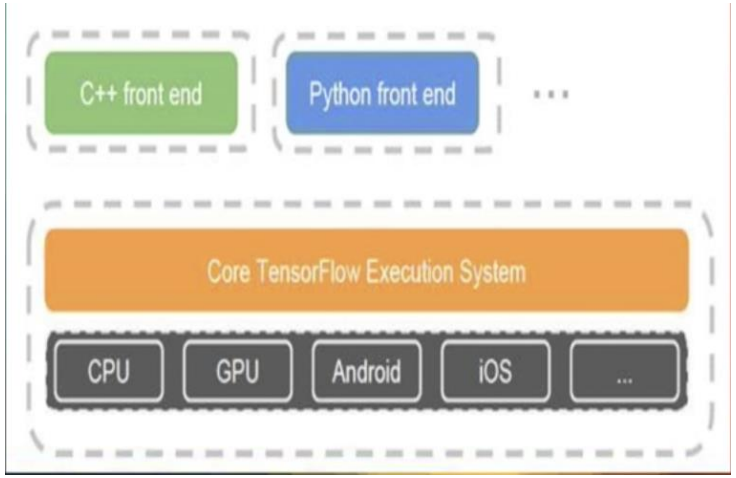
Hackathon Essentials

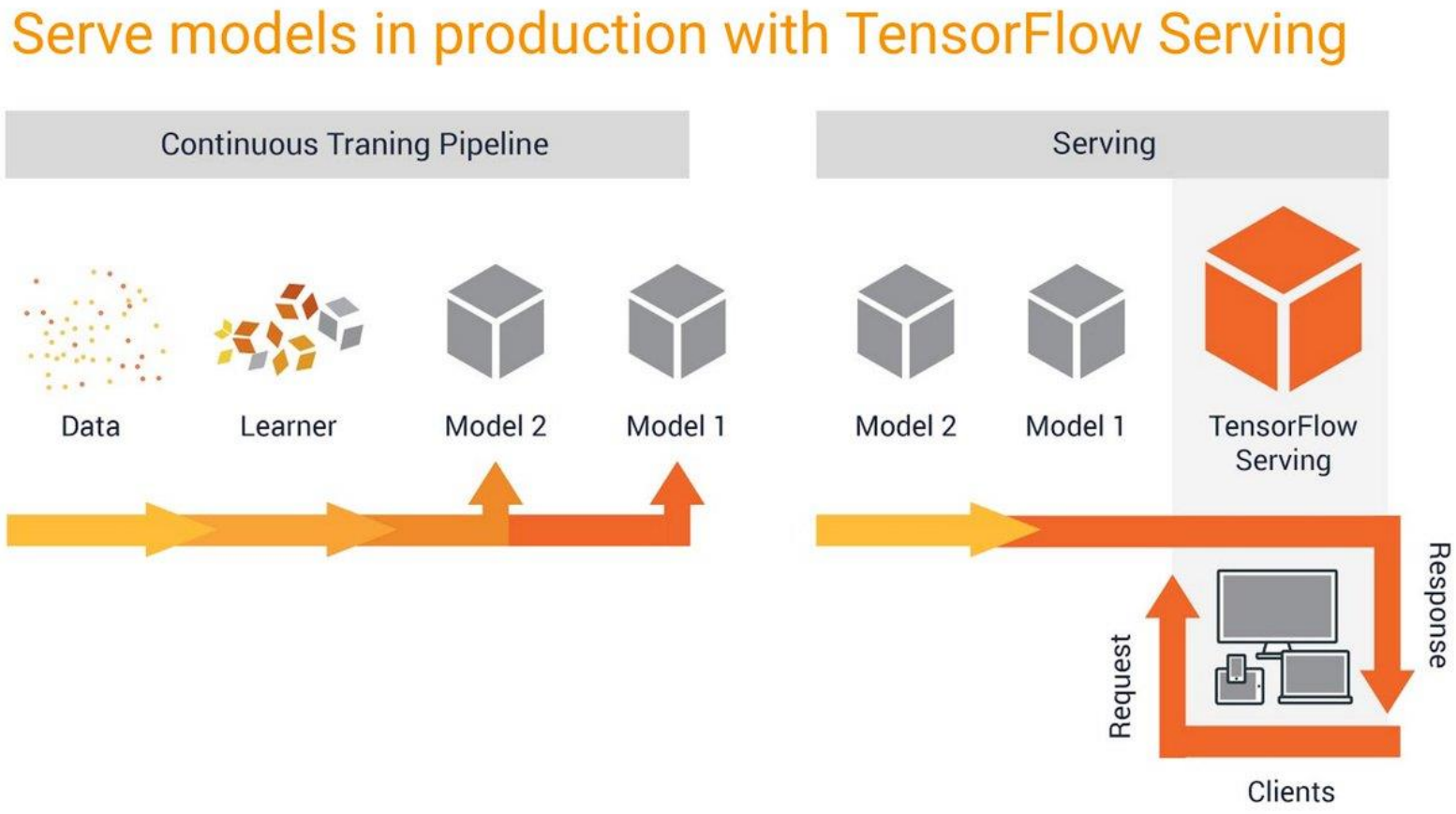
Tensorflow in 5 minutes

Tensorflow

- Wizards use TensorFlow!!
- Developed by Google. Now, Open Source YAY!
- Super duper Scalable. Supported on multiple OS's and can work with multiple programming languages



ML Programming with TensorFlow



Get the Data

```
# Import MNIST data
import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

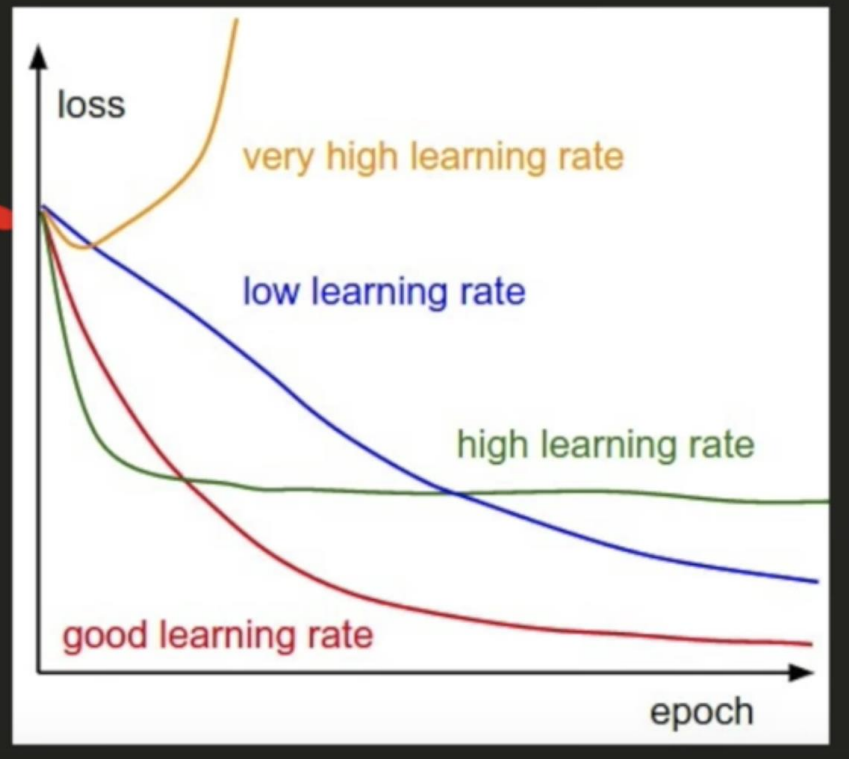
1. Downloads Dataset

2. Splits it into Training + Testing Data

3. Formats it

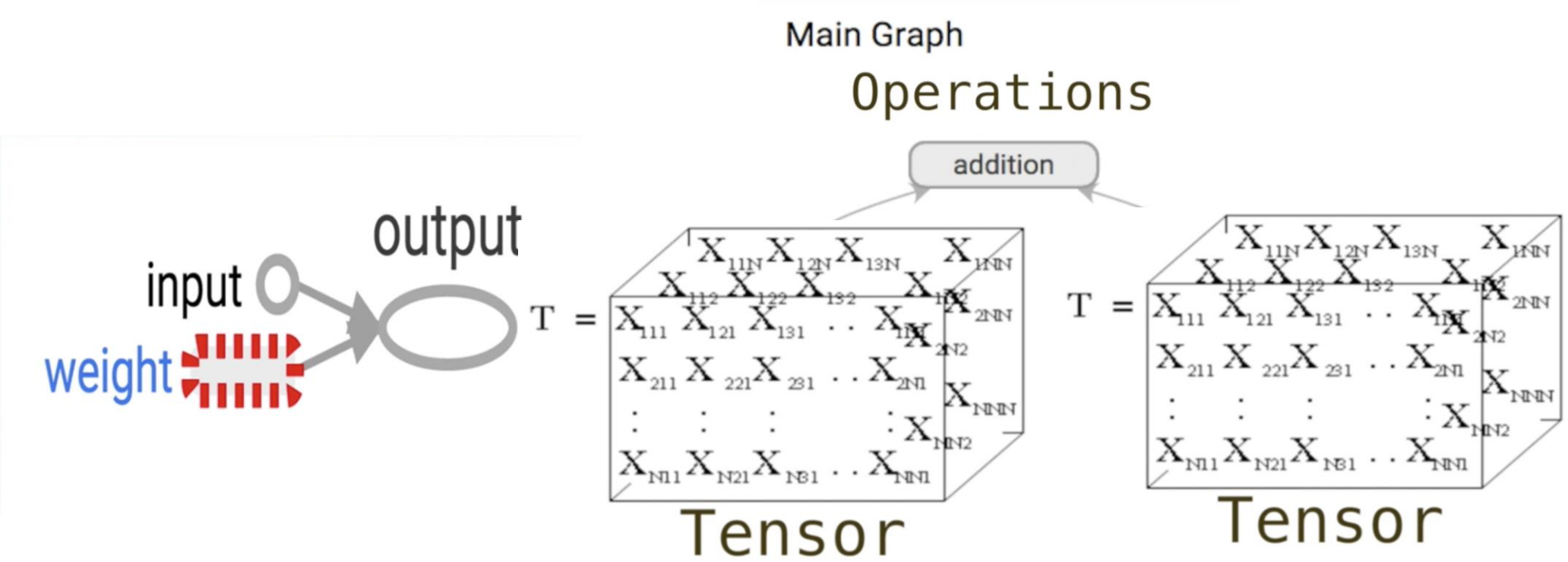
Learning Rate: How fast to update weights?

```python
# Import MNIST data
import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

import tensorflow as tf

# Set parameters
learning_rate = 0.01
training_iteration = 30
batch_size = 100
display_step = 2
```



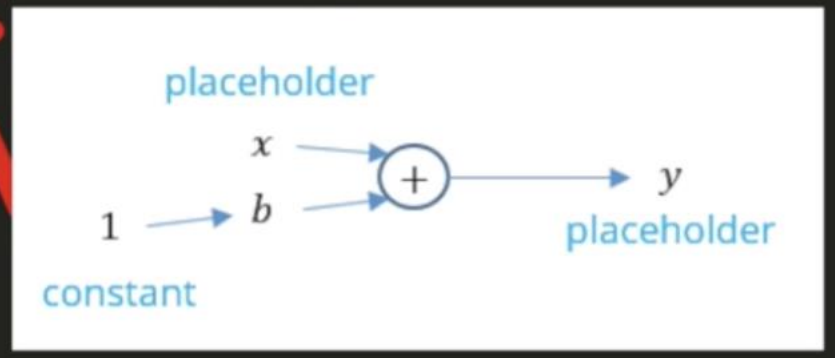
Create a Model

- In Tensorflow, model is a graph containing Nodes.
- Nodes take input (Tensor), perform operations, give some output (Tensor)!
- Tensor: Multi dimensional data type that flows between operations



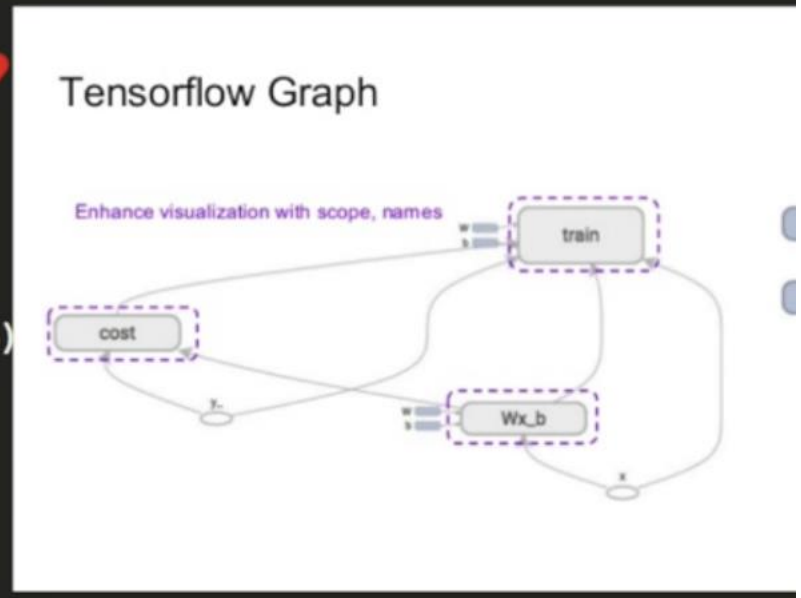
Placeholder, Weights, Bias

```
# TF graph input
x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

# Create a model

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

with tf.name_scope("Wx_b") as scope:
    # Construct a linear model
    model = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Add summary ops to collect data
w_h = tf.histogram_summary("weights", W)
b_h = tf.histogram_summary("biases", b)
```



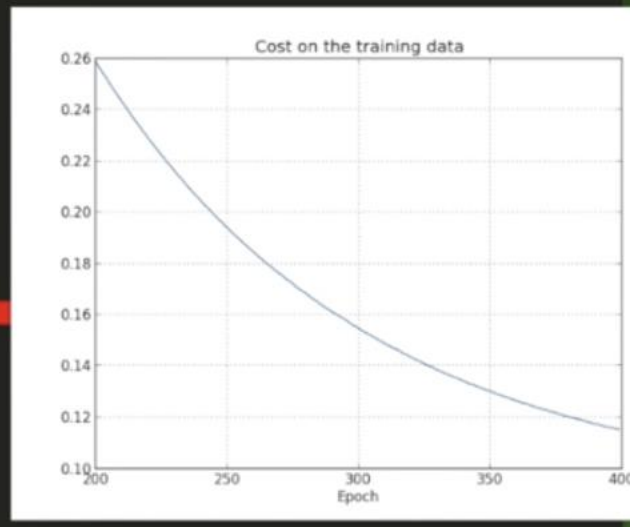
Scope, Summary for Tensorboard

```
# TF graph input
x = tf.placeholder("float", [None, 784]) # mnist data image of shape 28*28=784
y = tf.placeholder("float", [None, 10]) # 0-9 digits recognition => 10 classes

# Create a model

# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

with tf.name_scope("Wx_b") as scope:
    # Construct a linear model
    model = tf.nn.softmax(tf.matmul(x, W) + b)

# Add summary ops to collect data
w_h = tf.histogram_summary("weights", W)
b_h = tf.histogram_summary("biases", b)
```



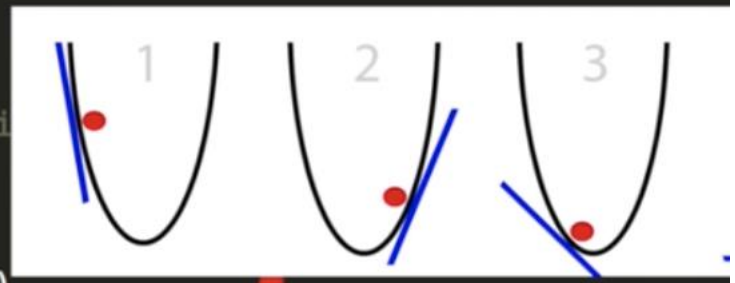
Cost Function

```
# Add summary ops to collect data
w_h = tf.histogram_summary("weights", W)
b_h = tf.histogram_summary("biases", b)

# More name scopes will clean up graph representation
with tf.name_scope("cost_function") as scope:
    # Minimize error using cross entropy
    # Cross entropy
    cost_function = -tf.reduce_sum(y*tf.log(model))
    # Create a summary to monitor the cost function
    tf.scalar_summary("cost_function", cost_function)

with tf.name_scope("train") as scope:
    # Gradient descent
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_function)

# Initializing the variables
init = tf.initialize_all_variables()

# Merge all summaries into a single operator
merged_summary_op = tf.merge_all_summaries()
```



Optimization Function

```
# Add summary ops to collect data
w_h = tf.histogram_summary("weights", W)
b_h = tf.histogram_summary("biases", b)

# More name scopes will clean up graph representati
with tf.name_scope("cost_function") as scope:
    # Minimize error using cross entropy
    # Cross entropy
    cost_function = -tf.reduce_sum(y*tf.log(model))
    # Create a summary to monitor the cost function
    tf.scalar_summary("cost_function", cost_function)

with tf.name_scope("train") as scope:
    # Gradient descent
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_function)

# Initializing the variables
init = tf.initialize_all_variables()

# Merge all summaries into a single operator
merged_summary_op = tf.merge_all_summaries()
```



Session: Launch the graph

```
# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    # Set the logs writer to the folder /tmp/tensorflow_logs
    summary_writer = tf.train.SummaryWriter('/home/sergo/work/logs', graph_def=sess.graph_def)

    # Training cycle
    for iteration in range(training_iteration):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Fit training using batch data
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # Compute the average loss
            avg_cost += sess.run(cost_function, feed_dict={x: batch_xs, y: batch_ys})/total_batch
            # Write logs for each iteration
            summary_str = sess.run(merged_summary_op, feed_dict={x: batch_xs, y: batch_ys})
            summary_writer.add_summary(summary_str, iteration*total_batch + i)
        # Display logs per iteration step
        if iteration % display_step == 0:
            print "Iteration:", '%04d' % (iteration + 1), "cost=", "{:.9f}".format(avg_cost)
```

Test the model

```
        # Display logs per iteration step
        if iteration % display_step == 0:
            print "Iteration:", '%04d' % (iteration + 1), "cost=", "{:.9f}".format(avg_cost)

    print "Tuning completed!"

    # Test the model
    predictions = tf.equal(tf.argmax(model, 1), tf.argmax(y, 1))
    # Calculate accuracy
    accuracy = tf.reduce_mean(tf.cast(predictions, "float"))
    print "Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels})
```

Questions, Brain jogging facts?

Credits: Siraj Rawal and Google! Source: https://github.com/llSourcell/tensorflow_demo