TensorFlow

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.

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

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

1) Low level API - TensorFlow Core is the low level API of TensorFlow.

2) High level API - built on top of TensorFlow Core

Installing TensorFlow

import tensorflow as tf

Features of TensorFlow

1) Models can be developed easily: TensorFlow supports high-level APIs, through which Machine Learning models can be built easily using Neural Networks.

2) Complex Numeric Computations can be done: As the input dataset is huge, the mathematical computations/calculations can be done easily.

3) Consists of Machine Learning APIs: TensorFlow is rich in Machine Learning APIs that are of both low-level and high-level. Stable APIs are

available in Python and C. Presently, working on APIs for Java, JavaScript, Julia, Matlab, R, etc.

4) Easy deployment and computation using CPU, GPU: TensorFlow supports training and building models on CPU and GPU. Computations can be done on both CPU and GPU and can be compared too.

5) Contains pre-trained models and datasets: Google has included many datasets and pre-trained models in TensorFlow. Datasets include mnist, vgg_face2, ImageNet, coco etc.

1) The **module tensorflow.math** provides support for many basic mathematical operations.

Function tf.log() [alias tf.math.log] provides support for the natural logarithmic function in Tensorflow. It expects the input in form of complex numbers as $a+bi$ or floating point numbers. The input type is tensor and if the input contains more than one element, an element-wise logarithm is computed, y=\log_e x}$.

Syntax: tf.log(x, name=None) or tf.math.log(x, name=None)

Parameters:

x: A Tensor of type bfloat16, half, float32, float64, complex64 or complex128.

name (optional): The name for the operation.

Return type: A Tensor with the same size and type as that of x.

2) The module tensorflow.math provides support for many basic logical operations.

Function tf.logical_and() [alias tf.math.logical_and] provides support for the logical AND function in Tensorflow. It expects the input of bool type. The input types are tensor and if the tensors contains more than one element, an element-wise logical AND is computed, $x AND y$.

Syntax: tf.logical_and(x, y, name=None) or tf.math.logical_and(x, y, name=None)

Parameters:

x: A Tensor of type bool.

y: A Tensor of type bool.

name (optional): The name for the operation.

Return type: A Tensor of bool type with the same size as that of x or y.

3) **The module tensorflow.math provides support for many basic mathematical operations. Function tf.logical_or() [alias tf.math.logical_or] provides support for the logical OR function in Tensorflow.** It expects the input of bool type. The input types are tensor and if the tensors contains more than one element, an element-wise logical OR is computed, $x OR y$.

Syntax: tf.logical_or(x, y, name=None) or tf.math.logical_or(x, y, name=None)

Parameters:

x: A Tensor of type bool.

y: A Tensor of type bool.

name (optional): The name for the operation.

4) The module tensorflow.math provides support for many basic logical operations.

Function tf.logical_xor() [alias tf.math.logical_xor] provides support for the logical XOR function in Tensorflow. It expects the inputs of bool type. The input types are tensor and if the tensors contains more than one element, an element-wise logical XOR is computed, $x XOR y = (x \| \, y) \, \&\& \, ^^21(x \&\& y)$.

Syntax: tf.logical_xor(x, y, name=None) or tf.math.logical_xor(x, y, name=None)

Parameters:

x: A Tensor of type bool.

y: A Tensor of type bool.

name (optional): The name for the operation.

Return type: A Tensor of bool type with the same size as that of x or y.

5) The module tensorflow.math provides support for many basic logical operations.

Function tf.logical_not() [alias tf.math.logical_not or tf.Tensor.__invert__] provides support for the logical NOT function in Tensorflow. It expects the input of bool type. The input type is tensor and if the input contains more than one element, an element-wise logical NOT is computed, $ NOT x $

Syntax: tf.logical_not(x, name=None) or tf.math.logical_not(x, name=None) or tf.Tensor.__invert__(x, name=None)

Parameters:

x: A Tensor of type bool.

name (optional): The name for the operation.

Return type: A Tensor of bool type with the same size as that of x.

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.

Features of Keras

1)It is a multi backend and supports multi-platform, which helps all the encoders come together for coding.

2)Research community present for Keras works amazingly with the production community.

3)Easy to grasp all concepts.

4)It supports fast prototyping.

5)It seamlessly runs on CPU as well as GPU.

6)It provides the freedom to design any architecture, which then later is utilized as an API for the project.

7)It is really very simple to get started with.

Keras is compact, easy to learn, high-level Python library run on top of TensorFlow framework. It is made with focus of understanding deep learning techniques, such as creating layers for neural networks maintaining the concepts of shapes and mathematical details.

The creation of freamework can be of the following two types –

1)Sequential API

2)Functional API

Consider the following eight steps to create deep learning model in Keras –

1)Loading the data

2)Preprocess the loaded data

3)Definition of model

4)Compiling the model

5)Fit the specified model

6)Evaluate it

7)Make the required predictions

8)Save the model

Advantages of Keras

1. It is very easy to understand and incorporate the faster deployment of network models.

2. It has huge community support in the market as most of the AI companies are keen on using it.

3. It supports multi backend, which means you can use any one of them among TensorFlow, CNTK, and Theano with Keras as a backend according to your requirement.

4. Since it has an easy deployment, it also holds support for cross-platform. Following are the devices on which Keras can be deployed:

1)iOS with CoreML

2)Android with TensorFlow Android

3)Web browser with .js support

4)Cloud engine

5)Raspberry pi

5. It supports Data parallelism, which means Keras can be trained on multiple GPU's at an instance for speeding up the training time and processing a huge amount of data.

Keras Installation Steps

Step 1: Create virtual environment

Windows

Windows user can use the below command,

py -m venv keras

Step 2: Activate the environment

This step will configure python and pip executables in your shell path.

Windows

Windows users move inside the “kerasenv” folder and type the below command,

.\env\Scripts\activate

Step 3: Python libraries

Keras depends on the following python libraries.

1)Numpy

2)Pandas

3)Scikit-learn

4)Matplotlib

5)Scipy

6)Seaborn

Keras Installation Using Python

pip install keras

summary method

Prints a string summary of the network.

get_layer method

Model.get_layer(name=None, index=None)

Retrieves a layer based on either its name (unique) or index.

If name and index are both provided, index will take precedence. Indices are based on order of horizontal graph traversal (bottom-up).

Arguments

name: String, name of layer.

index: Integer, index of layer.

Returns

A layer instance.

add method

Sequential.add(layer)

Adds a layer instance on top of the layer stack.

Arguments

layer: layer instance.

Raises

TypeError: If layer is not a layer instance.

ValueError: In case the layer argument does not know its input shape.

ValueError: In case the layer argument has multiple output tensors, or is already connected somewhere else (forbidden in Sequential models).

pop method

Sequential.pop()

Removes the last layer in the model.

Raises

TypeError: if there are no layers in the model.

compile method

Model.compile(

optimizer="rmsprop",

```
    loss=None,
    metrics=None,
    loss_weights=None,
    weighted_metrics=None,
    run_eagerly=None,
    steps_per_execution=None,
    jit_compile=None,
    pss_evaluation_shards=0,
    **kwargs
)
```

Configures the model for training.

fit method

```
Model.fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose="auto",
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
```

```
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
)
```

Trains the model for a fixed number of epochs (dataset iterations).

evaluate method

```
Model.evaluate(
    x=None,
    y=None,
    batch_size=None,
    verbose="auto",
    sample_weight=None,
    steps=None,
    callbacks=None,
    max_queue_size=10,
```

workers=1,

use_multiprocessing=False,

return_dict=False,

**kwargs

)

Returns the loss value & metrics values for the model in test mode.

Computation is done in batches (see the batch_size arg.)

predict method

Model.predict(

x,

batch_size=None,

verbose="auto",

steps=None,

callbacks=None,

max_queue_size=10,

workers=1,

use_multiprocessing=False,

)

Generates output predictions for the input samples.

train_on_batch method

Model.train_on_batch(

x,

y=None,

sample_weight=None,

class_weight=None,

reset_metrics=True,

return_dict=False,

)

Runs a single gradient update on a single batch of data.

test_on_batch method

Model.test_on_batch(

x, y=None, sample_weight=None, reset_metrics=True, return_dict=False

)

Test the model on a single batch of samples.

predict_on_batch method

Model.predict_on_batch(x)

Returns predictions for a single batch of samples.