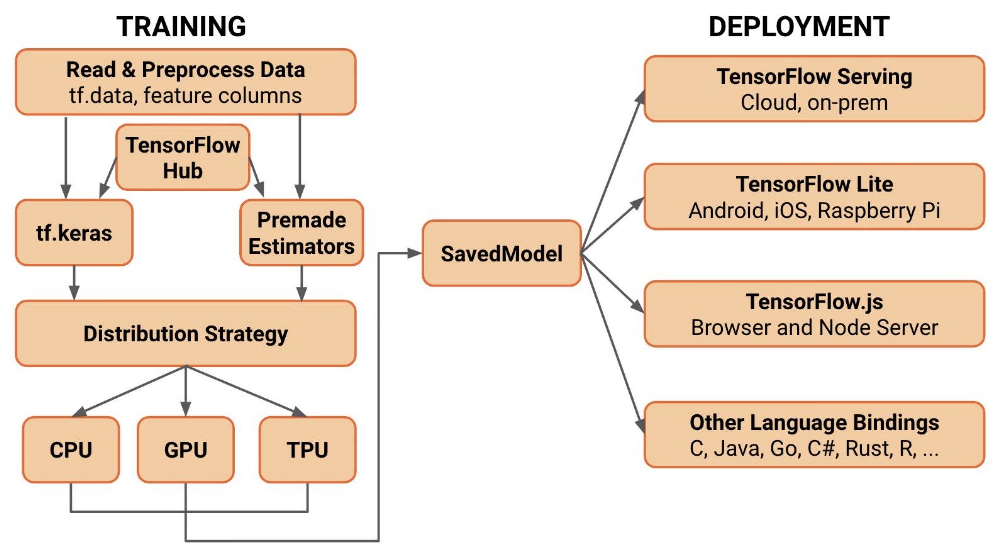
CS 4501: Intro to TensorFlow

```
import tensorflow as tf
```

Why TensorFlow

- Fast
  - Hardware accelerations
  - Optimized functions/models
- Flexible
  - Extensible low-level API
  - Ecosystem: Browser, Mobile, IOT...
  - Libraries: TF Probability, TF Federated, TF Ranking, TF Agents, TF Hub...
- Simple
  - Accessible high-level API to develop: Keras, Eager execution
  - Debugging and visualization tool: tfdbg, Tensorboard
  - Deploy at large scale and in heterogeneous environments



TensorFlow overview

- Low Level API
  - Tensor: N-dimensional array
  - Flow: Computational Graph
  - Session: Execution Context
- High Level API
  - Keras
  - Eager Execution
  - Estimator API
  - Accelerator API

TensorFlow overview

Access TF workshop notebook at

https://git.wujibang.com/TFWorkshop/blob/master/tf_workshop.ipynb

Also linked on course schedule page

Tensor

```
# numpy

>>> np.zeros((2, 1))
array([[ 0.],
       [ 0.]])

>>> np.ones((1, 2),dtype=int)
array([[ 1, 1]])
```

```
# tensorflow

>>> tf.zeros([2, 1])

tensor([[ 0.],
        [ 0.]])

>>> tf.ones([1, 2],
dtype=tf.int32)

tensor([[ 1, 1]])
```

Tensor

```
# Constant 1-D Tensor populated with value list.

tensor = tf.constant([1, 2, 5]) => [1 2 5]

# Constant 2-D tensor populated with scalar value -1.

tensor = tf.constant(-1.0, shape=[1, 3]) => [[-1. -1. -1.]]

# Create a tensor of shape [2, 3] consisting of random normal
values, with mean -1 and standard deviation 4.

norm = tf.random_normal([2, 3], mean=-1, stddev=4)
```

```python
# numpy

a = np.zeros((2, 1))

b = np.ones((1, 2))

>>> a[0,0], a[:,0], a[0,:]

>>> np.reshape(a,(2,1))

>>> a.shape

(2, 1)

>>> np.sum(b, axis=1)

2
```

```python
# tensorflow

a = tf.zeros([2, 1])

b = tf.ones([1, 2])

>>> a[0,0], a[:,0], a[0,:]

>>> tf.reshape(a,(2,1))

>>> a.shape

(2, 1)

>>> tf.reduce_sum(b, [1])

???
```

tf.session *environment in which Operation objects are executed*

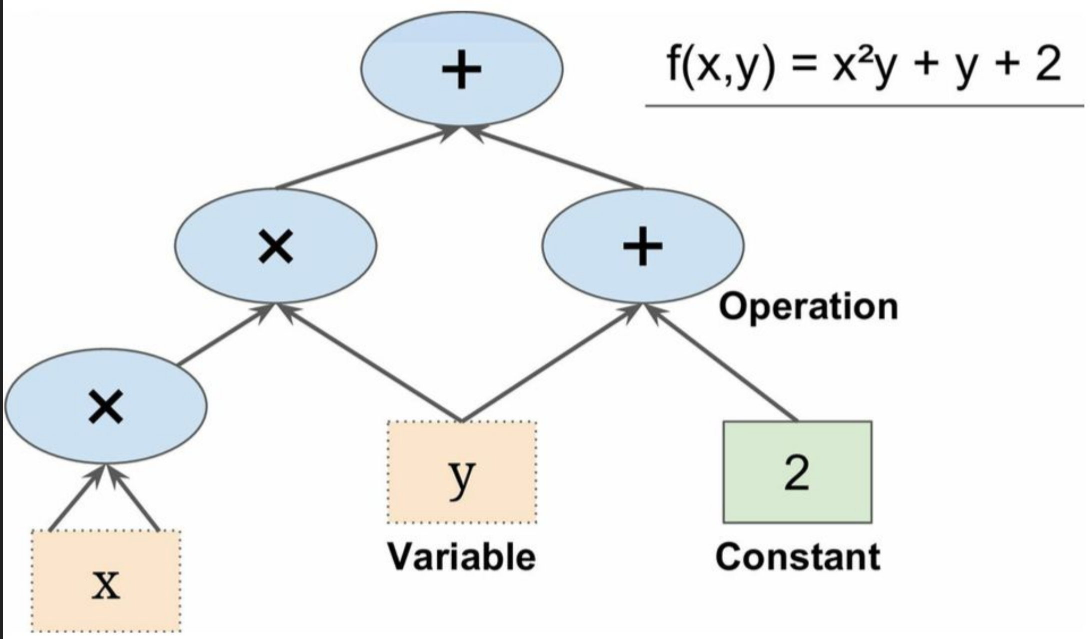
```
tf.Session().run( tensor1d )

tf.Session().run( [tensor2d, tensornorm] )

tf.Session().run( tf.reduce_sum(b, [1]) )
```

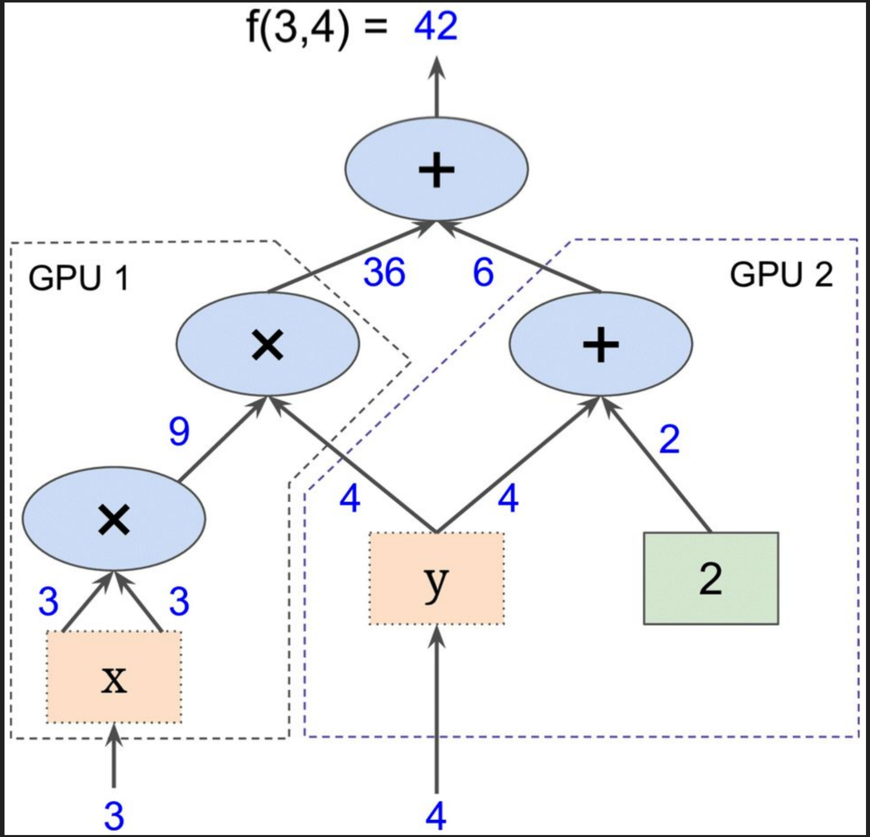
TensorFlow *Idea of Computation Graph*

- Node: Operation, Variable, Constant, Placeholder
- Edge: input/output Data (Tensor)



TensorFlow *Why Computation Graph*

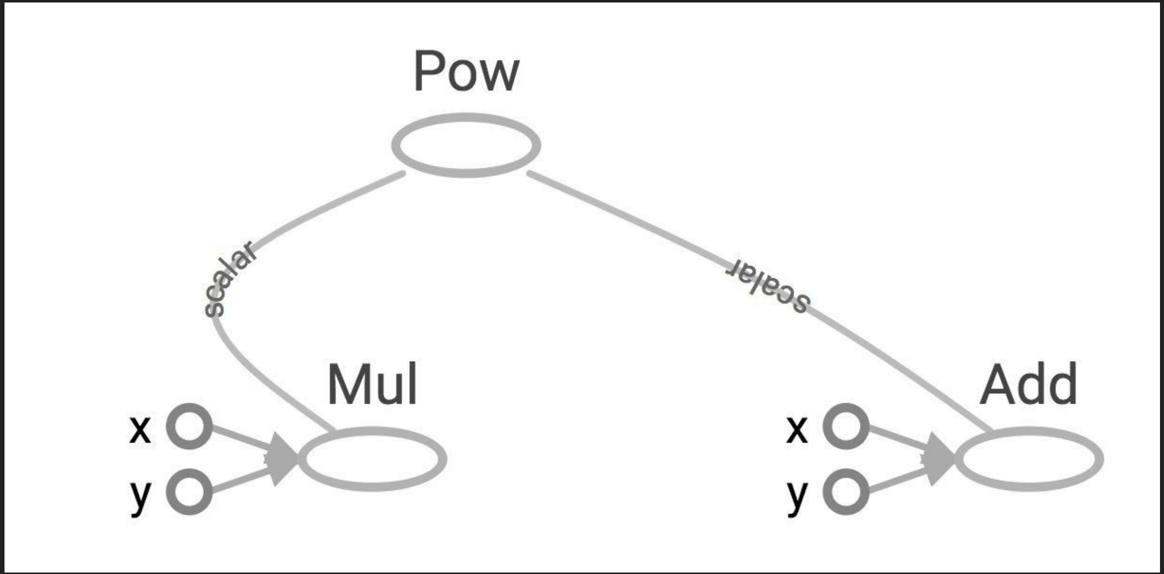
- Separates definition of computations from their execution
- Store Computation State
  - Auto-differentiation in Backprop
- Dynamic Control Flow
- Partial Execution
- Concurrent & Distributed Execution
  - Compiler Optimization
  - Scale & Speed up



```
# Phase 1: assemble a graph, no computation happen at this phase!!!

x = tf.constant(2)
y = tf.constant(3)
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)
op3 = tf.pow(op2, op1)
# Phase 2: use a session to execute operations in the graph.
with tf.Session() as sess:
     print(sess.run([op3, op2]))

### [7776, 6]
```



TensorBoard *visualizing learning*

```
x = tf.constant(2, name='a') # explicit naming the node
y = tf.constant(3, name='b')
op1 = tf.add(x, y, name='add')

with tf.Session() as sess:

    # save computational graph in directory used by tensorboard
    writer = tf.summary.FileWriter('./logs', sess.graph)

    print(sess.run(op1))
writer.close() # close the writer when you’re done using it
```

tf.placeholder *hold the place for input/output tensor*

```
x_placeholder = tf.placeholder(tf.float32, shape=())
y = tf.constant(3)
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)

with tf.Session() as sess:
    print(sess.run([op1, op2], feed_dict={x_placeholder : 1}))

### [4, 3]

    print(sess.run([op1, op2], feed_dict={x_placeholder : 2}))

### [5, 6]
```

tf.Variable *store the tensor whose value/state can be changed (trained)*

```
# Simulate training Data in numpy
num_samples = 1000; num_features = 5;
data_X = np.random.rand(num_samples, num_features)
data_Y = np.dot(data_X, np.random.rand(num_features, 1))  #simulate a linear relation
X_train, X_test, Y_train, Y_test = train_test_split(data_X, data_Y, test_size=0.1)

# Graph Input Nodes
X = tf.placeholder(tf.float64, shape=(None, num_features), name="features")
Y = tf.placeholder(tf.float64, shape=(None, 1), name="label")

# let model weight be Variable that is learnable
W = tf.Variable(np.random.rand(num_features, 1), name="weight")
b = tf.Variable(np.random.rand(1, 1), name="bias")

pred = tf.add(tf.matmul(X, W), b)    # Construct a linear model y = xW + b
cost = tf.reduce_mean(tf.pow(pred-Y, 2))    # Mean squared error
```

tf.train.optimizer *optimizer API to train a model*

```
# construct Gradient Descent optimizer
learning_rate = 0.1
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
trainstep = optimizer.minimize(cost)


# log your costs in tensorboard
trn_summary = tf.summary.scalar("training_cost", cost)
eval_summary = tf.summary.scalar("evaulation_cost", cost)
```

```python
with tf.Session() as sess:
    writer = tf.summary.FileWriter('./logs', sess.graph) # log info to visualize in tensorboard
    sess.run(tf.global_variables_initializer()) #initialize Variables
    for epoch in range(100):
        # compute on train subgraph
        _, trn = sess.run([optimizer, trn_summary], feed_dict={X: X_train, Y: Y_train})
        writer.add_summary(trn, epoch)
        # compute on evaluation subgraph
        eval = sess.run(evaluation_summary, feed_dict={X: X_test, Y: Y_test})
        writer.add_summary(eval, epoch)
```

Hands-on: *Logistic Regression in* TF

tf.keras *high-level API*

1. `Model Definition # layer structure`
2. `Model.compile() # specific loss, optimizer, metrics`
3. `Model.fit() # training phrase, with automation options`
4. `Model.evaluate() # testing phrase`

Tensorflow 2.0: A peak at Eager Execution

- Eager by default, so Value is immediately evaluated
- `tf.GradientTape() # Calculate gradient on demand`
- `tf.ragged.*     # Allowing Ragged Tensor`
- Supports natural control flow i.e. if, while
- `@tf.function() # JIT wrapper for tf.Session()`

Effective TF *syntax sugar and elementwise broadcasting*

```
z = -x             z = tf.negative(x)
z = x + y          z = tf.add(x, y)
z = x - y          z = tf.subtract(x, y)
z = x * y          z = tf.mul(x, y)
z = x / y          z = tf.div(x, y)
z = x // y         z = tf.floordiv(x, y)
z = x % y          z = tf.mod(x, y)
z = x ** y         z = tf.pow(x, y)
z = x @ y          z = tf.matmul(x, y)
z = x > y          z = tf.greater(x, y)
z = x >= y         z = tf.greater_equal(x, y)
z = x < y          z = tf.less(x, y)
z = x <= y         z = tf.less_equal(x, y)
z = abs(x)         z = tf.abs(x)
z = x & y          z = tf.logical_and(x, y)
z = x | y          z = tf.logical_or(x, y)
z = x ^ y          z = tf.logical_xor(x, y)
z = ~x             z = tf.logical_not(x)
```

Effective TF *good debugging practice*

- `tf.Print`
- `Tf.compute_gradient_error`
- `tf.assert*`
- `tf.add_check_numerics_ops`

Effective TF *performance optimization*

- Tf.dataset API
  - Batchize training
  - Sequence Padding
- Remove debug ops

Effective TF *scopes and when to use them*

- `name_scope`
- `variable_scope`

TensorFlow *installation*

- Use cloud environment on **Google Colab**
- Use local CPU/GPU environment with tensorflow
  - `# install latest release for CPU-only`
  - `$ pip install tensorflow`
  - `# OR install latest release for GPU`
  - `$ pip install tensorflow-gpu`
  - Troubleshoot here https://www.tensorflow.org/install/pip
- Use GPU/TPU environment on cloud provider
  - gcloud tutorial here
  - aws tutorial here

References

For Installation: https://www.tensorflow.org/install/

For API Doc: https://www.tensorflow.org/guide/

Stanford CS 20: Tensorflow for Deep Learning Research:
http://web.stanford.edu/class/cs20si/syllabus.html

Effective Tensorflow: https://github.com/vahidk/EffectiveTensorflow