# TensorFlow Tutorials

Kishan KC
Oct 10, 2018

Slides compiled from
CS 20: TensorFlow for Deep Learning Research
Stanford University

# What’s TensorFlow?

- Open source software library for numerical computation using data flow graphs

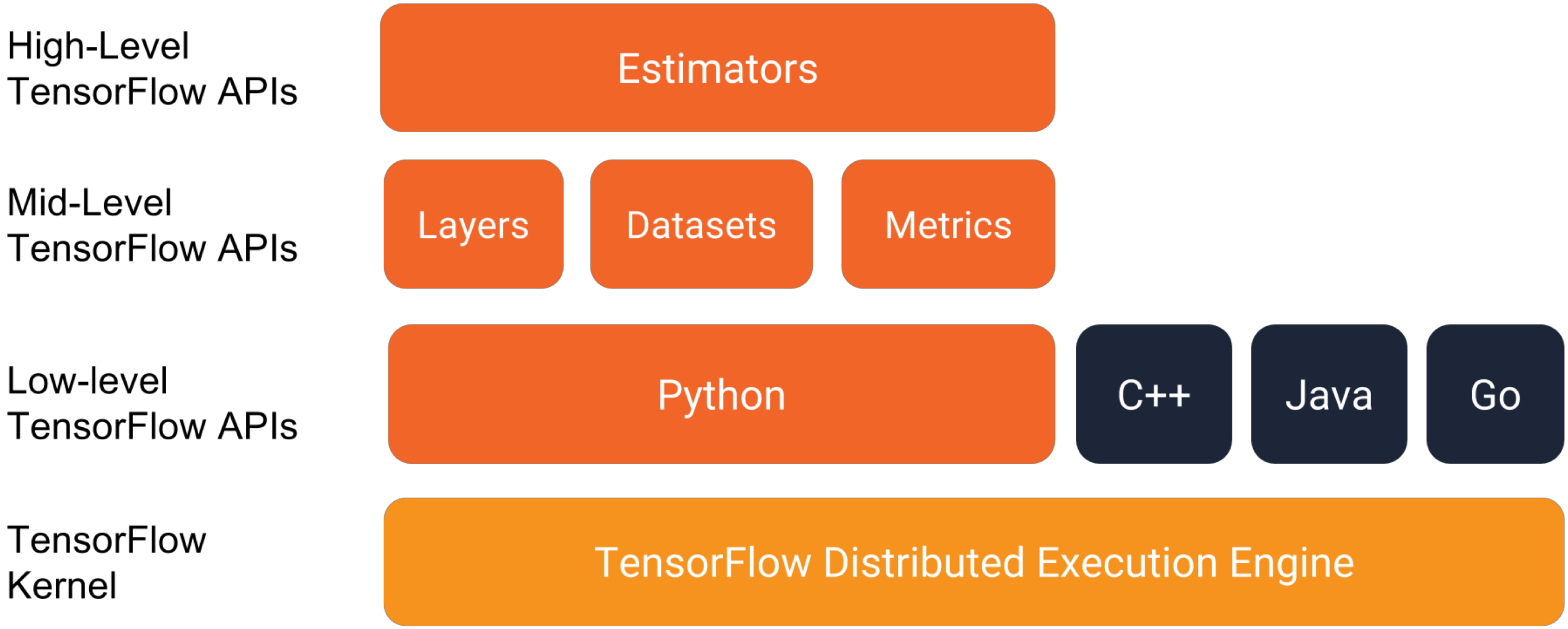
- Developed by Google Brain

- Provides primitives for defining functions on tensors and automatically computing their derivatives

# Companies using TensorFlow

From https://www.tensorflow.org

# The programming stack



# Goals

- Understand TF’s computation graph approach
- Explore TF’s built-in functions and classes
- Learn how to build and structure models best suited for a deep learning project

# Getting Started

```
import tensorflow as tf
```

# Tensor

- Generalization of vectors and matrices to higher dimensions
- TensorFlow represents tensors as n-dimensional arrays of base datatypes
- A tf.Tensor has the following properties:
  - a data type (float32, int32, or string, for example)
  - a shape
- Special tensors:
  tf.Variable, tf.constant, tf.placeholder, tf.SparseTensor

# Variables

- the best way to represent shared, persistent state manipulated by your program
- Variables are manipulated via the tf.Variable class
- Variable is basically a wrapper on tensor that maintains states across multiple calls to run

- Example:

```
# create variables with tf.Variable
s = tf.Variable(2, name="scalar")
m = tf.Variable([[0, 1], [2, 3]], name="matrix")
W = tf.Variable(tf.zeros([784,10]))

# create variables with tf.get_variable
s = tf.get_variable("scalar", initializer=tf.constant(2))
m = tf.get_variable("matrix", initializer=tf.constant([[0, 1], [2, 3]]))
W = tf.get_variable("big_matrix", shape=(784, 10), initializer=tf.zeros_initializer())
```

# Initialize your variables

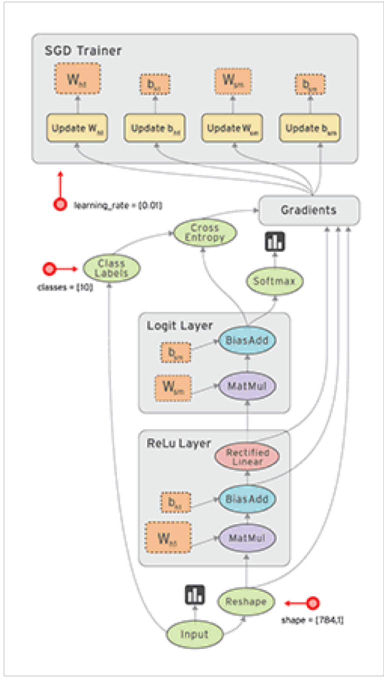
The easiest way is initializing **all variables at once**:

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
```

Initialize **only a subset** of variables:

```
with tf.Session() as sess:
    sess.run(tf.variables_initializer([a, b]))
```

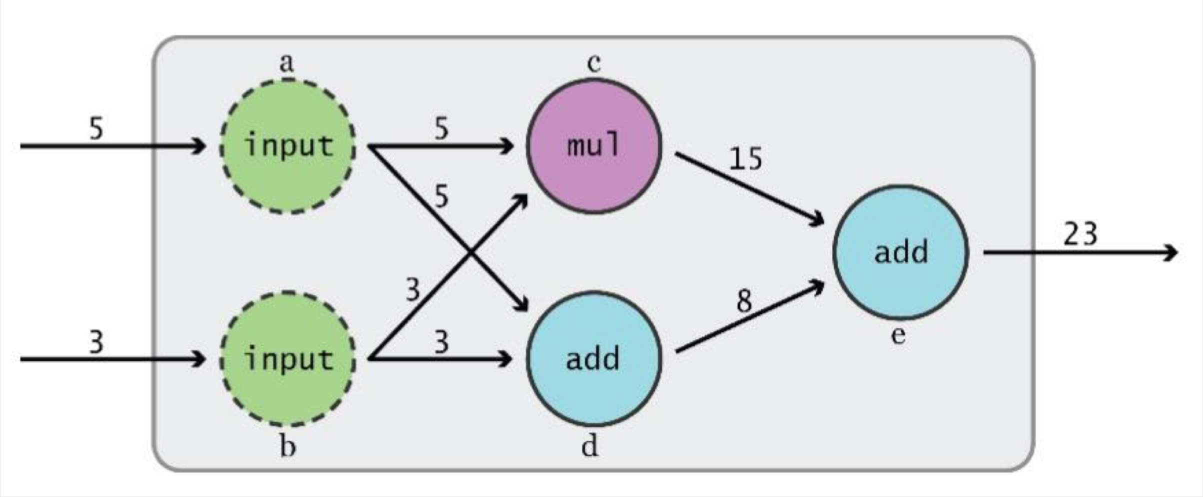
# Graphs and Sessions




From https://www.tensorflow.org

# Data Flow Graphs

- Dataflow: programming model for parallel programming
- TensorFlow separates definition of computations from their execution

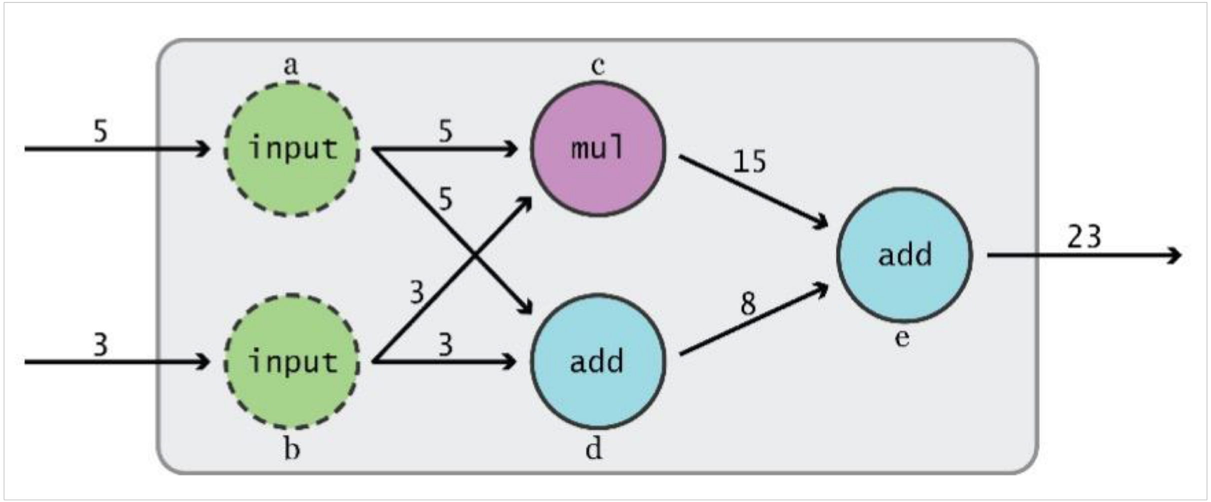



Graph from *TensorFlow for Machine Intelligence*

# Data Flow Graphs

Phase 1: assemble a graph

Phase 2: use a session to execute operations in the graph




Graph from *TensorFlow for Machine Intelligence*

# Data Flow Graphs

```
import tensorflow as tf
a = tf.add(3, 5)
print(a)
```

**What will it print?**

**Not 8, why??**

# How to get the value of a?

Create a **session**, assign it to variable sess so we can call it later

Within the session, evaluate the graph to fetch the value of **a**

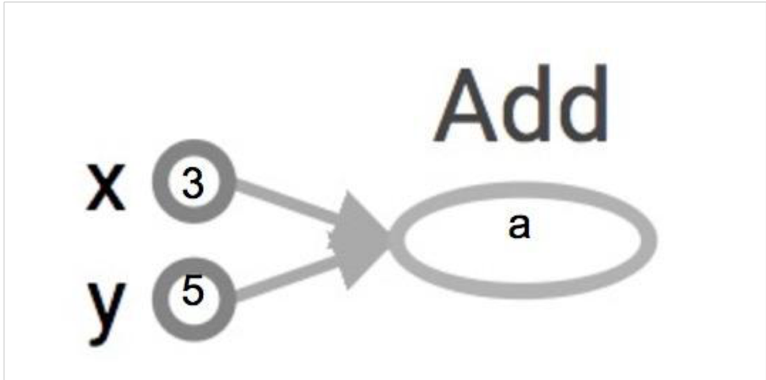
**What is session?**

# tf.Session()

- Encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated
- Allows to execute graphs or part of graphs
- Allocates resources (on one or more machines) for that and holds the actual values of intermediate results and variables

# How to get the value of a?

```
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
print(sess.run(a))
sess.close()
```

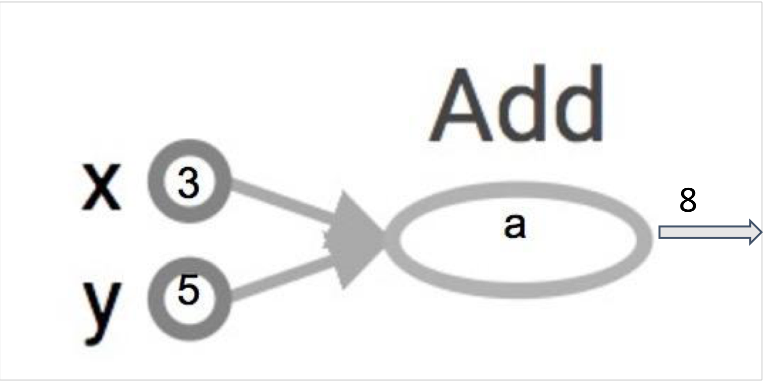



The session will look at the graph, trying to think: hmm, how can I get the value of a, then it computes all the nodes that leads to a.

# How to get the value of a?

```python
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
print(sess.run(a))
sess.close()
```

**>> 8**




Alternative approach:

```python
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
with tf.Session() as sess:
    print(sess.run(a))
sess.close()
```

The session will look at the graph, trying to think: hmm, how can I get the value of a, then it computes all the nodes that leads to a.

# Operations

| Category | Examples |
|---|---|
| Element-wise mathematical operations | Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ... |
| Array operations | Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ... |
| Matrix operations | MatMul, MatrixInverse, MatrixDeterminant, ... |
| Stateful operations | Variable, Assign, AssignAdd, ... |
| Neural network building blocks | SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ... |
| Checkpointing operations | Save, Restore |
| Queue and synchronization operations | Enqueue, Dequeue, MutexAcquire, MutexRelease, ... |
| Control flow operations | Merge, Switch, Enter, Leave, NextIteration |

# Arithmetic Ops

- `tf.abs`
- `tf.negative`
- `tf.sign`
- `tf.reciprocal`
- `tf.square`
- `tf.round`
- `tf.sqrt`
- `tf.rsqrt`
- `tf.pow`
- `tf.exp`

# TensorFlow Data Types

- `tf.float16` : 16-bit half-precision floating-point.
- `tf.float32` : 32-bit single-precision floating-point.
- `tf.float64` : 64-bit double-precision floating-point.
- `tf.bfloat16` : 16-bit truncated floating-point.
- `tf.complex64` : 64-bit single-precision complex.
- `tf.complex128` : 128-bit double-precision complex.
- `tf.int8` : 8-bit signed integer.
- `tf.uint8` : 8-bit unsigned integer.
- `tf.uint16` : 16-bit unsigned integer.
- `tf.int16` : 16-bit signed integer.
- `tf.int32` : 32-bit signed integer.
- `tf.int64` : 64-bit signed integer.
- `tf.bool` : Boolean.
- `tf.string` : String.
- `tf.qint8` : Quantized 8-bit signed integer.
- `tf.quint8` : Quantized 8-bit unsigned integer.
- `tf.qint16` : Quantized 16-bit signed integer.
- `tf.quint16` : Quantized 16-bit unsigned integer.
- `tf.qint32` : Quantized 32-bit signed integer.
- `tf.resource` : Handle to a mutable resource.

# Constants

```
import tensorflow as tf
a = tf.constant([2, 2], name='a')
b = tf.constant([[0, 1], [2, 3]], name='b')


tf.constant(
    value,
    dtype=None,
    shape=None,
    name='Const',
    verify_shape=False
)
```

# Randomly Generated Constants

- TF has several ops that create random tensors with different distributions
- the initialization of variables

- `tf.random_normal`
- `tf.truncated_normal`
- `tf.random_uniform`
- `tf.random_shuffle`
- `tf.random_crop`
- `tf.multinomial`
- `tf.random_gamma`
- `tf.set_random_seed`

# What's wrong with constants?

- Constants are stored in the graph definition

```
my_const = tf.constant([1.0, 2.0], name="my_const")
with tf.Session() as sess:
    print(sess.graph.as_graph_def())
```

```
attr {
  key: "value"
  value {
    tensor {
      dtype: DT_FLOAT
      tensor_shape {
        dim {
          size: 2
        }
      }
      tensor_content: "\000\000\200?\000\000\000@"
    }
  }
}
```

- This makes loading graphs expensive when constants are big
- Only use constants for primitive types.
- Use variables or readers for more data that requires more memory

# Placeholders

**A quick reminder:**

A TF program often has 2 phases:

1. Assemble a graph
2. Use a session to execute operations in the graph

⇒ Assemble the graph first without knowing the values needed for computation

**Analogy**:

Define the function **$f(x, y) = 2 * x + y$** without knowing value of x or y.
x, y are placeholders for the actual values.

# Why placeholders?

We, or our clients, can later supply their own data when they need to execute the computation.

# Placeholders: Example

## Syntax

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

# create a placeholder for a vector of 3 elements, type tf.float32
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # short for tf.add(a, b)

with tf.Session() as sess:
    print(sess.run(c))          # >> Error (??)
```

# Placeholders: Example

**Solution:** Supplement the values to placeholders using a dictionary

**Syntax**

```
tf.placeholder(dtype, shape=None, name=None)
# create a placeholder for a vector of 3 elements, type tf.float32
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b  # short for tf.add(a, b)

with tf.Session() as sess:
    print(sess.run(c, feed_dict={a: [1, 2, 3]}))
    # the tensor a is the key, not the string ‘a’

# >> [6, 7, 8]
```

**You can feed_dict any feedable tensor.**
**Placeholder is just a way to indicate that something must be fed**

# Feeding values to TF ops

We can evaluate TF operations individually as well.

```
# create operations, tensors, etc (using the default graph)
a = tf.add(2, 5)
b = tf.multiply(a, 3)

with tf.Session() as sess:
    # compute the value of b given a is 15
    sess.run(b, feed_dict={a: 15})              # >> 45
```

# Extremely helpful for testing

Feed in dummy values to test parts of a large graph

# Putting it together

```
DATA_FILE = 'data/birth_life_2010.txt'

# Step 1: read in data from the .txt file
data, n_samples = utils.read_birth_life_data(DATA_FILE)

# Step 2: create placeholders for X (birth rate) and Y (life expectancy)
X = tf.placeholder(tf.float32, name='X')
Y = tf.placeholder(tf.float32, name='Y')

# Step 3: create weight and bias, initialized to 0
w = tf.get_variable('weights', initializer=tf.constant(0.0))
b = tf.get_variable('bias', initializer=tf.constant(0.0))

# Step 4: build model to predict Y
Y_predicted = w * X + b

# Step 5: use the squared error as the loss function
# you can use either mean squared error or Huber loss
loss = tf.square(Y - Y_predicted, name='loss')
```

**How does TensorFlow know what variables to update?**

# Optimizers

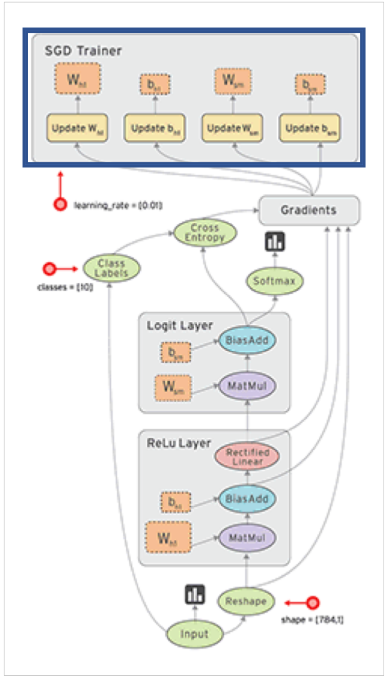
Session looks at all **trainable** variables that optimizer depends on and update them

Specify if a variable should be trained or not
By default, all variables are trainable

tf.Variable(initial_value=None, trainable=True,...)

Add a **optimizer** to the program

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(loss)
_, l = sess.run([optimizer, loss], feed_dict={X: x, Y:y})
```



# Linear Regression

**Dataset** : World Development Indicators dataset

**Target**: Find a linear relationship between X and Y to predict Y from X

**Model**:

Inference: Y_predicted = w * X + b

Mean squared error: $E[(y - y\_predicted)^2]$

**Lets execute the program**

# Logistic Regression

**Dataset** : Mnist Dataset

**Target**: Recognize the digit in the image

**Model**:

Inference: Y_predicted = softmax(X * w + b)

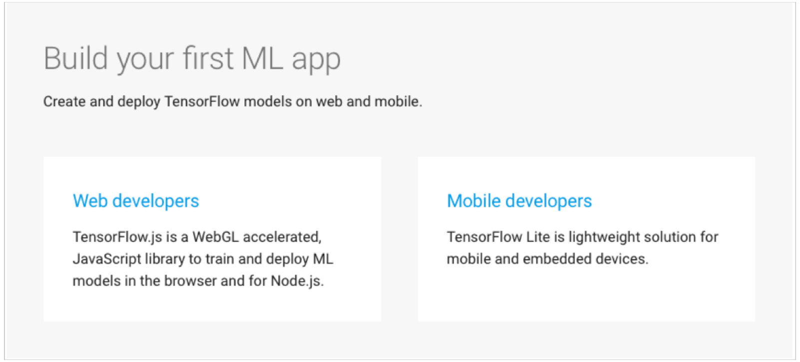
Cross entropy loss : -log(Y_predicted)

**Lets execute the program**

# Resources

- **Get Started with TensorFlow** (https://www.tensorflow.org/tutorials/)
- **Google Colab**: An easy way to learn and use TensorFlow

  - hosted Jupyter notebook environment that is free to use and requires no setup
- **CS 20: Tensorflow for Deep Learning Research** (http://web.stanford.edu/class/cs20si/)
- **Model Zoo Tensorflow** (https://modelzoo.co/framework/tensorflow)
- **For mobile and web developer**



# Questions

- Email me: kk3671@rit.edu
- Visit me in my lab 74-1050

Thanks