A Introduction to TensorFlow

Huixiong Qin

2018 12 27

Why Python and TensorFlow?

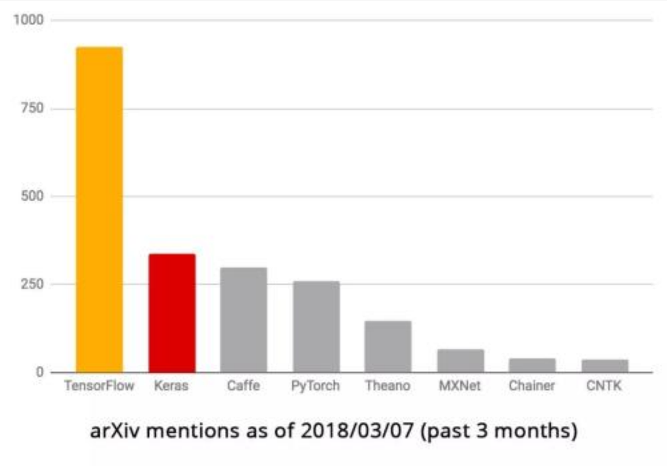
Glue Language
python is designed specifically to write and manage program and code, which connects together different software components.

API	language
Cython	Python Extending with C/ C++
rpy2	R in Python
JPype	Java in Python
PyExecJS	Run JavaScript code from Python
python-sql	Connect SQL Server in Python

Table: examples

Machine Learning/Deep Learning Frameworks

- ML: Theano&Ecosystem, Torch, TensorFlow, Caffe, CNTK, DSSTNE, Speed
- DL: Scikit-learn, Apache Mahout, SystemML, Microsoft DMTK



High Availability

- TensorFlow is perfectly compatible to NumPy, containing plenty of high-level APIs.
- TensorFlow programs are allow to deploy in a wide range of devices, from computer groups to smart phones.

Completed Communities and Abundant Sort of Study Resource

- An online open course
- The most popular deep learning framework in Github.

TensorBoard: Visualizing Learning

- Visualize your TensorFlow graph.
- Plot quantitative metrics about the execution of your graph.
- Show additional data like images that pass through it.

Basic Classes

When writing a TensorFlow program, the main object you manipulate and pass around is the tf.Tensor.

constructed function

__init__(op, value_index, dtype)
Args:

- op: An **Operation**. Operation that computes this tensor.
- value_index: An **int**. Index of the operation's endpoint that produces this tensor.
- dtype: A **DType**. Type of elements stored in this tensor.

Properties

- device: The name of the device on which this tensor will be produced, or None.
- dtype: The DType of elements in this tensor.
- graph: The Graph that contains this tensor.
- name: The string name of this tensor.
- op: The Operation that produces this tensor as an output.
- shape: Returns the TensorShape that represents the shape of this tensor.

Method

__add__(), __eq__(), __matmul__(), __eval__()
...

Remark

These methods are the same as tf.add_n(), tf.matmul(), ...

tf.variable

A tf.variable represents a tensor whose value can be changed by running ops on it.

__init__(initial_value=None, validate_shape=True, name=None)

```
v = tf.Variable([1, 2], validate_shape=[1, 2], name="v1")
```

tf.constant

Creates a constant tensor.

__init__(value, dtype=None, shape=None, name='Const')

```
# Constant 1D Tensor populated with value list.
tensor = tf.constant([1, 2, 3, 4, 5, 6, 7])
=> [1, 2, 3, 4, 5, 6, 7]
# Constant 2D tensor populated with scalar value -1.
tensor = tf.constant(-1.0, shape=[2, 3])
=> [[-1. -1. -1.]
    [-1. -1. -1.]]
```

tf.placeholder

Inserts a placeholder for a tensor that will be always fed.

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

```
x = tf.placeholder(tf.float32, shape=(1024, 1024))
with tf.Session() as sess:
rand_array = np.random.rand(1024, 1024)
print(sess.run(y, feed_dict={x: rand_array}))
```

TensorFlow.Operation

An op is a node in a TensorFlow Graph that takes zero or more Tensor objects as input, and produces zero or more Tensor objects as output.
Objects of type Operation are created by calling a Python op constructor (such as **tf.matmul**) or **tf.Graph.create_op**.

Create a op

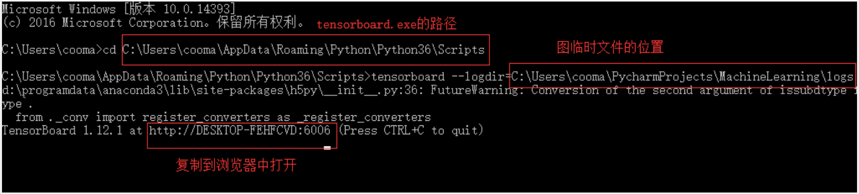
create_op(op_type, inputs, dtypes,)
Args:

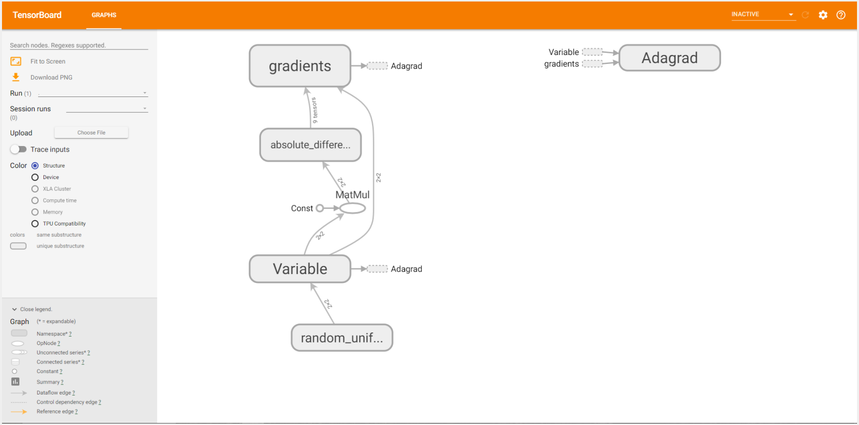
- op_type: The Operation type to create. This corresponds to the **OpDef.name** field.
- inputs: A list of Tensor objects that will be inputs to the Operation.
- dtypes: A list of DType objects that will be the types of the tensors that the operation produces.

TensorFlow.Graph

TensorFlow uses a dataflow graph to represent your computation in terms of the dependencies between individual operations. In a dataflow graph, the nodes represent units of computation(**tf.Operation**), and the edges represent the data consumed or produced by a computation(**tf.Tensor**). TensorFlow would build a default graph that is an implicit argument to all API functions in the same context.

```python
# Build your graph.
x = tf.constant([[37.0, -23.0], [1.0, 4.0]])
w = tf.Variable(tf.random_uniform([2, 2]))
y = tf.matmul(x, w)
y_ = tf.constant([[0, 0], [1, 1]])

loss = tf.losses.absolute_difference(y_, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)

with tf.Session() as sess:
    writer = tf.summary.FileWriter("../logs", sess.graph)
    init = tf.global_variables_initializer()
    sess.run(init)
    for i in range(1000):
        sess.run(train_op)
    writer.close()
```


Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation。保留所有权利。
tensorboard.exe的路径
C:\Users\cooma>cd C:\Users\cooma\AppData\Roaming\Python\Python36\Scripts
图临时文件的位置
C:\Users\cooma\AppData\Roaming\Python\Python36\Scripts>tensorboard --logdir=C:\Users\cooma\PycharmProjects\MachineLearning\logs
d:\programdata\anaconda3\lib\site-packages\h5py__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype
ype .
from ._conv import register_converters as _register_converters
TensorBoard 1.12.1 at http://DESKTOP-FEHFCVD:6006 (Press CTRL+C to quit)
复制到浏览器中打开


TensorBoard
GRAPHS
INACTIVE
Search nodes. Regexes supported.
Fit to Screen
Download PNG
Run (1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
TPU Compatibility
colors
same substructure
unique substructure
Close legend.
Graph
(* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?
gradients
Adagrad
9 tensors
absolute_differe...
2x2
MatMul
Const
Variable
random_unif...
Variable
gradients
Adagrad

TensorFlow computing relies on a efficient C++ server. The connection to this back end is tf.session. Generally, we need to build a graph before running it in the session.

session.run(fetches)

fetches argument may be a single graph element, or an arbitrarily nested list, tuple, namedtuple, or dictionary, including objects from **tf.Operation** and **tf.tensor**.

There is a easier way to run your graphs: **tf.InteractiveSession()**

Remark

Before Variables can be used within a session, they must be initialized using that session.

```python
#session.run(fetches)
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
with tf.Session() as sess:
  sess.run(tf.initialize_all_variables())
  print(sess.run(c))
#tf.InteractiveSession()
sess = tf.InteractiveSession()
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
print(c.eval())
sess.close()
```

tf.train.saver.save(sess, save_path)
tf.train.saver.restore(sess, save_path)

Remark

Restored variables does not need to initiate again.

```python
# to save variables
v1 = tf.get_variable("v1", shape=[3], initializer = tf.
                                      zeros_initializer)
v2 = tf.get_variable("v2", shape=[5], initializer = tf.
                                      zeros_initializer)
inc_v1 = v1.assign(v1+1)
dec_v2 = v2.assign(v2-1)
init_op = tf.global_variables_initializer()
saver = tf.train.Saver()
with tf.Session() as sess:
  sess.run(init_op)
  inc_v1.op.run()
  dec_v2.op.run()
  save_path = saver.save(sess, "/tmp/model.ckpt")
# to restore variables
tf.reset_default_graph()
v1 = tf.get_variable("v1", shape=[3])
v2 = tf.get_variable("v2", shape=[5])
saver = tf.train.Saver()
with tf.Session() as sess:
  saver.restore(sess, "/tmp/model.ckpt")
  print("Model restored.")
```

Experiment: Handwritten Digits Recognition

IMPORT DATA

A Typical pipeline for loading data contains the following steps:

1. A list for file names.
2. A file reader/interpretor for certain file formats.
3. Preprocessing, including normalizing, shuffling and adding additional noise.
4. batching.
5. A DataSet class as the return object.

```python
SOURCE_URL = 'http://yann.lecun.com/exdb/mnist/'
def maybe_download(filename, work_directory)
def _read32(bytestream)
def extract_images(filename)
def dense_to_one_hot(labels_dense, num_classes=10)
def extract_labels(filename, one_hot=False)
class DataSet(object)
  def __init__(self, images, labels, fake_data=False,
                              one_hot=False, dtype=tf.
                              float32)
    def next_batch(self, batch_size, fake_data=False)
def read_data_sets(train_dir, fake_data=False, one_hot=
                              False, dtype=tf.float32):
    class DataSets(object):
        pass
    data_sets = DataSets()
    if fake_data:
        def fake():
            return DataSet([], [], fake_data=True, one_hot=
                                          one_hot, dtype=
                                          dtype)
        data_sets.train = fake()
```

```
        data_sets.validation = fake()
        data_sets.test = fake()
        return data_sets
    TRAIN_IMAGES = 'train-images-idx3-ubyte.gz'
    TRAIN_LABELS = 'train-labels-idx1-ubyte.gz'
    TEST_IMAGES = 't10k-images-idx3-ubyte.gz'
    TEST_LABELS = 't10k-labels-idx1-ubyte.gz'
    VALIDATION_SIZE = 5000
    local_file = maybe_download(TRAIN_IMAGES, train_dir)
    train_images = extract_images(local_file)
    local_file = maybe_download(TRAIN_LABELS, train_dir)
    train_labels = extract_labels(local_file, one_hot=
                                  one_hot)
    local_file = maybe_download(TEST_IMAGES, train_dir)
    test_images = extract_images(local_file)
    local_file = maybe_download(TEST_LABELS, train_dir)
    test_labels = extract_labels(local_file, one_hot=
                                  one_hot)
    validation_images = train_images[:VALIDATION_SIZE]
    validation_labels = train_labels[:VALIDATION_SIZE]
    train_images = train_images[VALIDATION_SIZE:]
    train_labels = train_labels[VALIDATION_SIZE:]
    data_sets.train = DataSet(train_images, train_labels,
```

```
                                dtype=dtype)
    data_sets.validation = DataSet(validation_images,
                                   validation_labels,
                                    dtype=dtype)
    data_sets.test = DataSet(test_images, test_labels,
                                   dtype=dtype)
    return data_sets
```

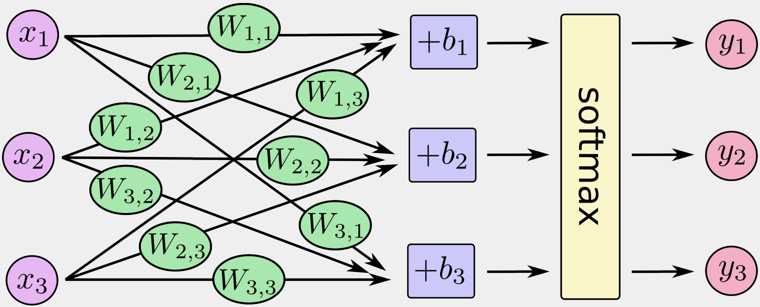
Firstly, to build a elementary model that mainly contains the following steps.

1. Sortmax regression model

$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{N} x_j}$

$y = softmax(Wx + b)$

Where W is the weight matrix, x is the input data, b is the bias.



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

2. model training

cross-entropy is a nice loss function which is information compression coding technique in information theory.

$H_{y'}(y) = -\sum_i y'_i log(y_i)$

Where y is the predicted value and y_ is the precise value.

Once the optimizer is set, TensorFlow would automatically calculate the gradient and optimizer the model.

3. model estimating

Use tf.argmax – a useful function the return the index of the maximum of certain tensors object – to check if the prediction is correct.

Then compute the accuracy by tf.reduce_mean.

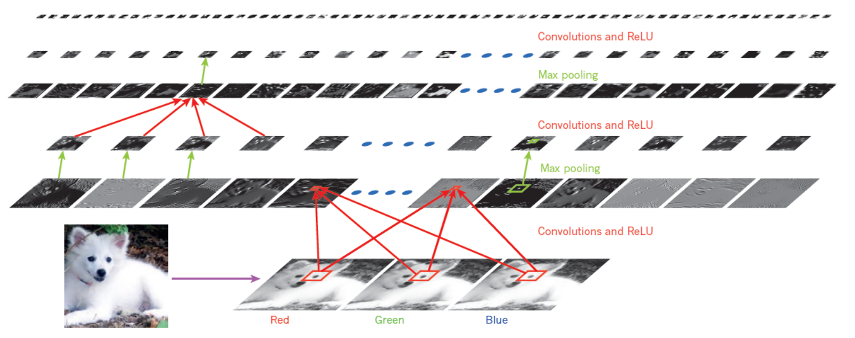
```
mnist = input_data.read_data_sets("MNIST_data/")
x = tf.placeholder("float", [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder("float", [None, 10])
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).
                              minimize(cross_entropy)
init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_:
                                  batch_ys})
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_
                              , 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "
                              float"))
print(sess.run(accuracy, feed_dict={x: mnist.test.images,
                              y_: mnist.test.labels}))
```

```
D:\ProgramData\Anaconda3\lib\site-packages\h5py\__init__.py:36: F
  from ._conv import register_converters as _register_converters
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting MNIST_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:\Users\cooma\AppData\Roaming\Python\Pyt
Instructions for updating:
Use `tf.global_variables_initializer` instead.
2018-12-26 01:01:46.556012: I tensorflow/core/platform/cpu_featur
0.9147

Process finished with exit code 0
```

Figure: Output

A CNN-besed model may further improve the accuracy of handwritten digits recognition.
There are some key ideas behind ConvNets that take advantages of natural signals:

- Local connections.
    - Local groups of values are often highly correlated.
    - Conv op is efficient algorithm to make use of the correlation.
- Pooling.
    - To merge semantically similar features into one.
    - A typical pooling unit computes the maximum of a local patch of units in one feature map(such as conv op), hence the name — maximum pooling.
- The use of many layers.

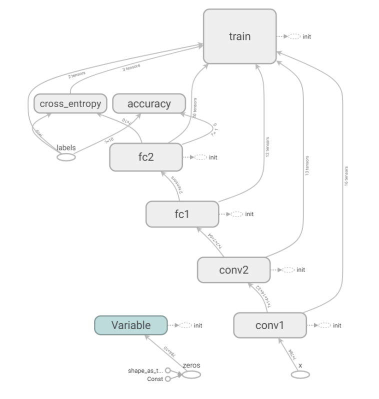

Convolutions and ReLU
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

```
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
sess = tf.InteractiveSession()
x = tf.placeholder("float", shape=[None, 784], name="x")
y_ = tf.placeholder("float", shape=[None, 10], name="labels")
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial, name="W")
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial, name="B")
def conv2d(x1, W1):
    return tf.nn.conv2d(x1, W1, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x2):
    return tf.nn.max_pool(x2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
# convolutional layer
```

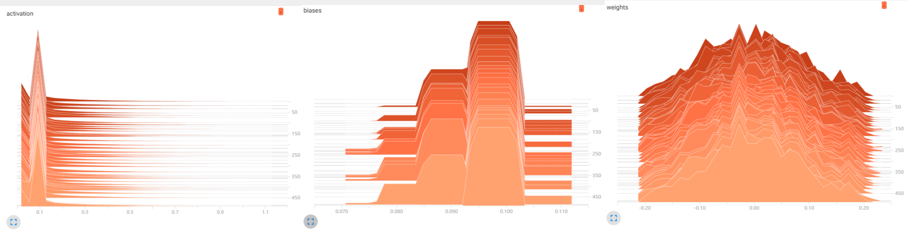
```python
with tf.name_scope(name="conv1"):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    x_image = tf.reshape(x, [-1, 28, 28, 1])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
    h_pool1 = max_pool_2x2(h_conv1)

with tf.name_scope(name="conv2"):
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
    h_pool2 = max_pool_2x2(h_conv2)
# fully-connected layer
with tf.name_scope(name="fc1"):
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])
    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

with tf.name_scope(name="fc2"):
```

```
    keep_prob = tf.placeholder("float")
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])
    y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) +
                              b_fc2)
#draw histograms
tf.summary.histogram("weights", W_conv1)
tf.summary.histogram("weights", W_conv2)
tf.summary.histogram("biases", b_conv2)
tf.summary.histogram("biases", b_conv1)
tf.summary.histogram("activation", h_conv1)
tf.summary.histogram("activation", h_conv2)
with tf.name_scope(name="cross_entropy"):
    cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
tf.summary.scalar('cross_entropy', cross_entropy)
with tf.name_scope(name="train"):
    train_step = tf.train.AdamOptimizer(1e-4).minimize(
                              cross_entropy)
with tf.name_scope(name="accuracy"):
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.
                              argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "
```

```
                                    float"))
tf.summary.scalar('accuracy', accuracy)
sess.run(tf.initialize_all_variables())
writer = tf.summary.FileWriter("/tmp/log/mnist")
writer.add_graph(sess.graph)
for i in range(500):
    batch = mnist.train.next_batch(50)
    if i % 100 == 0:
        train_accuracy = accuracy.eval(feed_dict={x: batch[
                                        0], y_: batch[1],
                                        keep_prob: 1.0})
        print("step %d, training accuracy %g" % (i,
                                        train_accuracy))
    train_step.run(feed_dict={x: batch[0], y_: batch[1],
                                    keep_prob: 0.5})
writer.close()
print("test accuracy %g" % accuracy.eval(feed_dict={x:
                              mnist.test.images, y_: mnist.
                              test.labels, keep_prob: 1.0})
                              )
```



train
init
2 tensors
3 tensors
cross_entropy
accuracy
labels
fc2
init
fc1
init
conv2
init
Variable
init
conv1
init
zeros
shape_as_t...
Const
x

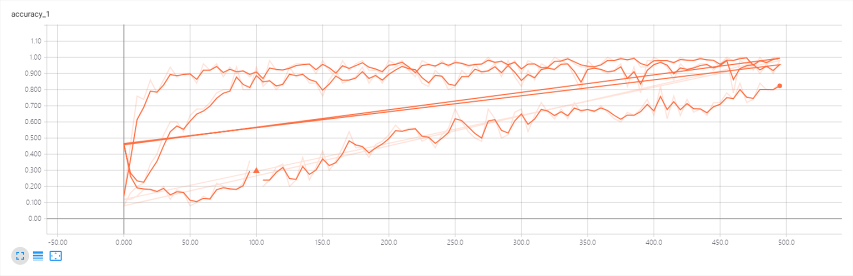


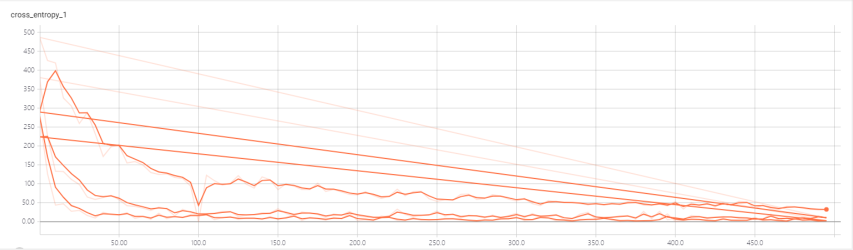
activation
biases
weights

A Game of arguments adjusting?

```
for learning_rates in [1e-3, 1e-4, 1e-5]:
    with tf.name_scope(name="train"):
        train_step = tf.train.AdamOptimizer(learning_rates)
                                          .minimize(
                                          cross_entropy)
    sess.run(tf.initialize_all_variables())
    merged_summary = tf.summary.merge_all()
    writer = tf.summary.FileWriter("/tmp/log/mnist")
    writer.add_graph(sess.graph)
    for i in range(500):
        batch = mnist.train.next_batch(50)
        if i % 5 == 0:
            s = sess.run(merged_summary, feed_dict={x:
                                              batch[0], y_:
                                              batch[1],
                                              keep_prob: 1.0})
            writer.add_summary(s, i)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={x:
                                              batch[0], y_:
                                              batch[1],
```

```
                                  keep_prob: 1.0})
         print("step %d, training accuracy %g" % (i,
                                  train_accuracy))
      train_step.run(feed_dict={x: batch[0], y_: batch[1]
                               , keep_prob: 0.5})
   writer.close()
   print("test accuracy %g" % accuracy.eval(feed_dict={x:
                            mnist.test.images, y_:
                            mnist.test.labels,
                            keep_prob: 1.0}))
```


accuracy_1
1.10
1.00
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.00
-50.00
0.000
50.00
100.0
150.0
200.0
250.0
300.0
350.0
400.0
450.0
500.0


cross_entropy_1
500
450
400
350
300
250
200
150
100
50.0
0.00
50.00
100.0
150.0
200.0
250.0
300.0
350.0
400.0
450.0

Thank you!