Practice Room Account

- https://goo.gl/i7cdZa
- Change your password with passwd command
IPython Notebook

• Run following command in the terminal
  • jupyter notebook

• Type following URL in your browser
  • http://localhost:8888/
IPython Notebook
```python
import tensorflow as tf
import numpy as np

# Create 100phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.randn(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but TensorFlow will)
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1, 10], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.squared_difference(y, y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()  

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in range(20):  
    sess.run(train)  
    if step % 20 == 0:  
        print(step, sess.run(W), sess.run(b))

# Learn best fit is W: [ 0.1], b: [0.3]
sess.close()
```

Get Started
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

• np.random.rand(100) Returns a vector of 100 random values ranged [0, 1)
• astype(np.float32) NumPy uses float64 as default, so we change the type to float32 which is the optimized data type in GPU.
• x_data * 0.1 Performs element-wised multiplication.
# Try to find values for W and b that compute \( y_{data} = W \times x_{data} + b \)
# (We know that W should be 0.1 and b 0.3, but TensorFlow will figure that out for us.)
\[
W = \text{tf.Variable}(\text{tf.random_uniform([1], -1.0, 1.0)})
\]
\[
b = \text{tf.Variable}(\text{tf.zeros([1]))}
\]
\[
y = W \times x_{data} + b
\]

- \text{tf.Variable(<initial-value>, ...)} Creates a variable in the graph.
- \text{tf.random_uniform([1], -1.0, 1.0)} Creates a tensor (not a value) which will generate a value shaped [1] from uniform distribution [-1, 1].
- \text{tf.zeros([1])} Creates a constant tensor (not a value) with all elements set to zero. [1] is shape.
- \text{y} is not a specific value but a symbolic variable defined by structured expressions.
Get Started

```python
# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```

- `tf.square(...)` Computes square of x element-wise.
- `tf.reduce_mean(...)` Computes the mean of elements across dimensions of a tensor.
- `tf.train.GradientDescentOptimizer(...)` Construct a new gradient descent optimizer.
- `optimizer.minimize(loss)` Calling minimize() takes care of both computing the gradients and applying them to the variables. It returns an operation object that updates trainable variables using gradient descent.
Get Started

```python
# Before starting, initialize the variables. We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)
```

- **tf.Session()** A **Session** object encapsulates the environment in which **Operation** objects are executed, and **Tensor** objects are evaluated.
- **sess.run(init)** **init** is an operation object. `sess.run(...)` executes operations or evaluate values of tensor objects.
# Fit the line.
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

sess.close()

• for step in range(201): Iterate from 0 to 200.
• train is an operation object.
• W and b are tensor objects.
MNIST Classification

```python
In [2]: import tensorflow as tf

In [3]: from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz

In [4]:

with tf.Session() as sess:
    # Define placeholders
    x = tf.placeholder(tf.float32, [None, 784])
    y_ = tf.placeholder(tf.float32, [ None, 10])

    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))

    y = tf.nn.softmax(tf.matmul(x, W) + b)

    # Define loss and optimizer
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
    train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

    # Train
    tf.initialize_all_variables().run()

    for i in range(1000):
        batch_size = mnist.train.next_batch(100)
        train_step.run(feed_dict={x: batch_size[0], y_: batch_size[1]})

    # Test trained model
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print('accuracy eval: %s' % accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))

0.92
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

• one_hot Represents labels as vectors of zeros and ones where only one of the entries is one.
  • 0 -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  • 1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  • 2 -> [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
  • ...
MNIST Classification – With

```python
with tf.Session() as sess:
    ...
```

- The above code is similar to the following:

```python
sess = tf.Session()
try:
    ...

    catch something
    sess.close()

    sess.close()
```
# Create the model \( y = xW + b \)

\[
x = \text{tf.placeholder}(\text{tf.float32}, [\text{None}, 784]) \\
W = \text{tf.Variable}(\text{tf.zeros}([784, 10])) \\
b = \text{tf.Variable}(\text{tf.zeros}([10])) \\
y = \text{tf.nn.softmax}(\text{tf.matmul}(x, W) + b)
\]

- **tf.placeholder** Placeholders are tensors that user feed specific values when running the session.
- **[None, 784]** In this case, we put None to allow any size of the mini-batch.
- **tf.matmul(x, W)** Computes dot product. The shape of the result is [None, 10].
- **... + b** It adds bias vector for each instance (broadcasting).
- **tf.nn.softmax(some_matrix)** Computes softmax for each instance (row).

Quiz: What is the shape of \( y \)?
MNIST Classification – Loss and Optimizer

```python
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

- `y_` It is true output of the model.
- `tf.reduce_sum(some_matrix, reduction_indices=[1])` Computes summation per row.
- `tf.log(y + 1E-14)` would be more stable than `tf.log(y)`. 
MNIST Classification – Train

```python
# Train
tf.initialize_all_variables().run()
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    train_step.run({x: batch_xs, y_: batch_ys})
```

- `.run()` We can execute operators by calling run() when session is bounded.
- `run({x: batch_xs, y_: batch_ys})` Always don’t forget to feed specific values to the placeholders.
# Test trained model

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))

• `tf.argmax(y, dimension=1)` Finds an index of maximum value for each row.
• `tf.equal(x1, x2)` Returns the truth value of \((x1 == x2)\) element-wise.
• `tf.cast(correct_prediction, tf.float32))` We cast the data type from bool to float32.
• `tensor_object.eval(...)` Evaluates value of the tensor object.

Quiz: What is the shape of `correct_prediction`?
Bonus – Draw Numbers

In [11]:
import matplotlib.pyplot as plt
%matplotlib inline

img = mnist.test.images[0]
print(img.shape)
img = img.reshape(28, 28)
plt.imshow(img, cmap='gray')

Out[11]:
<matplotlib.image.AxesImage at 0x7fe47f2e6a50>
Bonus – Putting a Hole

In [12]:

    hole_img = img
    hole_img[8:8+7, 10:10+7] = 0
    plt.imshow(hole_img, cmap='gray')

Out[12]: <matplotlib.image.AxesImage at 0x7fe47b962110>