gh2.py

```python
# Experiments with deep learning neural nets

from our_mnist import *

import tensorflow as tf
mnist = read_data_sets('MNIST_data',one_hot = True)
sess = tf.InteractiveSession()

X = tf.placeholder(tf.float32, shape=[None, 784])

Y = tf.placeholder(tf.float32, shape=[None, 10])
#*******************************************************************
# DO NOT MODIFY THE CODE ABOVE THIS LINE
# MAKE CHANGES TO THE CODE BELOW:

# a method for initializing weights. Initialize to small random values
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

# a method for initializing bias. Initialize to 0.1
def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)

# first layer: fully connected ReLU with 1024 nodes

W_fc1 = weight_variable([784, 1024])
b_fc1 = bias_variable([1024])
v_fc1 = tf.nn.relu(tf.matmul(X, W_fc1) + b_fc1) # v_fc1 ?x1024

# second layer: fully connected ReLU with 512 nodes and dropouts

W_fc2 = weight_variable([1024, 20])
b_fc2 = bias_variable([20])
v_fc2 = tf.nn.relu(tf.matmul(v_fc1, W_fc2) + b_fc2) # v_fc2 ?x512

# dropouts for second layer
keep_prob = tf.placeholder(tf.float32)
v_fc2_drop = tf.nn.dropout(v_fc2, keep_prob) # v_fc2_drop ?x512

# third layer: readout Layer. Fully connected linear with 10 nodes
W_fc3 = weight_variable([20, 10])
b_fc3 = bias_variable([10])
v_fc3 = tf.matmul(v_fc2_drop, W_fc3) + b_fc3  # v_fc3 ?x10

predicted_Y = v_fc3;                          # predicted_Y ?x10

cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=Y, logits=predicted_Y))

# regularization parameters
```

```
lambda_1 = 0
lambda_2 = 0
lambda_3 = 0
regularizer = (
  lambda_1*tf.nn.l2_loss(W_fc1) +
  lambda_2*tf.nn.l2_loss(W_fc2) +
  lambda_3*tf.nn.l2_loss(W_fc3)
  )

loss = tf.reduce_mean(cross_entropy + regularizer)

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(loss)
sess.run(tf.global_variables_initializer())

print("Starting␣Training...")
print("epoch\ttrain_accuracy\ttest_accuracy")

for i in range(3000):
    batch = mnist.train.next_batch(100)
    if i % 100 == 0:
        correct_prediction = tf.equal(tf.argmax(predicted_Y,1), tf.argmax(Y,1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        test_accuracy = accuracy.eval(feed_dict={
            X: mnist.test.images,
            Y: mnist.test.labels,
            keep_prob: 1})
        train_accuracy = accuracy.eval(feed_dict={
            X: mnist.train.images,
            Y: mnist.train.labels,
            keep_prob: 1})

        print("%d␣\t␣%.3f␣\t\t␣%.2f" % (i, train_accuracy, test_accuracy))
    # TRAIN STEP
    train_step.run(feed_dict={
             X: batch[0],
             Y: batch[1],
             keep_prob:0.5})
#end for loop

correct_prediction = tf.equal(tf.argmax(predicted_Y,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
test_accuracy = accuracy.eval(feed_dict={
            X: mnist.test.images,
            Y: mnist.test.labels,
            keep_prob:1})

print("test␣accuracy:␣", test_accuracy)
```