TENSORFLOW 2.0

What is TensorFlow?

TensorFlow is an open-source library used to make end-to-end Machine Learning apps.

TensorFlow library is used in Numerical computation and Large-scale machine learning.

TensorFlow was developed by Google Brain Team in 2015 under (Apache Licence 2.0) for internal use but its now so comman that it has been extensively used in wide areas of AI & ML.

TensorFlow 2.0 is launched in the year 2019.

TensorFlow can also be called as a mathematical library used in training of deep learning neural networks.

For eg .like Google AI uses Tensorflow to optimize search engine user experience so that it automatically shows next word to come as we try to type something inside the search tab.

Tensorflow can be used in wide variety of programming languages like Python, JavaScript, C++ & Java.

It was designed to work on multiple CPUs or GPUs, as well as mobile operating systems in some circumstances, and it includes wrappers in Python, C++, and Java.

The most important thing to realize about TensorFlow is that, for the most part, ***the core is not written in Python***: It's written in a combination of highly-optimized C++ and CUDA (Nvidia's language for programming GPUs).

Why to use Tensorflow?

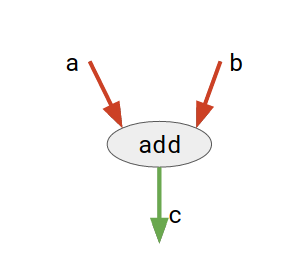
It is open source library that is free to use and develope software without any constraint.

It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Easy model building because of availibility of high-end API like Keras for debugging and iteration of model.

Robust model production lets you deploy model on any platform with no matter what language you use.

How TensorFlow Works?



TensorFlow allows developers to create graph or structure to show how data flows through graphs or processing nodes.

So each node in this graph represents mathematical operation and the connection or edges between these shows tensor that is called Tensors(Multi Dimensional Array).

TensorFlow applications can be run on most any target that’s convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs.

A trained model can be used to deliver predictions as a service via a Docker container using REST or gRPC APIs.

TensorFlow using Python

Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries.

Python just directs traffic between the pieces and provides high-level programming abstractions to hook them together.

High-level work in TensorFlow—creating nodes and layers and linking them together—uses the [Keras](https://github.com/keras-team/keras) library.

The Keras API is outwardly simple; a basic model with three layers can be defined in less than 10 lines of code, and the training code for the same takes just a few more lines of code. But if you want to "lift the hood" and do more fine-grained work, such as writing your own training loop, you can do that.

Applications of TensorFlow

1) Face Recognition

2) Self Driving Cars

3) Alexa

4) Human Robotics

& Many More

Pre-requisites / Dependencies for TensorFlow

1) Python

2) NumPy, SciPy, Pandas, Matplotlib & few more

3) Matrix - Vector, Statistics, Algebra, Calculus, Differentials

Installation of TensorFlow -

1) Create virtual enviorment

python3 -m venv env ---To create virtual env.

source env/bin/activate ---To activate virtual env.

2) pip install tensorflow (CPU installation only)

-------------------------OR----------------------------------

3) pip install tensorflow-gpu (N-vidia GPU with CUDA)

For Anaconda Nav users -

CPU Only TensorFlow :

1) conda create -n ***env\_name*** tensorflow

2) conda activate ***env\_name***

GPU only TensorFlow :

1) conda create -n ***env\_name***-gpu tensorflow-gpu

2) conda activate ***env\_name***-gpu

For CUDA version -

GPU Tensorflow uses CUDA

1) conda create -n ***env\_name-***gpu-cuda8 tensorflow-gpu cudatoolkit

2) conda activate ***env\_name-***gpu-cuda8

Importing TensorFlow

***Note - first activate the created env otherwise it will throw error.***

import tensorflow as tf

tf is usually used notation for tensorflow as standard while importing the library.

A Quick Example

**Lets see, quick example of TensorFlow using code snippet.**

*1) Import & get installed version*

# Importing tensorflow as tf

import tensorflow as tf

# To get curent version of tensorflow installed

print("TensorFlow version:", tf.\_\_version\_\_)

*2) Load a dataset*

1) Lets use MNIST dataset which is default and provided with tensorflow.

mnist = tf.keras.datasets.mnist

2) Convert the sample data from integers to floating-point numbers:

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  
x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

*3) Build a machine learning model*

Build a [tf.keras.Sequential](https://www.tensorflow.org/api_docs/python/tf/keras/Sequential) model by stacking layers.

model = tf.keras.models.Sequential([  
 tf.keras.layers.Flatten(input\_shape=(28, 28)),  
 tf.keras.layers.Dense(128, activation='relu'),  
 tf.keras.layers.Dropout(0.2),  
 tf.keras.layers.Dense(10)  
])

***4) Lets Predict***

For each example, the model returns a vector of [logits](https://developers.google.com/machine-learning/glossary#logits) or [log-odds](https://developers.google.com/machine-learning/glossary#log-odds) scores, one for each class.

predictions = model(x\_train[:1]).numpy()  
predictions

array([[ 0.2760778 , -0.39324787, -0.17098302, 1.2016621 , -0.03416392, 0.5461229 , -0.7203061 , -0.41886678, -0.59480035, -0.7580608 ]], dtype=float32)

***4) Lets Convert these Logits into class Probabilities***

tf.nn.softmax(predictions).numpy()

array([[0.11960829, 0.06124588, 0.0764901 , 0.30181262, 0.08770514, 0.15668967, 0.04416083, 0.05969675, 0.05006609, 0.04252464]], dtype=float32)

***4) Lets Define Loss Function***

loss\_fn = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)

This loss is equal to the negative log probability of the true class: The loss is zero if the model is sure of the correct class.

This untrained model gives probabilities close to random (1/10 for each class), so the initial loss should be close to -tf.math.log(1/10) ~= 2.3.

loss\_fn(y\_train[:1], predictions).numpy()

1.8534881

Before you start training, configure and compile the model using Keras [Model.compile](https://www.tensorflow.org/api_docs/python/tf/keras/Model#compile). Set the [optimizer](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers) class to adam, set the loss to the loss\_fn function you defined earlier, and specify a metric to be evaluated for the model by setting the metrics parameter to accuracy.

model.compile(optimizer='adam', loss=loss\_fn, metrics=['accuracy'])

***4) Lets train & evaluate model***

Use the [Model.fit](https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit) method to adjust your model parameters and minimize the loss:

model.fit(x\_train, y\_train, epochs=5)

Epoch 1/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.2950 - accuracy: 0.9143

Epoch 2/5 1875/1875 [==============================] - 3s 2ms/step - loss: 0.1451 - accuracy: 0.9567

Epoch 3/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.1080 - accuracy: 0.9668

Epoch 4/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.0906 - accuracy: 0.9717

Epoch 5/5 1875/1875 [==============================] - 4s 2ms/step - loss: 0.0749 - accuracy: 0.9761

<keras.callbacks.History at 0x7f062c606850>

The [Model.evaluate](https://www.tensorflow.org/api_docs/python/tf/keras/Model#evaluate) method checks the models performance, usually on a "[Validation-set](https://developers.google.com/machine-learning/glossary#validation-set)" or "[Test-set](https://developers.google.com/machine-learning/glossary#test-set)".

model.evaluate(x\_test, y\_test, verbose=2)

313/313 - 1s - loss: 0.0783 - accuracy: 0.9755 - 588ms/epoch - 2ms/step

[0.07825208455324173, 0.9754999876022339]

The image classifier is now trained to ~98% accuracy on this dataset.

If you want your model to return a probability, you can wrap the trained model, and attach the softmax to it:

probability\_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])

probability\_model(x\_test[:5])

<tf.Tensor: shape=(5, 10), dtype=float32, numpy= array([[2.72807270e-08, 2.42517650e-08, 7.75602894e-06, 1.28684027e-04, 7.66215633e-11, 3.54162950e-07, 3.04894151e-14, 9.99857187e-01, 2.32766553e-08, 5.97762892e-06], [7.37396704e-08, 4.73638036e-04, 9.99523997e-01, 7.20633352e-07, 4.54133671e-17, 1.42298268e-06, 5.96959016e-09, 1.23534145e-13, 7.77225608e-08, 6.98619169e-16], [1.95462448e-07, 9.99295831e-01, 1.02249986e-04, 1.86699708e-05, 5.65737491e-06, 1.12115902e-06, 5.32719559e-06, 5.22767776e-04, 4.79981136e-05, 1.76624681e-07], [9.99649286e-01, 1.80224735e-09, 3.73612856e-05, 1.52324446e-07, 1.30824594e-06, 2.82781020e-05, 6.99703523e-05, 3.30940424e-07, 2.13184350e-07, 2.13106396e-04], [1.53770895e-06, 1.72272063e-08, 1.98980865e-06, 3.97882580e-08, 9.97192323e-01, 1.10544443e-05, 1.54713348e-06, 2.81727880e-05, 3.48721733e-06, 2.75991508e-03]], dtype=float32)>

Congratulations!

You have done your job of training model using Keras API.