0. how tf works

In [ ]:

```
import tensorflow as tf
import numpy as np

data = np.array([[1,2,3],[4,5,6]])

# computational graph
x1 = tf.placeholder(tf.float32, [1,3])
x2 = tf.placeholder(tf.float32, [3,1])
product = tf.matmul(x1, x2)

with tf.Session() as sess:
    print(sess.run(product, feed_dict={x1:data[0].reshape(1,3), x2:data[1].reshape(3,1)}))
# note: reshape is necessary because (3,) can not work
```

1, how to use tf to do optimization

In [2]:

```
data = np.array([2, 2]).reshape(1,-1)

# computational graph
x = tf.placeholder(tf.float32, [1, None]) # None means dont know
w = tf.Variable(tf.random_normal([1, 2]))
cost = tf.matmul(x-w, tf.transpose((x-w), [1,0]))

# more complex computational graph -> a training step
optimizer = tf.train.AdamOptimizer(0.1)
train_step = optimizer.minimize(cost)

with tf.Session() as sess:
    # since we use Variable we need to initialize it
    sess.run(tf.initializers.global_variables())

    for _ in range(100):
        sess.run(train_step, feed_dict={x: data})
        print(sess.run(cost, feed_dict={x: data}))
    print('\nw suppose to be [2,2], we got: ', sess.run(w))
```

```
...
```

bonus

In [3]:

```
# constant

c1 = tf.constant(1)
c2 = tf.constant(2)
c3 = tf.constant(3)

add = tf.add(c1, c2)
ma = tf.multiply(add, c3)

# m1 = tf.constant([[3,3]])
# m2 = tf.constant([[2],[3]])
```

In [6]:

```
p1 = tf.placeholder(tf.float32)  # tensor
p2 = tf.placeholder(tf.float32)
state = tf.multiply(p1, p2)        # elementwise

a=1
b=2
# pic = tf.placeholder(tf.float32, [None, n_dim])
# w1 = tf.Variable(tf.random_normal([n_dim, nb_n_1]))
update = tf.assign(state, tf.add(state, 1))  # only variable can be assigned. Constant can not be as
```

...

In [7]:

```
# fetch and feed

with tf.Session() as sess:
    # fetch
    print(sess.run([add, ma]))

    # feed
    print(sess.run(state, feed_dict={p1: [a], p2: [b]}))
```

```
[3, 9]
[2.]
```

build a NN

theoritical one

In [ ]:

```
# assume we already have data, and we ignore the feed for the train
# so the following programming can not work without some concret cases

x_train =
y_train =

# assume the x_train is (1000, 5) data , y_train is (1000,1)
x = tf.placeholder(tf.float32, [None, 5])
y = tf.placeholder(tf.float32, [None, 1])

n1 = 4
w1 = tf.Variable(tf.random_normal([5, n1]))  # it is transpose of the form in textbook of NN
b1 = tf.Variable(tf.zeros([1, n1])) # b1 is (1,4) because the output is 1 dimensional
z1 = tf.matmul(x, w1) + b1 # here broadcast will happen
a1 = tf.nn.relu(z1)

n2 = 1
w2 = tf.Variable(tf.random_normal([n1, 1]))
b2 = tf.Variable(tf.zeros([1, n2]))
z2 = tf.matmul(a1,w1) + b2
a2 = z2

loss = tf.reduce_mean(tf.square(a2-y))

optimizer = tf.train.GradientDescentOptimizer(0.2)
train = optimizer.minimize(loss)

init = tf.initializers.global_variables()
with tf.Session() as sess:
    sess.run(init)
    for i in range(200):
        sess.run(train, feed_dict={x: x_train, y: y_train})
    print(sess.run(loss, feed_dict={x: x_train, y: y_train}))
```

real one

In [13]:

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

# new we going to build the NN; lets start with placeholder for data and parameter
pic             = tf.placeholder(tf.float32, [None, 784])
the_label       = tf.placeholder(tf.float32, [None, 10])
learning_rate   = tf.placeholder(tf.float32)

# layer 1
nb_n_1 = 500
w1 = tf.Variable(tf.truncated_normal([784, nb_n_1], stddev = 0.1))
b1 = tf.Variable(tf.zeros([nb_n_1]) + 0.1)
l1 = tf.nn.tanh(tf.matmul(pic, w1) + b1)
nb_n = nb_n_1

# layer 2
nb_n_2 = 300
w2 = tf.Variable(tf.truncated_normal([nb_n, nb_n_2], stddev = 0.1))
b2 = tf.Variable(tf.zeros([nb_n_2]) + 0.1)
l2 = tf.nn.tanh(tf.matmul(l1, w2) + b2)
nb_n = nb_n_2

# layer 3
nb_n_3 = 10
w3 = tf.Variable(tf.truncated_normal([nb_n, 10], stddev = 0.1))
b3 = tf.Variable(tf.zeros([nb_n_3]) + 0.1)
l3 = tf.nn.softmax(tf.matmul(l2, w3) + b3) # softmax layer takes no activation function

loss = tf.losses.softmax_cross_entropy(the_label, l3)
# loss = tf.reduce_mean(tf.square(the_label-l3))

optimizer =  tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(loss)

# ww also want to see some performance during the training
correct_predict = tf.equal(tf.argmax(l3, 1), tf.argmax(the_label, 1))
accuracy        = tf.reduce_mean(tf.cast(correct_predict, tf.float32))
# tf.cast make True to be 1

n_epoch = 5
batch_size = 5000
n_batch = mnist.train.num_examples // batch_size
# we need n_batch to write the loop
# in the loop we will use:
# batch_xs, batch_ys = mnist.train.next_batch(batch_size)

with tf.Session() as sess:
    sess.run(tf.initializers.global_variables())
    for epoch in range(n_epoch + 1):
        for batch in range(n_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            sess.run(train, feed_dict={
                pic:           batch_xs,
                the_label:     batch_ys,
                learning_rate: 0.001 * (0.98**epoch)}
                    )

            if epoch % 2 == 0:
                acc = sess.run(accuracy, feed_dict={pic: mnist.test.images, the_label: mnist.test.la
                print('{}%\tIteration {} : accuracy : {}'.format(float(epoch)*100/n_epoch, epoch, ac

tf.reset_default_graph()
```

...

In [ ]: