TensorFlow Introduction

TensorFlow, at its heart, is a library for dataflow programming. It leverages various optimization techniques to make the calculation of mathematical expressions easier and more performant.

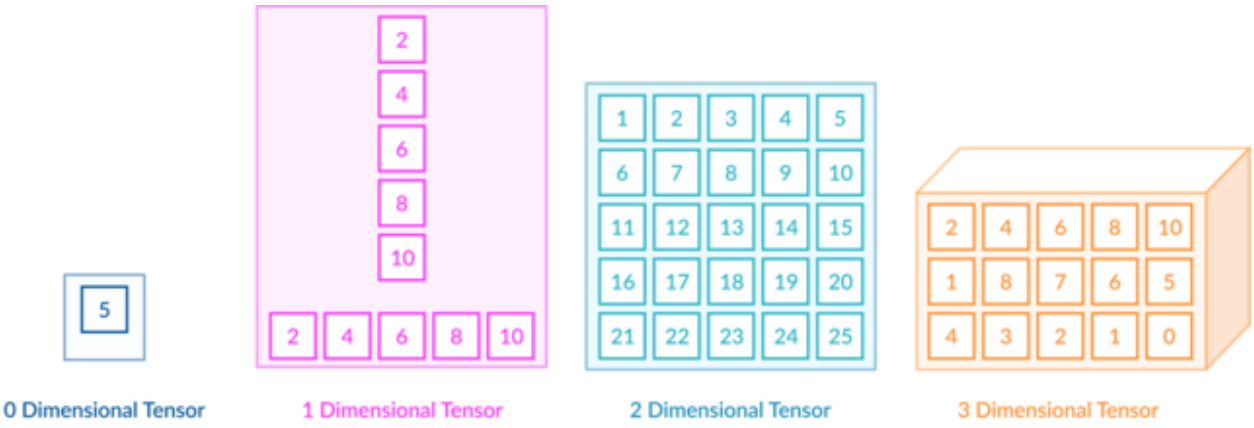
Some of the key features of TensorFlow are:

- Efficiently works with mathematical expressions involving multi-dimensional arrays
- Good support of deep neural networks and machine learning concepts
- GPU/CPU computing where the same code can be executed on both architectures
- High scalability of computation across machines and huge data sets

operation and special edges, which are used to control dependency between two nodes to set the order of operation where one node waits for another to finish.

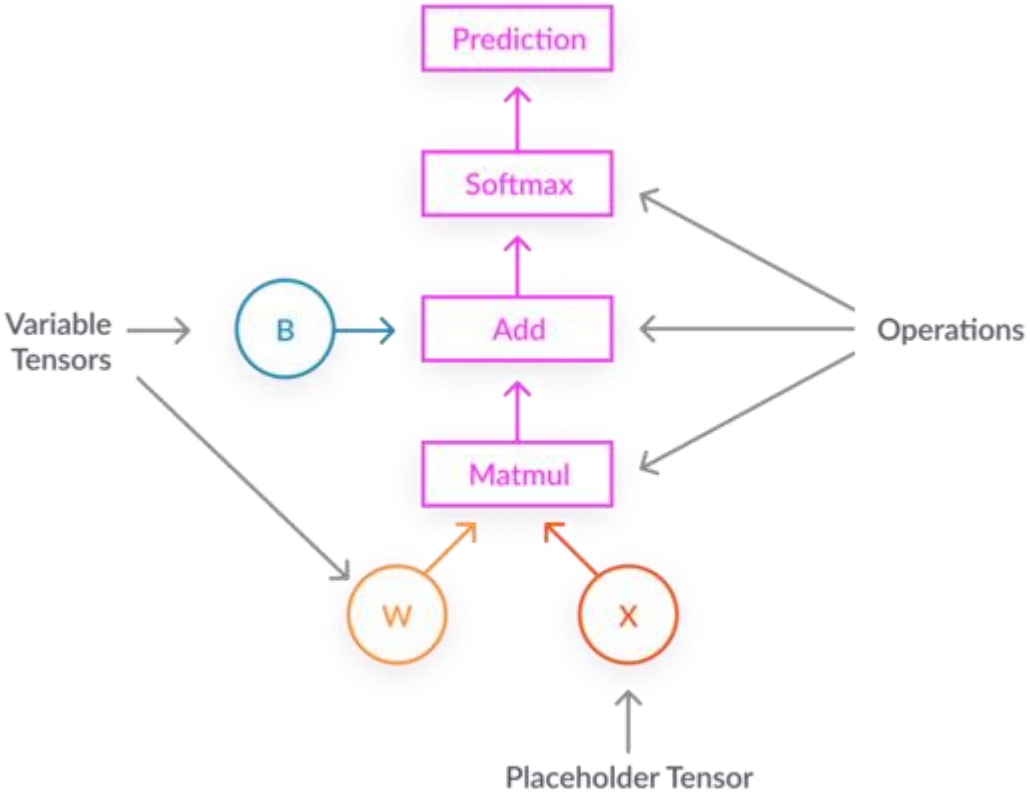
What is a tensor?

A tensor is a way to represent deep learning data. It is a multidimensional array, used to store data for multiple features of a dataset, where each feature represents an additional dimension. For example, a 3-dimensional tensor is a “cube” storing values along three axes.



Computational Graphs

In TensorFlow, computation is described using data flow graphs. Each node of the graph represents an instance of a mathematical operation (like addition, division, or multiplication) and each edge is a multi-dimensional data set (tensor) on which the operations are performed.



COMPUTATIONAL GRAPH

As TensorFlow works with computational graphs, they are managed where each node represents the instantiation of an operation where each operation has zero or more inputs and zero or more outputs.

Edges in TensorFlow can be grouped in two categories: Normal edges transfer data structure (tensors) where it is possible that the output of one operation becomes the input for another

Tensor flow Basics

Constants

In TensorFlow, constants are created using the function constant, which has the signature `constant(value, dtype=None, shape=None, name='Const', verify_shape=False)`, where `value` is an actual constant value which will be used in further computation, `dtype` is the data type parameter (e.g., float32/64, int8/16, etc.), `shape` is optional dimensions, `name` is an

optional name for the tensor, and the last parameter is a boolean which indicates verification of the shape of values.

If you need constants with specific values inside your training model, then the `constant` object can be used as in following example:

```
z = tf.constant(5.2, name="x", dtype=tf.float32)
```

Variables

Variables

Variables in TensorFlow are in-memory buffers containing tensors which have to be explicitly initialized and used in-graph to maintain state across session. By simply calling the constructor the variable is added in computational graph.

Variables are especially useful once you start with training models, and they are used to hold and update parameters. An initial value passed as an argument of a constructor represents a tensor or object which can be converted or returned as a tensor. That means if we want to fill a variable with some predefined or random values to be used afterwards in the training process and updated over iterations, we can define it in the following way:

```
k = tf.Variable(tf.zeros([1]), name="k")
```

Another way to use variables in TensorFlow is in calculations where that variable isn't trainable and can be defined in the following way:

```
k = tf.Variable(tf.add(a, b), trainable=False)
```

Placeholder

A placeholder is TensorFlow's way of allowing developers to inject data into the computation graph through placeholders which are bound inside some expressions. The signature of the placeholder is:

```
placeholder(dtype, shape=None, name=None)
```

where dtype is the type of elements in the tensors and can provide both the shape of the tensors to be fed and the name for the operation.

If the shape isn't passed, this tensor can be fed with any shape. An important note is that the placeholder tensor has to be fed with data, otherwise, upon execution of the session and if that part is missing, the placeholder generates an error.

Sessions

In order to actually evaluate the nodes, we must run a computational graph within a session.

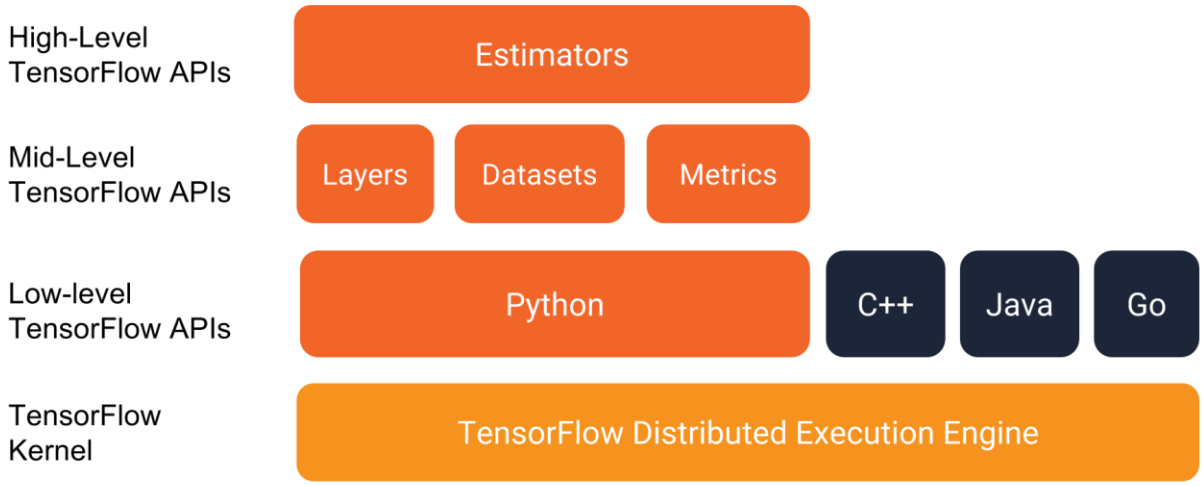
A session encapsulates the control and state of the TensorFlow runtime. A session without parameters will use the default graph created in the current session, otherwise the session class accepts a graph parameter, which is used in that session to be executed.

Below is a brief code snippet that shows how the terms defined above can be used in TensorFlow to calculate a simple linear function.

```
import tensorflow as tf

x = tf.constant(-2.0, name="x", dtype=tf.float32)
a = tf.constant(5.0, name="a", dtype=tf.float32)
b = tf.constant(13.0, name="b", dtype=tf.float32)

y = tf.Variable(tf.add(tf.multiply(a, x), b))

init = tf.global_variables_initializer()

with tf.Session() as session:
    session.run(init)
    print session.run(y)
```

TensorFlow API levels

TensorFlow lets you work directly with Tensors to build a neural network from the ground up. However, instead of using these low-level APIs which can be quite complex, TensorFlow recommends working with the higher-level Estimators API. This API enables object detection in TensorFlow, allowing you to define an object, at a higher level of abstraction, which creates and trains deep learning structures


High-Level TensorFlow APIs
Estimators
Mid-Level TensorFlow APIs
Layers
Datasets
Metrics
Low-level TensorFlow APIs
Python
C++
Java
Go
TensorFlow Kernel
TensorFlow Distributed Execution Engine