missing values**
- 2. Imputing the Missing Values**

Deleting the Missing value

Generally, this approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values.

If the missing value is of the type Missing Not At Random (MNAR), then it should not be deleted.

If the missing value is of type Missing At Random (MAR) or Missing Completely At Random (MCAR) then it can be deleted.

The disadvantage of this method is one might end up deleting some useful data from the dataset.

There are 2 ways one can delete the missing values:

Deleting the entire row

If a row has many missing values then you can choose to drop the entire row.

If every row has some (column) value missing then you might end up deleting the whole data.

Imputing the Missing Value

There are different ways of replacing the missing values. You can use the python libraries Pandas and Sci-kit learn as follows:

Replacing With Arbitrary Value

If you can make an educated guess about the missing value then you can replace it with some arbitrary value using the following code.

Data Splitting:

What is data splitting?

Data splitting is when data is divided into two or more subsets. Typically, with a two-part split, one part is used to evaluate or test the data and the other to train the model.

Data splitting is an important aspect of data science, particularly for creating models based on data. This technique helps ensure the creation of [data models](#) and processes that use data models -- such as [machine learning](#) -- are accurate.

How data splitting works

In a basic two-part data split, the training data set is used to train and develop models. Training sets are commonly used to estimate different parameters or to compare different model performance.

The [testing data](#) set is used after the training is done. The training and test data are compared to check that the final model works correctly. With machine learning, data is commonly split into three or more sets. With three sets, the additional set is the dev set, which is used to change learning process parameters.

There is no set guideline or metric for how the data should be split; it may depend on the size of the original data pool or the number of predictors in a predictive model. Organizations and data modelers may choose to separate split data based on [data sampling](#) methods, such as the following three methods:

1. **Random sampling.** This data sampling method protects the data modeling process from [bias](#) toward different possible data characteristics. However, random splitting may have issues regarding the uneven distribution of data.
2. **Stratified random sampling.** This method selects data samples at random within specific parameters. It ensures the data is correctly distributed in training and test sets.
3. **Nonrandom sampling.** This approach is typically used when data modelers want the most recent data as the test set.

Keras Dense Layer:

Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.

```
output = activation(dot(input, kernel) + bias)
```

where,

- **input** represent the input data
- **kernel** represent the weight data
- **dot** represent numpy dot product of all input and its corresponding weights
- **bias** represent a biased value used in machine learning to optimize the model
- **activation** represent the activation function.

Let us consider sample input and weights as below and try to find the result –

- input as 2 x 2 matrix [[1, 2], [3, 4]]
- kernel as 2 x 2 matrix [[0.5, 0.75], [0.25, 0.5]]
- bias value as 0
- activation as **linear**. As we learned earlier, linear activation does nothing.

```
>>> import numpy as np

>>> input = [ [1, 2], [3, 4] ]
>>> kernel = [ [0.5, 0.75], [0.25, 0.5] ]
>>> result = np.dot(input, kernel)
>>> result array([[1. , 1.75], [2.5 , 4.25]])
>>>
```

result is the output and it will be passed into the next layer.

The output shape of the Dense layer will be affected by the number of neuron / units specified in the Dense layer. For example, if the input shape is **(8,)** and number of unit is 16, then the output shape is **(16,)**. All layer will have batch size as the first dimension and so, input shape will be represented by **(None, 8)** and the output shape as **(None, 16)**. Currently, batch size is None as it is not set. Batch size is usually set during training phase.

```
>>> from keras.models import Sequential
>>> from keras.layers import Activation, Dense

>>> model = Sequential()
>>> layer_1 = Dense(16, input_shape = (8,))
>>> model.add(layer_1)
>>> layer_1.input_shape
(None, 8)
>>> layer_1.output_shape
(None, 16)
>>>
```

where,

- **layer_1.input_shape** returns the input shape of the layer.

- **layer_1.output_shape** returns the output shape of the layer.

The argument supported by **Dense layer** is as follows –

- **units** represent the number of units and it affects the output layer.
- **activation** represents the activation function.
- **use_bias** represents whether the layer uses a bias vector.
- **kernel_initializer** represents the initializer to be used for kernel.
- **bias_initializer** represents the initializer to be used for the bias vector.
- **kernel_regularizer** represents the regularizer function to be applied to the kernel weights matrix.
- **bias_regularizer** represents the regularizer function to be applied to the bias vector.
- **activity_regularizer** represents the regularizer function to be applied to the output of the layer.
- **kernel_constraint** represent constraint function to be applied to the kernel weights matrix.
- **bias_constraint** represent constraint function to be applied to the bias vector.

As you have seen, there is no argument available to specify the **input_shape** of the input data. **input_shape** is a special argument, which the layer will accept only if it is designed as first layer in the model.

Building Deep Neural Network with Keras Dense Layers: And - Create a complete end to end neural network model using Keras Sequential Model and Keras Layer API

The steps of building deep learning project are as follows:

1. Load Data
2. Define Keras Model
3. Compile Keras Model
4. Fit Keras Model
5. Evaluate Keras Model
6. Tie It All Together
7. Make Predictions

1. Load Data

The first step is to define the functions and classes you intend to use in this tutorial.

You will use the NumPy library to load your dataset and two classes from the Keras library to define your model.

The imports required are listed below.

```
1# first neural network with keras tutorial
2from numpy import loadtxt
3from tensorflow.keras.models import Sequential
4from tensorflow.keras.layers import Dense
5...
```

You can now load our dataset.

In this Keras tutorial, you will use the Pima Indians onset of diabetes dataset. This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.

As such, it is a binary classification problem (onset of diabetes as 1 or not as 0). All of the input variables that describe each patient are numerical. This makes it easy to use directly with neural networks that expect numerical input and output values and is an ideal choice for our first neural network in Keras.

The dataset is available here:

Dataset CSV File ([pima-indians-diabetes.csv](#))

Dataset Details

Download the dataset and place it in your local working directory, the same location as your Python file.

Save it with the filename:

1pima-indians-diabetes.csv

Take a look inside the file; you should see rows of data like the following:

```
16,148,72,35,0,33.6,0.627,50,1
21,85,66,29,0,26.6,0.351,31,0
38,183,64,0,0,23.3,0.672,32,1
41,89,66,23,94,28.1,0.167,21,0
50,137,40,35,168,43.1,2.288,33,1
6...
```

You can now load the file as a matrix of numbers using the NumPy function `loadtxt()`.

There are eight input variables and one output variable (the last column). You will be learning a model to map rows of input variables (X) to an output variable (y), which is often summarized as $y = f(X)$.

The variables can be summarized as follows:

Input Variables (X):

Number of times pregnant

Plasma glucose concentration at 2 hours in an oral glucose tolerance test

Diastolic blood pressure (mm Hg)

Triceps skin fold thickness (mm)

2-hour serum insulin (μ U/ml)

Body mass index (weight in kg/(height in m)²)

Diabetes pedigree function

Age (years)

Output Variables (y):

Class variable (0 or 1)

Once the CSV file is loaded into memory, you can split the columns of data into input and output variables.

The data will be stored in a 2D array where the first dimension is rows and the second dimension is columns, e.g., [rows, columns].

You can split the array into two arrays by selecting subsets of columns using the standard NumPy slice operator or `:`. You can select the first eight columns from index 0 to index 7 via the slice `0:8`. We can then select the output column (the 9th variable) via index 8.

```

1...
2# load the dataset
3dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
4# split into input (X) and output (y) variables
5X = dataset[:,0:8]
6y = dataset[:,8]
7...

```

You are now ready to define your neural network model.

Note: The dataset has nine columns, and the range 0:8 will select columns from 0 to 7, stopping before index 8. If this is new to you, then you can learn more about array slicing and ranges in this post:

2. Define Keras Model

Models in Keras are defined as a sequence of layers.

We create a [Sequential model](#) and add layers one at a time until we are happy with our network architecture.

The first thing to get right is to ensure the input layer has the correct number of input features. This can be specified when creating the first layer with the **input_shape** argument and setting it to (8,) for presenting the eight input variables as a vector.

You can piece it all together by adding each layer:

- The model expects rows of data with 8 variables (the input_shape=(8,) argument).
- The first hidden layer has 12 nodes and uses the relu activation function.
- The second hidden layer has 8 nodes and uses the relu activation function.
- The output layer has one node and uses the sigmoid activation function.

```

1...
2# define the keras model
3model = Sequential()
4model.add(Dense(12, input_shape=(8,), activation='relu'))
5model.add(Dense(8, activation='relu'))
6model.add(Dense(1, activation='sigmoid'))
7...

```

Note: The most confusing thing here is that the shape of the input to the model is defined as an argument on the first hidden layer. This means that the line of code that adds the first Dense layer is doing two things, defining the input or visible layer and the first hidden layer.

3. Compile Keras Model

Now that the model is defined, *you can compile it*.

Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware, such as CPU, GPU, or even distributed.

When compiling, you must specify some additional properties required when training the network. Remember training a network means finding the best set of weights to map inputs to outputs in your dataset.

You must specify the loss function to use to evaluate a set of weights, the optimizer used to search through different weights for the network, and any optional metrics you want to collect and report during training.

In this case, use cross entropy as the **loss** argument. This loss is for a binary classification problems and is defined in Keras as “**binary_crossentropy**”.

```
...  
# compile the keras model  
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])  
...
```

4. Fit Keras Model

You have defined your model and compiled it to get ready for efficient computation.

Now it is time to execute the model on some data.

You can train or fit your model on your loaded data by calling the **fit()** function on the model.

Training occurs over epochs, and each epoch is split into batches.

- **Epoch:** One pass through all of the rows in the training dataset
- **Batch:** One or more samples considered by the model within an epoch before weights are updated

The training process will run for a fixed number of epochs (iterations) through the dataset that you must specify using the epochs argument. You must also set the number of dataset rows that are considered before the model weights are updated within each epoch, called the batch size, and set using the batch_size argument.

This problem will run for a small number of epochs (150) and use a relatively small batch size of 10.

These configurations can be chosen experimentally by trial and error. You want to train the model enough so that it learns a good (or good enough) mapping of rows of input data to the output classification. The model will always have some error, but the amount of error will level out after some point for a given model configuration. This is called model convergence.

```
1...  
2# fit the keras model on the dataset  
3model.fit(X, y, epochs=150, batch_size=10)  
4
```

5. Evaluate Keras Model

You have trained our neural network on the entire dataset, and you can evaluate the performance of the network on the same dataset.

This will only give you an idea of how well you have modeled the dataset (e.g., train accuracy), but no idea of how well the algorithm might perform on new data. This was done for simplicity, but ideally, you could separate your data into train and test datasets for training and evaluation of your model.

You can evaluate your model on your training dataset using the **evaluate()** function and pass it the same input and output used to train the model.

This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy.

The **evaluate()** function will return a list with two values. The first will be the loss of the model on the dataset, and the second will be the accuracy of the model on the dataset. You are only interested in reporting the accuracy so ignore the loss value.

```
1...
2# evaluate the keras model
3_, accuracy = model.evaluate(X, y)
4print('Accuracy: %.2f % (accuracy*100))
```

6. Tie It All Together

You have just seen how you can easily create your first neural network model in Keras.

Let's tie it all together into a complete code example.

```
1 # first neural network with keras tutorial
2 from numpy import loadtxt
3 from tensorflow.keras.models import Sequential
4 from tensorflow.keras.layers import Dense
5 # load the dataset
6 dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
7 # split into input (X) and output (y) variables
8 X = dataset[:,0:8]
9 y = dataset[:,8]
10# define the keras model
11model = Sequential()
```

```

12model.add(Dense(12, input_shape=(8,), activation='relu'))
13model.add(Dense(8, activation='relu'))
14model.add(Dense(1, activation='sigmoid'))
15# compile the keras model
16model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
17# fit the keras model on the dataset
18model.fit(X, y, epochs=150, batch_size=10)
19# evaluate the keras model
20_, accuracy = model.evaluate(X, y)
21print('Accuracy: %.2f' % (accuracy*100))

```

You can copy all the code into your Python file and save it as “**keras_first_network.py**” in the same directory as your data file “**pima-indians-diabetes.csv**“. You can then run the Python file as a script from your command line (command prompt) as follows:

```
1python keras_first_network.py
```

Running this example, you should see a message for each of the 150 epochs, printing the loss and accuracy, followed by the final evaluation of the trained model on the training dataset.

It takes about 10 seconds to execute on my workstation running on the CPU.

Ideally, you would like the loss to go to zero and the accuracy to go to 1.0 (e.g., 100%). This is not possible for any but the most trivial machine learning problems. Instead, you will always have some error in your model. The goal is to choose a model configuration and training configuration that achieve the lowest loss and highest accuracy possible for a given dataset.

```

1 ...
2 768/768 [=====] - 0s 63us/step - loss: 0.4817 - acc: 0.7708
3 Epoch 147/150
4 768/768 [=====] - 0s 63us/step - loss: 0.4764 - acc: 0.7747
5 Epoch 148/150
6 768/768 [=====] - 0s 63us/step - loss: 0.4737 - acc: 0.7682
7 Epoch 149/150
8 768/768 [=====] - 0s 64us/step - loss: 0.4730 - acc: 0.7747
9 Epoch 150/150
10768/768 [=====] - 0s 63us/step - loss: 0.4754 - acc: 0.7799
11768/768 [=====] - 0s 38us/step
12Accuracy: 76.56

```

Note: If you try running this example in an IPython or Jupyter notebook, you may get an error.

The reason is the output progress bars during training. You can easily turn these off by setting **verbose=0** in the call to the **fit()** and **evaluate()** functions; for example:

```
1...
2# fit the keras model on the dataset without progress bars
3model.fit(X, y, epochs=150, batch_size=10, verbose=0)
4# evaluate the keras model
5_, accuracy = model.evaluate(X, y, verbose=0)
6...
```

Note: Your **results may vary** given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

For example, below are the accuracy scores from re-running the example five times:

```
1Accuracy: 75.00
2Accuracy: 77.73
3Accuracy: 77.60
4Accuracy: 78.12
5Accuracy: 76.17
```

You can see that all accuracy scores are around 77%, and the average is 76.924%.

7. Making Predictions:

Making predictions is as easy as calling the `predict()` function on the model. You are using a sigmoid activation function on the output layer, so the predictions will be a probability in the range between 0 and 1. You can easily convert them into a crisp binary prediction for this classification task by rounding them.

For example:

```
1...
2# make probability predictions with the model
3predictions = model.predict(X)
4# round predictions
5rounded = [round(x[0]) for x in predictions]
```

Alternately, you can convert the probability into 0 or 1 to predict crisp classes directly; for example:

```
1...
2# make class predictions with the model
3predictions = (model.predict(X) > 0.5).astype(int)
```

The complete example below makes predictions for each example in the dataset, then prints the input data, predicted class, and expected class for the first five examples in the dataset.

```
1 # first neural network with keras make predictions
2 from numpy import loadtxt
3 from tensorflow.keras.models import Sequential
4 from tensorflow.keras.layers import Dense
5 # load the dataset
```

```

6 dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
7 # split into input (X) and output (y) variables
8 X = dataset[:,0:8]
9 y = dataset[:,8]
10 # define the keras model
11 model = Sequential()
12 model.add(Dense(12, input_shape=(8,), activation='relu'))
13 model.add(Dense(8, activation='relu'))
14 model.add(Dense(1, activation='sigmoid'))
15 # compile the keras model
16 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
17 # fit the keras model on the dataset
18 model.fit(X, y, epochs=150, batch_size=10, verbose=0)
19 # make class predictions with the model
20 predictions = (model.predict(X) > 0.5).astype(int)
21 # summarize the first 5 cases
22 for i in range(5):
23     print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))

```

Running the example does not show the progress bar as before, as the verbose argument has been set to 0.

After the model is fit, predictions are made for all examples in the dataset, and the input rows and predicted class value for the first five examples is printed and compared to the expected class value.

You can see that most rows are correctly predicted. In fact, you can expect about 76.9% of the rows to be correctly predicted based on your estimated performance of the model in the previous section.

```

1 [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 0 (expected 1)
2 [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
3 [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
4 [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
5 [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)

```


Keras Optimizers:

During the training of the model, we tune the parameters(also known as hyperparameter tuning) and weights to minimize the loss and try to make our prediction accuracy as correct as possible. Now to change these parameters the optimizer's role came in, which ties the model parameters with the loss function by updating the model in response to the loss function output. Simply optimizers shape the model into its most accurate form by playing with model weights. The loss function just tells the optimizer when it's moving in the right or wrong direction.

Optimizers are Classes or methods used to change the attributes of your machine/deep learning model such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster.

Tensorflow Keras Optimizers Classes:

Gradient descent optimizers, the year in which the papers were published, and the components they act upon

TensorFlow mainly supports 9 optimizer classes, consisting of algorithms like Adadelta, FTRL, NAdam, Adadelta, and many more.

- Adadelta: Optimizer that implements the Adadelta algorithm.
- Adagrad: Optimizer that implements the Adagrad algorithm.
- Adam: Optimizer that implements the Adam algorithm.
- Adamax: Optimizer that implements the Adamax algorithm.
- Ftrl: Optimizer that implements the FTRL algorithm.
- Nadam: Optimizer that implements the NAdam algorithm.
- Optimizer class: Base class for Keras optimizers.
- RMSprop: Optimizer that implements the RMSprop algorithm.
- SGD: Gradient descent (with momentum) optimizer.

Keras Metrics:

A metric is a **function that is used to judge the performance of your model**. Metric functions are similar to loss functions, except that the results from evaluating a metric are not used when training the model. Note that you may use any loss function as a metric.

Keras Losses:

A loss is a callable with arguments `loss_fn(y_true, y_pred, sample_weight=None)`:

- **y_true**: Ground truth values, of shape (batch_size, d0, ... dN). For sparse loss functions, such as sparse categorical crossentropy, the shape should be (batch_size, d0, ... dN-1)
- **y_pred**: The predicted values, of shape (batch_size, d0, .. dN).
- **sample_weight**: Optional sample_weight acts as reduction weighting coefficient for the per-sample losses. If a scalar is provided, then the loss is simply scaled by the given value.

If `sample_weight` is a tensor of size `[batch_size]`, then the total loss for each sample of the batch is rescaled by the corresponding element in the `sample_weight` vector. If the shape of `sample_weight` is `(batch_size, d0, ... dN-1)` (or can be broadcasted to this shape), then each loss element of `y_pred` is scaled by the corresponding value of `sample_weight`. (Note on `dN-1`: all loss functions reduce by 1 dimension, usually `axis=-1`.)

Link to Infosys Springboard Tutorial:

https://infyspringboard.onwingspan.com/en/app/toc/lex_auth_01329493558684057638929_shared/contents

Keras Callbacks:

Callbacks API

A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc).

You can use callbacks to:

- Write TensorBoard logs after every batch of training to monitor your metrics
- Periodically save your model to disk
- Do early stopping
- Get a view on internal states and statistics of a model during training
- ...and more

Usage of callbacks via the built-in `fit()` loop

You can pass a list of callbacks (as the keyword argument `callbacks`) to the `.fit()` method of a model:

```
my_callbacks =  
[ tf.keras.callbacks.EarlyStopping(patience=2),  
  tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{  
{val_loss:.2f}.h5'),  
  tf.keras.callbacks.TensorBoard(log_dir='./logs'),  
]  
model.fit(dataset, epochs=10, callbacks=my_callbacks)
```

The relevant methods of the callbacks will then be called at each stage of the training.

Commonly used callbacks:

- [Base Callback class](#)
- [ModelCheckpoint](#)
- [BackupAndRestore](#)
- [TensorBoard](#)

- [EarlyStopping](#)
- [LearningRateScheduler](#)
- [ReduceLROnPlateau](#)
- [RemoteMonitor](#)
- [LambdaCallback](#)
- [TerminateOnNaN](#)
- [CSVLogger](#)
- [ProgbarLogger](#)

TENSORBOARD:

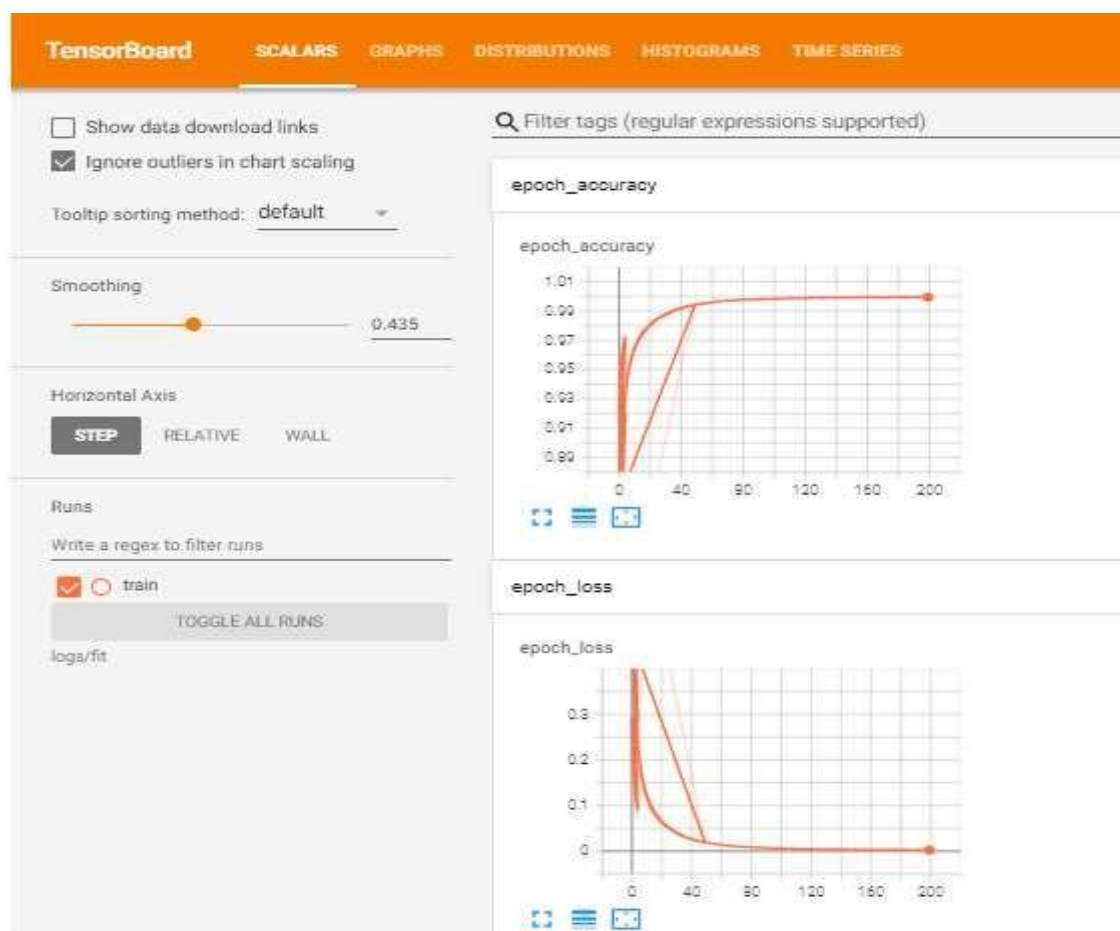
It is a visualization extension created by the TensorFlow team to decrease the complexity of neural networks. Various types of graphs can be created using it. A few of those are Accuracy, Error, weight distributions, etc.



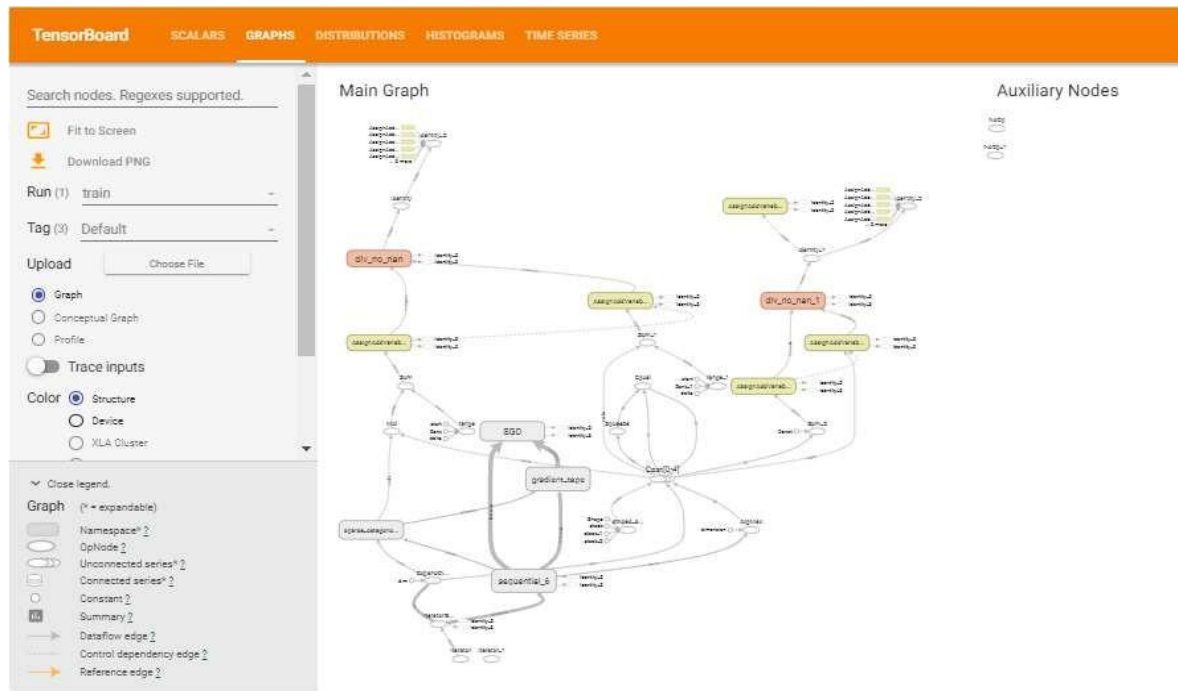
Components of Tensorboard

There are five components to it, namely:

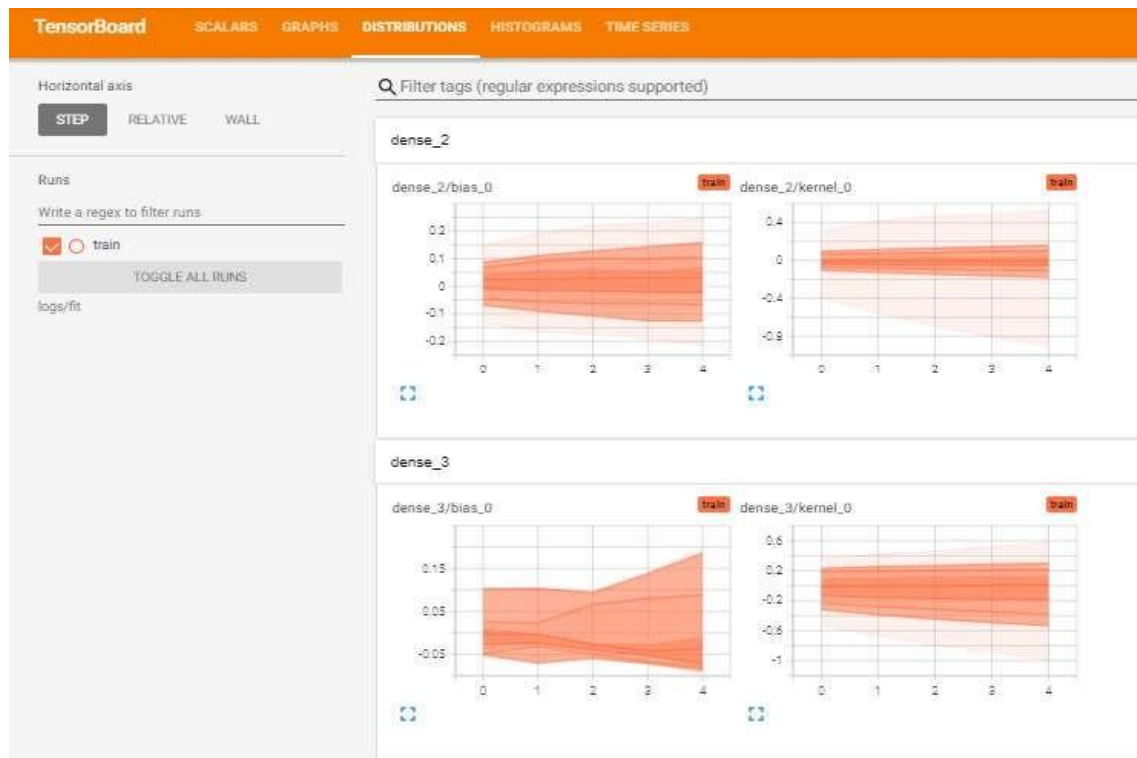
1) **Scalars:** It is very difficult to look at the accuracy and error for each epoch given the number of the epoch is very large and there are chances of ending up getting stuck in local minima instead of the global ones. These two problems can be solved using this section. It displays both accuracy and error graph wrt epoch.



2) **Graphs:** This section visualizes the “model.summary()” results. In other words, it makes the architecture of neural networks more appealing. This eases the process of architecture understanding easier.



3) **Distributions:** In general, neural networks contain a lot of layers and every layer is composed of numerous biases and weights. This section represents the distribution of these hyperparameters.



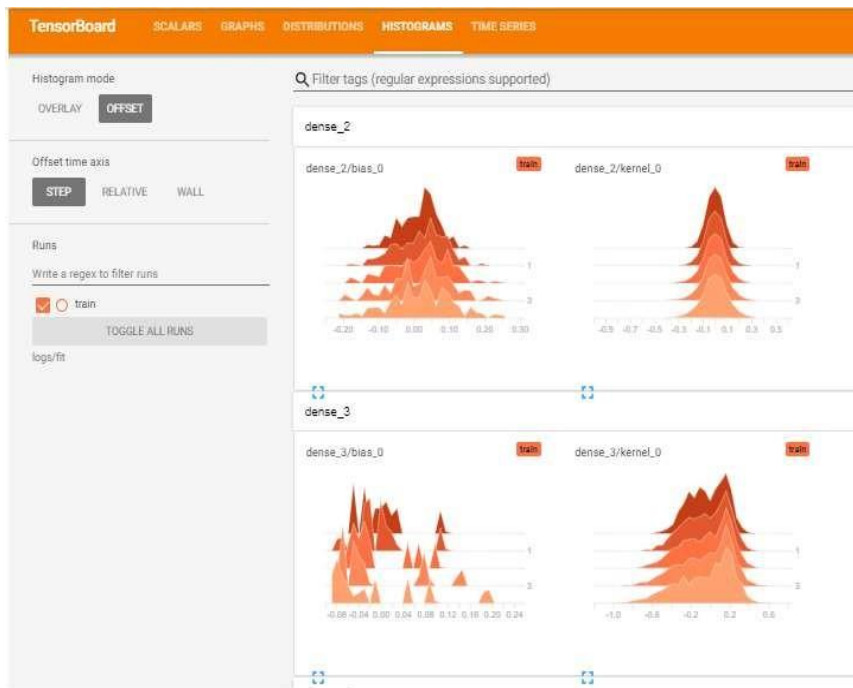
4) **Histograms:** Consists of the histogram of these hyperparameters.

Step 1: import the TensorFlow library.

```
import tensorflow as tf
```

Step 2: Load data and divide it into train and test

```
mnist = tf.keras.datasets.mnist
```



5) **Time-Series:** Consists of the over-time values for the same. These sections are useful in

Tensorboard Setup:

Code

There are two ways to publish tensorboards: localhost(powered by terminal) and within jupyter notebook. The below code looks at

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Step 3: Create model architecture

```
model =
    tf.keras.models.Sequential([ tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(56, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(10)
```

Step 4: Compile model

```
model.compile(optimizer='SGD',
              loss=loss_fn,
              metrics=['accuracy'])
```

Step 5(important): Create a callback function. A callback function periodically(epochs) saves the model and the results(error, accuracy, bias, weights, etc) of the model. This is required as the graphs are a result of these values wrt every epoch.

```
tf_callbacks = tf.keras.callbacks.TensorBoard(log_dir = "logs/fit" ,
                                              histogram_freq = 1)
```

Step 6(important): Model fitting (remember to pass callback function)

```
model.fit(x_train, y_train, epochs=200, callbacks = tf_callbacks)
```



```
Epoch 1/200
1875/1875 [=====] - 4s 2ms/step - loss: 1.3057 - accuracy: 0.5907
Epoch 2/200
1875/1875 [=====] - 4s 2ms/step - loss: 0.3766 - accuracy: 0.8884
Epoch 3/200
1875/1875 [=====] - 4s 2ms/step - loss: 0.2932 - accuracy: 0.9154
Epoch 4/200
1875/1875 [=====] - 3s 2ms/step - loss: 0.2363 - accuracy: 0.9306
Epoch 5/200
1875/1875 [=====] - 4s 2ms/step - loss: 0.2015 - accuracy: 0.9421
Epoch 6/200
1875/1875 [=====] - 3s 2ms/step - loss: 0.1821 - accuracy: 0.9458
Epoch 7/200
1875/1875 [=====] - 3s 2ms/step - loss: 0.1613 - accuracy: 0.9542
Epoch 8/200
1875/1875 [=====] - 4s 2ms/step - loss: 0.1496 - accuracy: 0.9563
Epoch 9/200
1875/1875 [=====] - 4s 2ms/step - loss: 0.1363 - accuracy: 0.9605
Epoch 10/200
1875/1875 [=====] - 4s 2ms/step - loss: 0.1250 - accuracy: 0.9633
Epoch 11/200
```

The results of these epochs are very difficult to remember.

Step 7: Create tensorboard using the following:

```
%reload_ext tensorboard
%tensorboard --logdir logs/fit
```

If this code is entered into jupyter notebook, it will produce a board there. In case of hosting it over the localhost, put this in the command prompt(without “%”) and open this link:

“https://localhost:6006”. Tensorboard gets launched on port number 6006.

TensorFlow model serving: https://www.tensorflow.org/tfx/tutorials/serving/rest_simple