TensorFlow

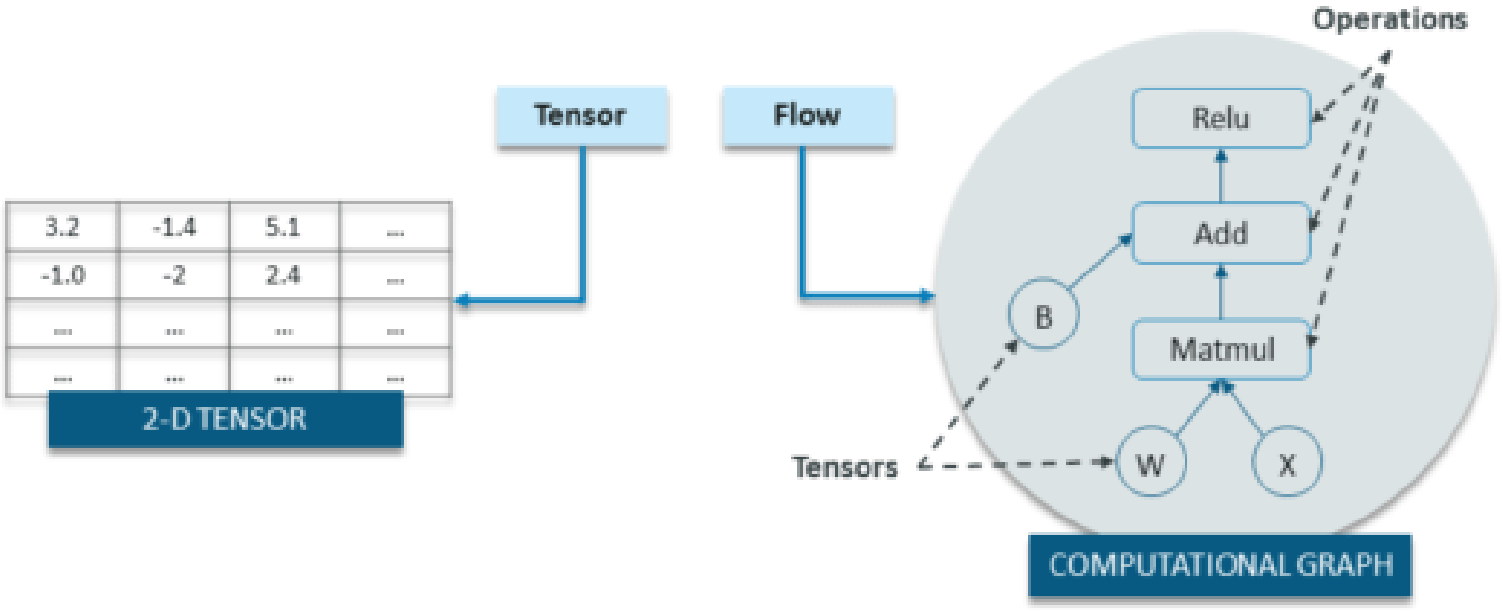
TensorFlow

An interface for expressing machine learning algorithms

and an implementation for executing such algorithms

A framework for creating ensemble algorithms for today's most challenging problems

- **What is TensorFlow?**
- TensorFlow is a library based on Python that provides different types of functionality for implementing Deep Learning Models.
- the term TensorFlow is made up of two terms – Tensor & Flow:

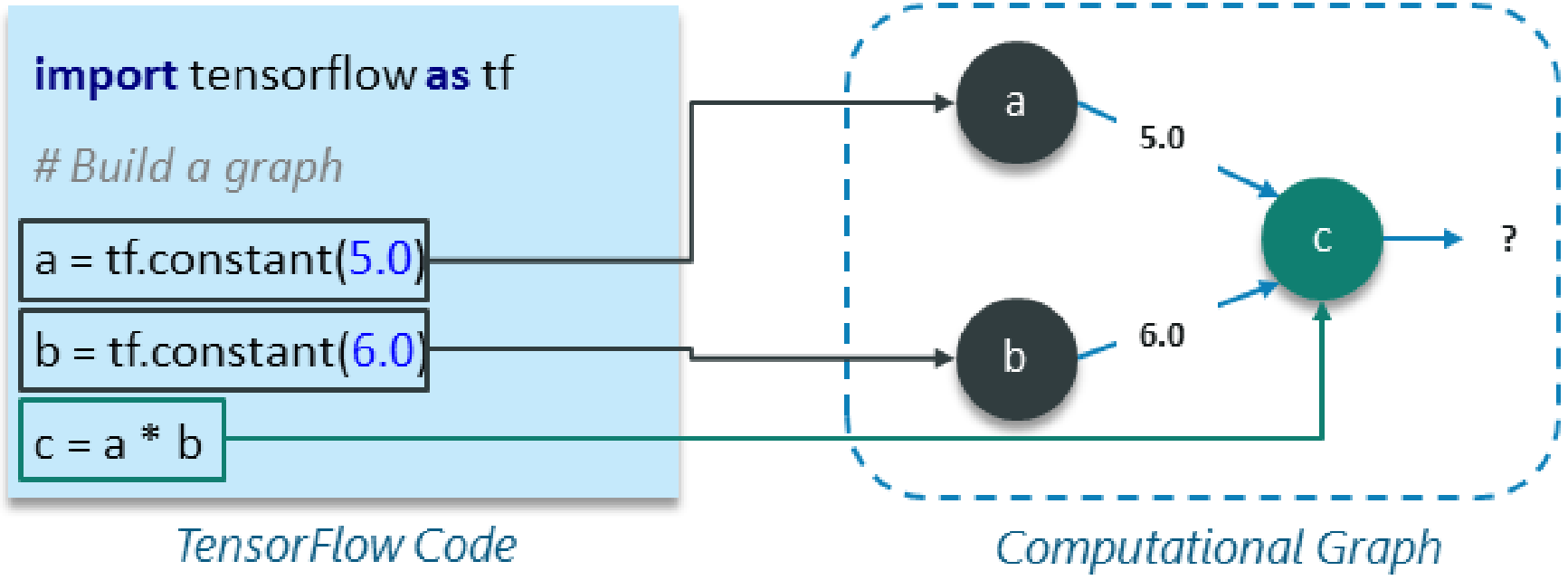


TensorFlow Tutorial: Code Basics

- Basically, the overall process of writing a TensorFlow program involves two steps:
- Building a Computational Graph
- Running a Computational Graph

1. Building a Computational Graph

- So, *what is a computational graph?* Well, a computational graph is a series of TensorFlow operations arranged as nodes in the graph. Each nodes take 0 or more tensors as input and produces a tensor as output. Let me give you an example of a simple computational graph which consists of three nodes – ***a***, ***b*** & ***c*** as shown below:



2. Running a Computational Graph

- https://www.edureka.co/blog/tensorflow-tutorial/

Tensor

An n-dimensional array or list used in Tensor to represent all data.

Tensor

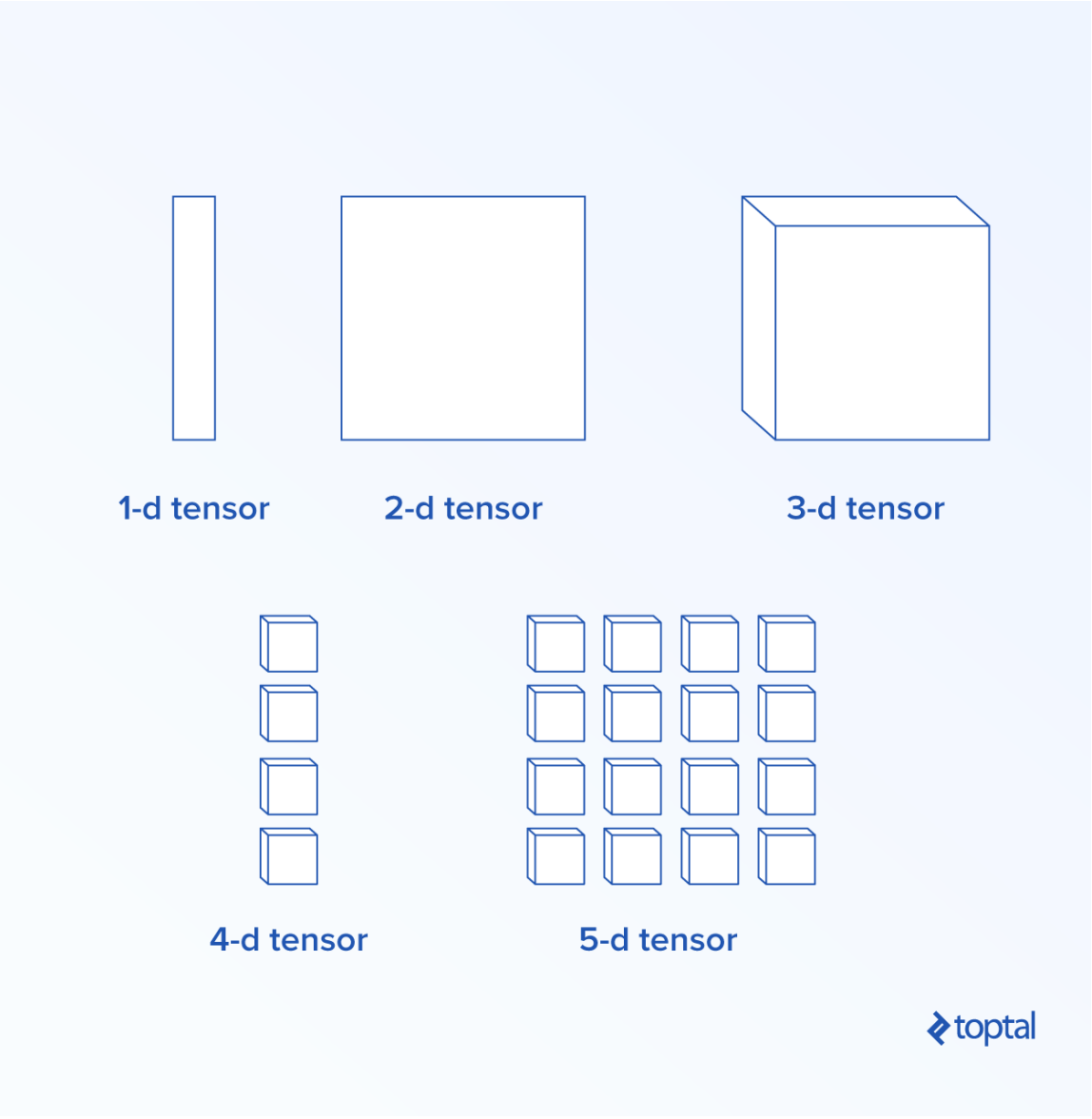
An n-dimensional array or list used in Tensor to represent all data.

Defined by the properties, Rank, Shape, and Type.

- A unique label (name)
- A dimension (shape)
- A data type (dtype)

- Mathematics with TensorFlow
- A tensor simply identifies a multidimensional array or list.
- The tensor structure can be identified with three parameters: **rank, shape, and type.**
- Rank: Identifies the number of dimensions of the tensor. A rank is known as the order or n-dimensions of a tensor, where for example rank 1 tensor is a vector or rank 2 tensor is matrix.
- Shape: The shape of a tensor is the number of rows and columns it has.
- Type: The data type assigned to tensor elements.

- To build a tensor in TensorFlow, we can build an n-dimensional array. This can be done easily by using the NumPy library, or by converting a Python n-dimensional array into a TensorFlow tensor.



Rank

Dimensionality of a Tensor.

Rank	Description	Example
0	Scalar	s = 145
1	Vector	v = [1, 3, 2, 5, 7]
2	Matrix	m = [[1,5,6], [5,3,4]]
3	3-Tensor (cube)	c = [[[1,5,6], [5,3,4]], [[9,3,5], [3,4,9]], [[4,3,2], [3,6,7]]]

Shape

Shape of data in Tensor. Related to Rank.

Rank	Description	Example	Shape
0	Scalar	s = 145	[]
1	Vector	v = [1, 3, 2, 5, 7]	[5]
2	Matrix	m = [[1,5,6], [5,3,4]]	[2,3]
3	3-Tensor (cube)	c = [[[1,5,6], [5,3,4]], [[9,3,5], [3,4,9]], [[4,3,2], [3,6,7]]]	[3,2,3]

DataType

float32, float64

int8, int16, int32, int64

uint8, uint16

string

bool

complex64, complex128

qint8, qint16, quint8

Quantitized values

Scaled to reduce size

Processed faster

TensorFlow Processing Units (TPUs) utilize quantitized values

- Create a tensor of n-dimension
- To create a tensor, you can use tf.constant()

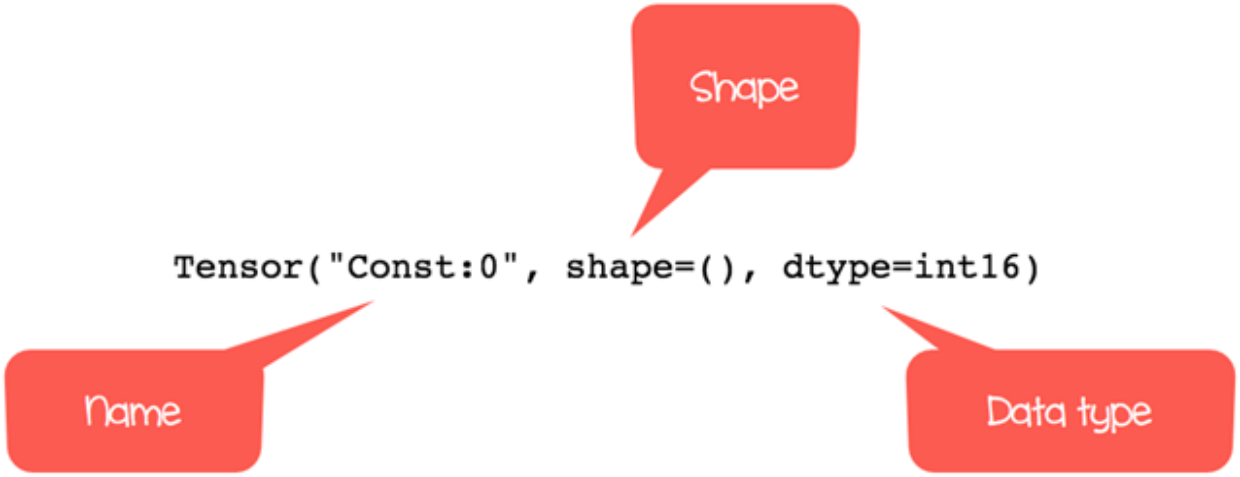
```
tf.constant(value, dtype, name = "")
arguments

- `value`: Value of n dimension to define the tensor. Optional
- `dtype`: Define the type of data:
   - `tf.string`: String variable
   - `tf.float32`: Flot variable
   - `tf.int16`: Integer variable
- "name": Name of the tensor. Optional. By default, `Const_1:0`
```

- To create a tensor of dimension 0, run the following code
- ## rank 0
- # Default name
- r1 = tf.constant(1, tf.int16)
- print(r1)

Output

Tensor("Const:0", shape=(), dtype=int16)



Each tensor is displayed by the tensor name. Each tensor object is defined with a unique label (name), a dimension (shape) and a data type (dtype).

- To build a 1-d tensor, we will use a NumPy array, which we'll construct by passing a built-in Python list.
- import numpy as np
- tensor_1d = np.array([1.45, -1, 0.2, 102.1])
-
-

```
> > print tensor1d
[   1.45   -1.      0.2   102.1 ]

> > print tensor1d[0]
1.45

> > print tensor1d[2]
0.2

> > print tensor1d.ndim
1

> > print tensor1d.shape
(4,)

> > print tensor1d.dtype
float64
```

- A NumPy array can be easily converted into a TensorFlow tensor with the auxiliary function convert_to_tensor, which helps developers convert Python objects to tensor objects.
- This function accepts tensor objects, NumPy arrays, Python lists, and Python scalars.
- tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64)
- Now if we bind our tensor to the TensorFlow session, we will be able to see the results of our conversion.
- tensor = tf.convert_to_tensor(tensor_1d, dtype=tf.float64)
-
- with tf.Session() as session:
- print session.run(tensor)
- print session.run(tensor[0])
- print session.run(tensor[1])

We can create a 2-d tensor, or matrix, in a similar way:

- tensor_2d = np.array(np.random.rand(4, 4), dtype='float32')
- tensor_2d_1 = np.array(np.random.rand(4, 4), dtype='float32')
- tensor_2d_2 = np.array(np.random.rand(4, 4), dtype='float32')
-
- m1 = tf.convert_to_tensor(tensor_2d)
- m2 = tf.convert_to_tensor(tensor_2d_1)
- m3 = tf.convert_to_tensor(tensor_2d_2)
- mat_product = tf.matmul(m1, m2)
- mat_sum = tf.add(m2, m3)
- mat_det = tf.matrix_determinant(m3)
-
- with tf.Session() as session:
- print session.run(mat_product)
- print session.run(mat_sum)
- print session.run(mat_det)

Tensor Operations

tf.square	x*x
tf.round	round(x)
tf.sqrt	sqrt(x)
tf.pow	x^y
tf.exp	e^x
tf.log	log(x)
tf.maximum	max(x, y)
tf.minimum	min(x, y)
tf.cos	cos(x)

TensorFlow operator	Description
tf.add	x+y
tf.subtract	x-y
tf.multiply	x*y
tf.div	x/y
tf.mod	x % y
tf.abs	|x|
tf.negative	-x
tf.sign	sign(x)

Methods

get_shape() – returns shape

reshape() – changes shape

rank – returns rank

dtype – return data type

cast – change data type

- Each operation you will do with TensorFlow involves the manipulation of a tensor. There are four main tensors you can create:
- tf.Variable
- tf.constant
- tf.placeholder
- tf.SparseTensor

- Variables
- So far, you have only created constant tensors. It is not of great use. Data always arrive with different values, to capture this, you can use the Variable class. It will represent a node where the values always change.
- To create a variable, you can use tf.get_variable() method

```
tf.get_variable(name = "", values, dtype, initializer)
argument
- `name = ""`: Name of the variable
- `values`: Dimension of the tensor
- `dtype`: Type of data. Optional
- `initializer`: How to initialize the tensor. Optional
If initializer is specified, there is no need to include the `values` as the shape of `initializer` is used.
```

- Placeholder
- A placeholder has the purpose of feeding the tensor. Placeholder is used to initialize the data to flow inside the tensors. To supply a placeholder, you need to use the method feed_dict. The placeholder will be fed only within a session.

```
tf.placeholder(dtype,shape=None,name=None )
arguments:
- `dtype`: Type of data
- `shape`: dimension of the placeholder. Optional. By default, shape of the data
- `name`: Name of the placeholder. Optional
data_placeholder_a = tf.placeholder(tf.float32, name = "data_placeholder_a")
print(data_placeholder_a)
```

- Session
- TensorFlow works around 3 main components:
- Graph
- Tensor
- Session

Components	Descritption
Graph	The graph is fundamental in TensorFlow. All of the mathematical operations (ops) are performed inside a graph. You can imagine a graph as a project where every operations are done. The nodes represent these ops, they can absorb or create new tensors.
Tensor	A tensor represents the data that progress between operations. You saw previously how to initialize a tensor. The difference between a constant and variable is the initial values of a variable will change over time.
Session	A session will execute the operation from the graph. To feed the graph with the values of a tensor, you need to open a session. Inside a session, you must run an operator to create an output.

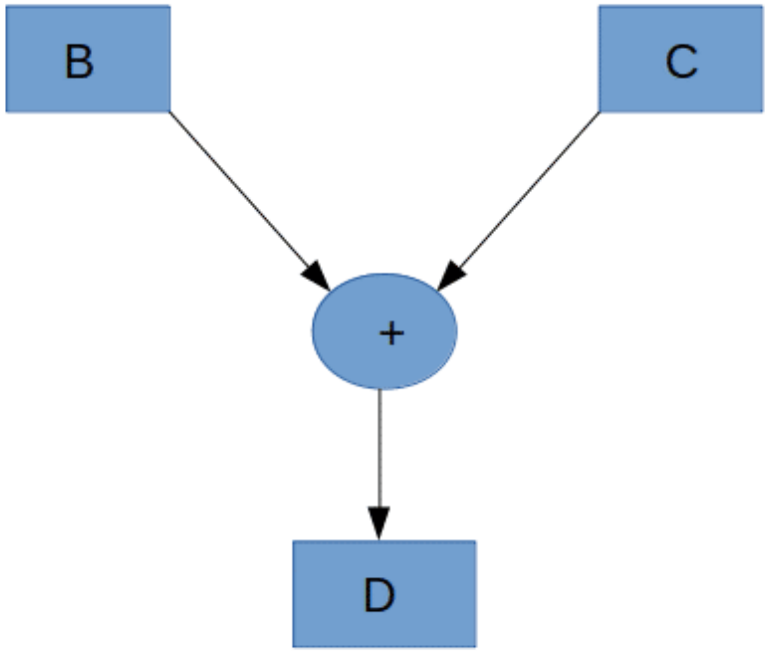
Open a session

Session	**object**
Create a session	tf.Session()
Run a session	tf.Session.run()
Evaluate a tensor	variable_name.eval()
Close a session	sess.close()
Session by block	with tf.Session() as sess:

Computational graphs

- A computational graph is a data flow graph which tensorflow internally represents while performing an operation.
- This directed graph has a set of nodes, each of which represents an operation and set of directed arcs, each representing the data on which operations are performed.
- It has two types of edges one called Normal, which are only Carriers of data structure between the nodes and the second one is called Special which does not carry any value but shows the controlled dependency between two nodes.

In this graph, we can see two arcs which are to be computed and then an arc where the value computed will be assigned, and also between them is the node where the operation is represented and connecting these all nodes and arcs are the normal edges.
Need of computational graph is that, tensorflow schedules the most efficient way while evaluating the complex expressions in learning phase which distributes the computational load.



TensorFlow is a library for numerical computation where data flows through the graph. Data in TensorFlow is represented by n-dimensional arrays called Tensors. Graph is made of data(Tensors) and mathematical operations.

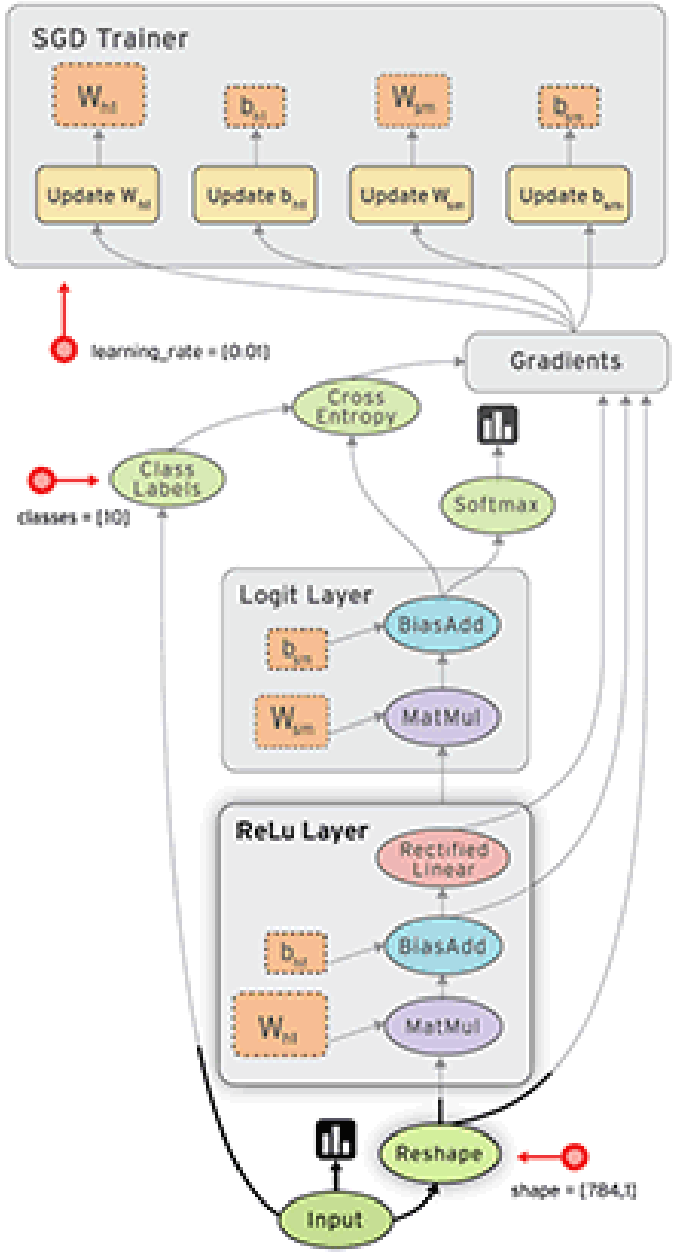
Nodes on the graph: represent mathematical operations.

Edges on the graph: represent the Tensors that flow between operations.

There is one more aspect in which TensorFlow is very different from any other programming language. In TensorFlow, you first need to create a blueprint of whatever you want to create. While you are creating the graph, variables don't have any value. **Later when you have created the complete graph, you have to run it inside a session**, **only then the variables have any values**. More on this later.

Graph in TensorFlow:

- Graph is the backbone of TensorFlow and everycomputation/operation/variables reside on the graph. Everything that happens in the code, resides on a default graph provided by TensorFlow.
- You can access this graph by:
- graph = tf.get_default_graph()
- You can get the list of all the operations by typing this:
- graph.get_operations()



https://adventuresinmachinelearning.com/python-tensorflow-tutorial/

- **TensorFlow Session**:
- A graph is used to define operations, but the operations are only run within a session. Graphs and sessions are created independently of each other. You can imagine graph to be similar to a blueprint, and a session to be similar to a construction site.
- Graph only defines the computations or builds the blueprint. However, there are no variables, no values unless we run the graph or part of the graph within a session.
- You can create a session like this:
- sess=tf.Session()
- ... your code ...
- ... your code ...
- sess.close()

- Whenever, you open a session, you need to remember to close it. **Or you can use ‘with block’ like this.**
- with tf.Session() as sess:
- sess.run(f)

- Advantage of with block is: session closes automatically at the end of the with block. We use with block in most of our code and recommend you to do so too.