TensorBoard

Glenn Bruns
CSUMB

Much material in this deck from Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow

Learning outcomes

After this lecture you should be able to:

1. Extend TensorFlow code to enable use of TensorBoard
2. Use TensorBoard
3. Use the following language features of TensorFlow:
   - saving and restoring models
   - name scopes
   - modular graph construction

Recap: TensorFlow

```
n_epochs = 1000
learning_rate = 0.01
```

```
X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42),
name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
gradients = 2/m * tf.matmul(tf.transpose(X), error)
training_op = tf.assign(theta, theta - learning_rate * gradients)

init = tf.global_variables_initializer()
```

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

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()
```

Saving models

```
n_epochs = 1000
learning_rate = 0.01
```

```
X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
…
gradients = 2/m * tf.matmul(tf.transpose(X), error)
training_op = tf.assign(theta, theta - learning_rate * gradients)
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

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

    for epoch in range(n_epochs):
        if epoch % 100 == 0:      # checkpoint every 100 epochs
            save_path = saver.save(sess, "/tmp/my_model.ckpt")
        sess.run(training_op)

    best_theta = theta.eval()
    save_path = saver_save(sess, "/tmp/my_model_final.ckpt")
```

"saving a model" = saving values of all variables in the model

Restoring models

```
n_epochs = 1000
learning_rate = 0.01
```

```
X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
…
gradients = 2/m * tf.matmul(tf.transpose(X), error)
training_op = tf.assign(theta, theta - learning_rate * gradients)
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

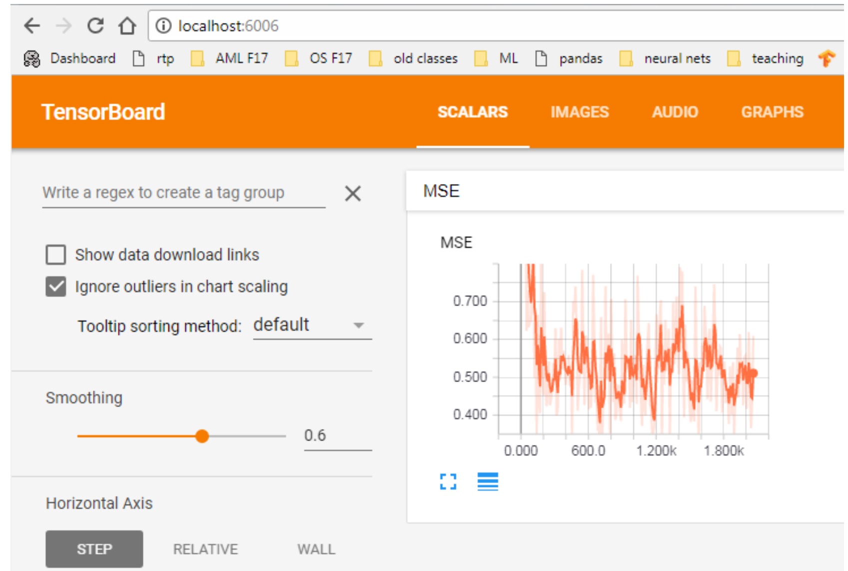
```
with tf.Session() as sess:
    # sess.run(init)
    saver.restore(sess, "/tmp/my_model_final.ckpt")  # instead of run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:      # checkpoint every 100 epochs
            save_path = saver.save(sess, "/tmp/my_model.ckpt")
        sess.run(training_op)

    best_theta = theta.eval()
```

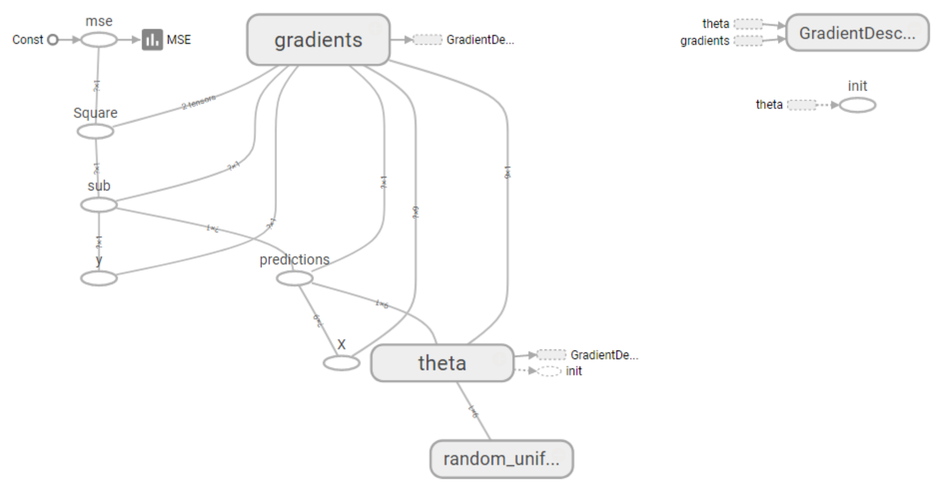
By default, variables are saved and restored under their own names.

TensorBoard

Provided with TensorFlow; allows browser-based visualization to:

- Visualize data collected during graph execution
  - e.g., to see training progress
- Visualize the computation graph itself





Modifying code for TensorFlow

```
now = datetime.utcnow().strftime("%Y%m%d%H%M%S")
root_logdir = "E:/Glenn/tf_logs"
logdir = "{}/run-{}/".format(root_logdir, now)
```

start of program: define unique log directory

```
mse_summary = tf.summary.scalar('MSE', mse)
file_writer = tf.summary.FileWriter(logdir,
                            tf.get_default_graph())
```

end of construction phase: create node to evaluate MSE

```
[…]
for batch_index in range(n_batches):
   X_batch, y_batch = fetch_batch(epoch, batch_index, batch_size)
   if batch_index % 10 == 0:
      summary_str = mse_summary.eval(feed_dict={X: X_batch, y: y_batch})
      step = epoch * n_batches + batch_index
      file_writer.add_summary(summary_str, step)
   sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
[…]
```

execution phase: write MSE values every so often

```
file_writer.close()
```

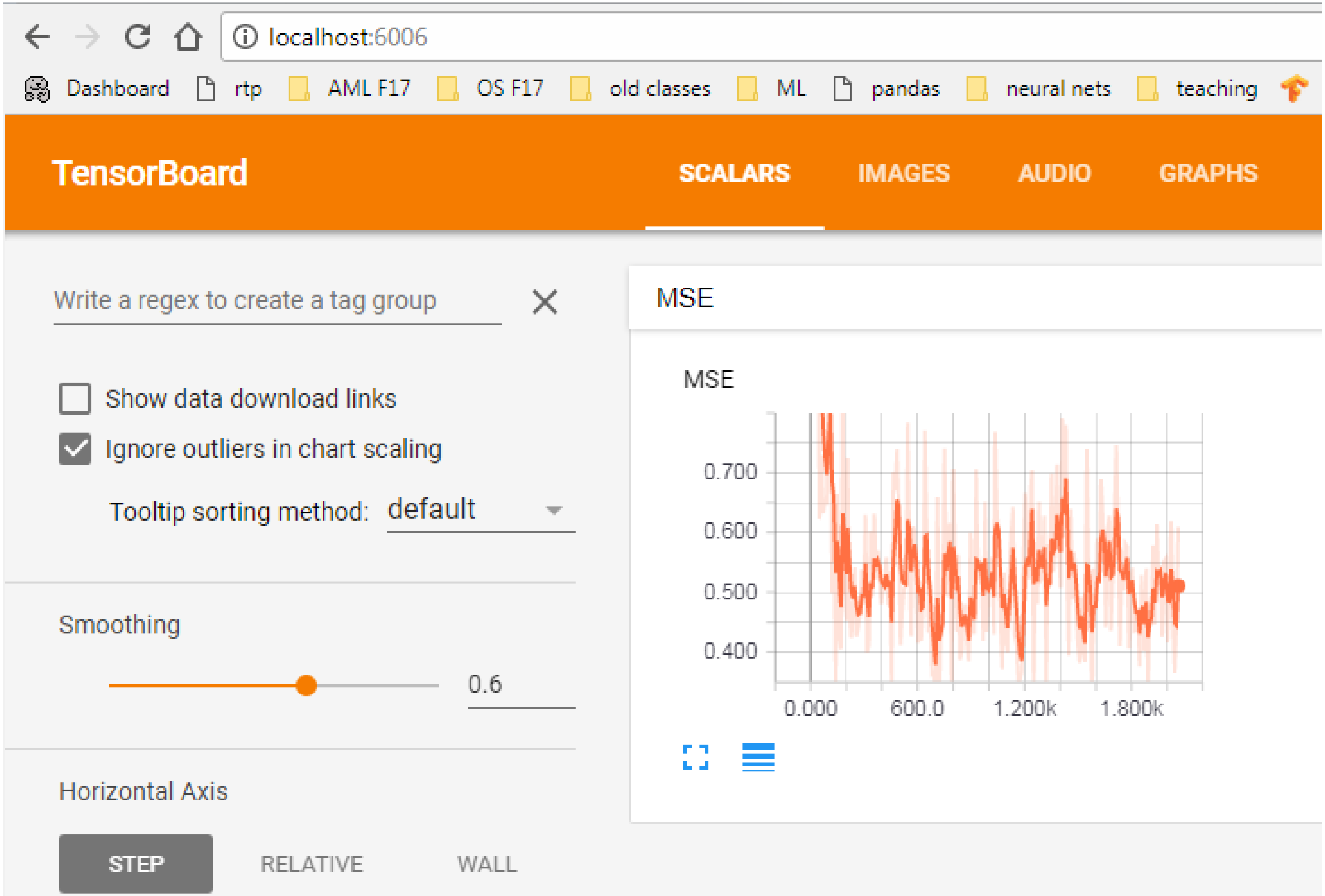
end of program: close FileWriter

Running TensorBoard

After running your TF program and creating the log directory:

1. Within an Anaconda window, activate your tensorflow environment
2. > tensorboard --logdir tf_logs/
3. Enter URL localhost:6006 in your browser
   - or, try 0.0.0.0:6006   (this didn't work for me)
4. Click on MSE to see line plot of MSE values
5. Click on 'GRAPHS' tab to see your graph

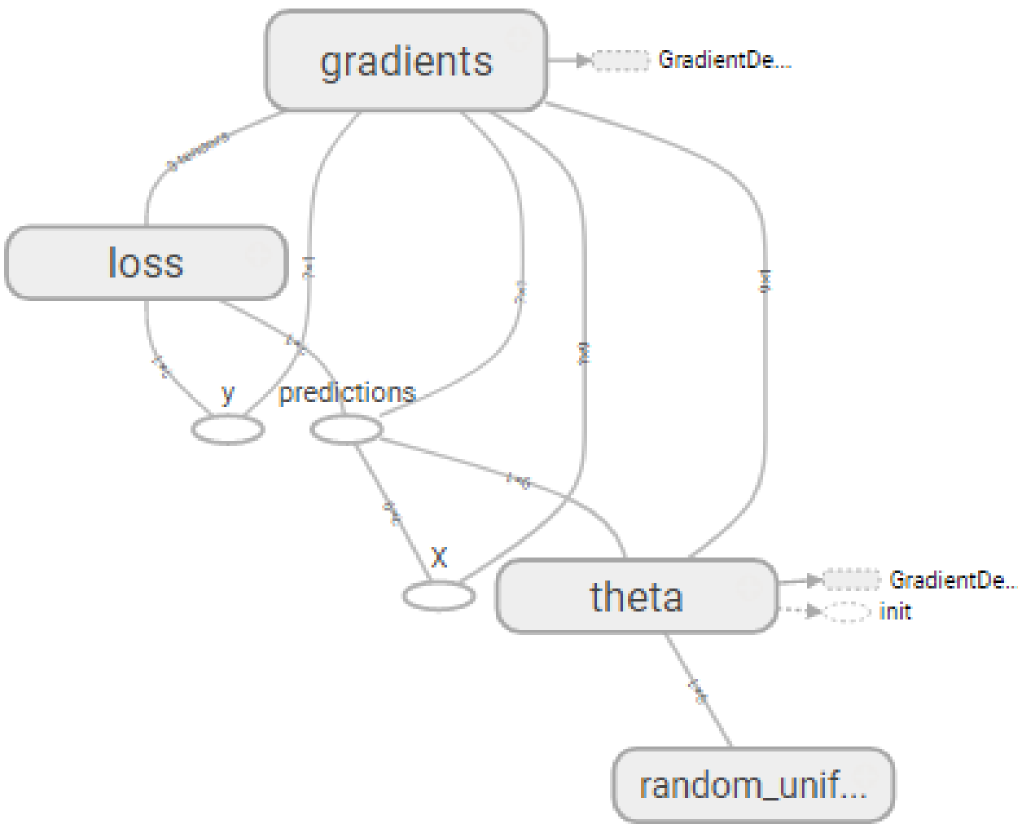
MSE plot



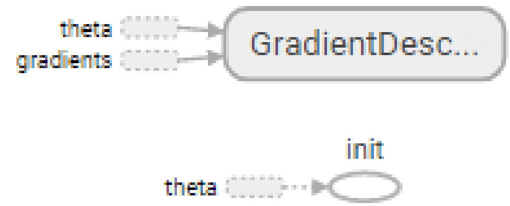
localhost:6006
Dashboard
rtp
AML F17
OS F17
old classes
ML
pandas
neural nets
teaching
TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
Write a regex to create a tag group
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
MSE
MSE
0.700
0.600
0.500
0.400
0.000
600.0
1.200k
1.800k

Graph visualization

Main Graph



Auxiliary Nodes



- scroll to zoom in and out
- hold left mouse button to pan
- hover over node, then click + to expand a subgraph

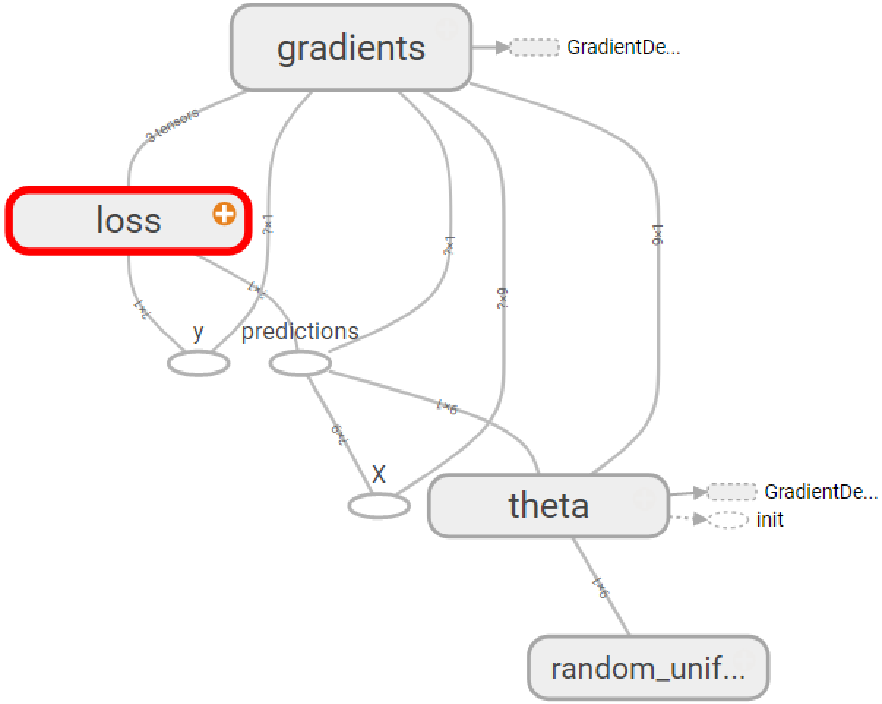
Name scopes

Reduce clutter in TensorBoard view of graph by grouping related nodes.

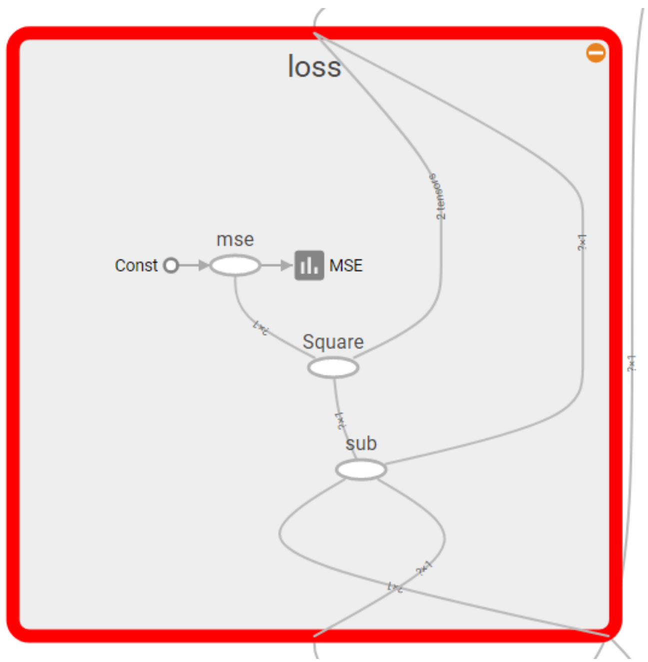
```
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42),
name="theta")
y_pred = tf.matmul(X, theta, name="predictions")

# put error and mse within a single name score
with tf.name_scope("loss") as scope:
    error = y_pred - y
    mse = tf.reduce_mean(tf.square(error), name="mse")

optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)
```

This code creates a "loss" name scope that simplifies the graph

Loss namespace



collapsed



expanded

Duplicated TF code

In any language, duplicated code leads to errors and maintenance problems. (DRY principle)

```
n_features = 3
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")

w1 = tf.Variable(tf.random_normal((n_features, 1)), name="weights1")
w2 = tf.Variable(tf.random_normal((n_features, 1)), name="weights2")
b1 = tf.Variable(0.0, name="bias1")
b2 = tf.Variable(0.0, name="bias2")

z1 = tf.add(tf.matmul(X, w1), b1, name="z1")
z2 = tf.add(tf.matmul(X, w2), b2, name="z2")

relu1 = tf.maximum(z1, 0., name="relu1")
relu2 = tf.maximum(z1, 0., name="relu2")
output = tf.add(relu1, relu2, name="output")
```

$h_{\mathbf{w},b}(\mathbf{X}) = \max(\mathbf{X} \cdot \mathbf{w} + b, 0)$ (rectified linear unit, ReLU)

Modularity

```
# build a ReLU
def relu(X):
    w_shape = (int(X.get_shape()[1]), 1)
    w = tf.Variable(tf.random_normal(w_shape), name="weights")
    b = tf.Variable(0.0, name="bias")
    z = tf.add(tf.matmul(X, w), b, name="linear")
    return tf.maximum(z, 0., name="relu")

n_features = 3
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = [relu(X) for i in range(5)]
output = tf.add_n(relus, name="output")
```

When a node is created, TF ensures name uniqueness when adding the node to the graph.

First ReLU contains "weights", "biases", etc.

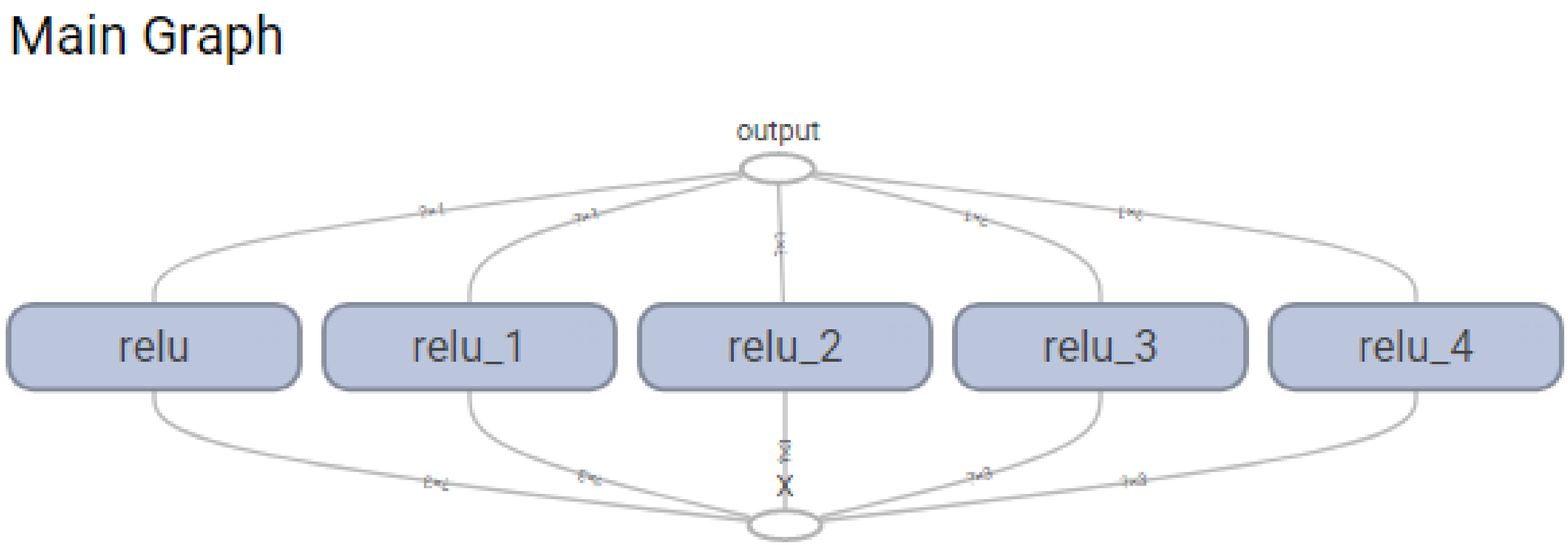
Second ReLU contains "weights_1", "biases_1", etc.

Modularity improved with name scope

```
# build a ReLU
def relu(X):
   with tf.name_scope("relu"):
      w_shape = (int(X.get_shape()[1]), 1)
      w = tf.Variable(tf.random_normal(w_shape), name="weights")
      b = tf.Variable(0.0, name="bias")
      z = tf.add(tf.matmul(X, w), b, name="linear")
      return tf.maximum(z, 0., name="relu")

n_features = 3
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = [relu(X) for i in range(5)]
output = tf.add_n(relus, name="output")
```

The graph in TensorBoard will now look much better.

Modularity + name scope in TB



As our computation graphs get bigger, the use of modularity and name scope will become more important.

Summary

- saving and restoring sessions
- TensorBoard
  - modifying your code for TensorBoard
  - using TensorBoard
- name scopes and modularity to clarify code and its TensorBoard graph