Adam, NN, ReLu, Xavier and Dropout

Lecture 10

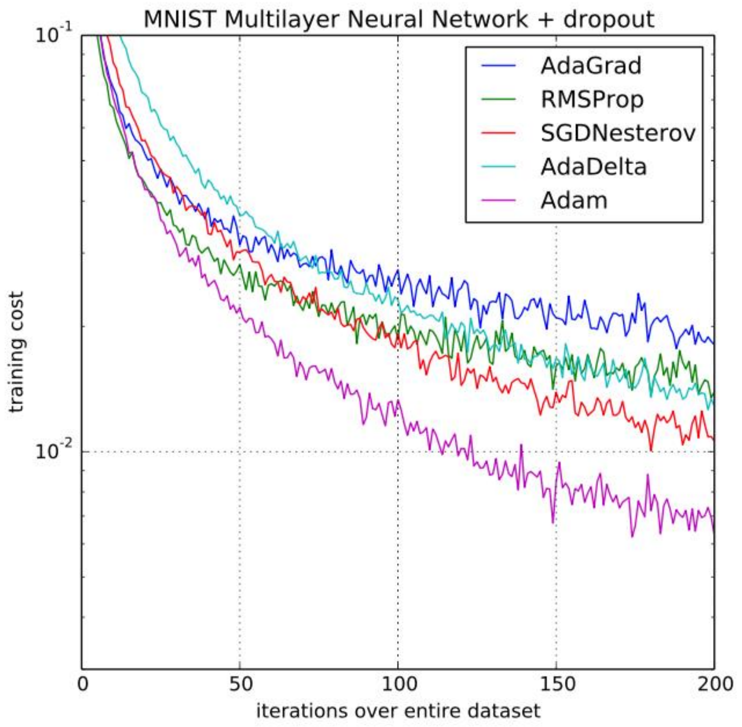
Optimizers

Optimizers

```
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer

ADAM: a method for stochastic optimization [Kingma et al. 2015]



Use Adam Optimizer

```
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
                                                       logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Sigmoid classifier for MNIST

ex10_1.ipynb (1/2)

Classifier for MNIST

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W = tf.Variable(tf.random_normal([784, 10]))
b = tf.Variable(tf.random_normal([10]))

hypothesis = tf.matmul(X, W) + b

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

ex10_1.ipynb (2/2)

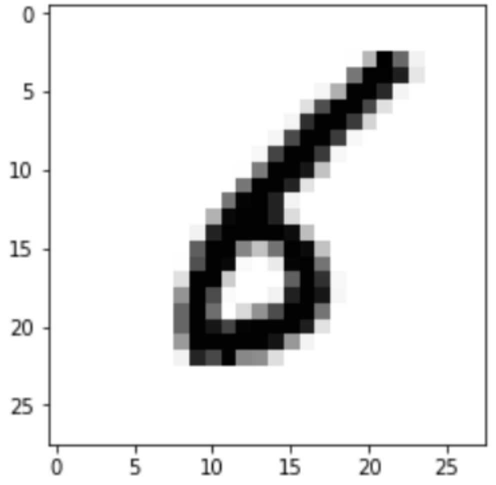
Classifier for MNIST

```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))
plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

ex10_1.ipynb (Notes)

Classifier for MNIST

```
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
Epoch: 0001 cost = 5.896178008
Epoch: 0002 cost = 1.807531157
Epoch: 0003 cost = 1.197546624
Epoch: 0004 cost = 0.953373779
Epoch: 0005 cost = 0.813679163
Epoch: 0006 cost = 0.720912020
Epoch: 0007 cost = 0.652785487
Epoch: 0008 cost = 0.600560204
Epoch: 0009 cost = 0.559620006
Epoch: 0010 cost = 0.526828700
Epoch: 0011 cost = 0.499622431
Epoch: 0012 cost = 0.476536487
Epoch: 0013 cost = 0.457392101
Epoch: 0014 cost = 0.440410323
Epoch: 0015 cost = 0.425807556
Learning Finished!
Accuracy: 0.8951
Label:  [6]
Prediction:  [6]
```



Softmax classifier for MNIST

ex10_2.ipynb

Softmax classifier for MNIST

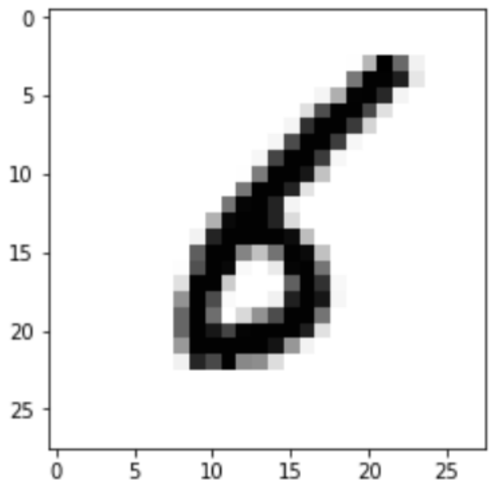
```
import tensorflow as tf
import random
import matplotlib.pyplot as plt
tf.set_random_seed(777)  # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))

# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
training_epochs = 15
batch_size = 100
print("train data 라인 수? ", mnist.train.num_examples)
print("test data 라인 수? ", mnist.test.num_examples)
```

```
with tf.Session() as sess:
   # Initialize TensorFlow variables
   sess.run(tf.global_variables_initializer())

   # Training cycle
   for epoch in range(training_epochs):
      avg_cost = 0
      total_batch = int(mnist.train.num_examples / batch_size)

      for i in range(total_batch):
         batch_xs, batch_ys = mnist.train.next_batch(batch_size)
         c, _ = sess.run([cost, optimizer], feed_dict={
                     X: batch_xs, Y: batch_ys})
         avg_cost += c / total_batch

      print('Epoch:', '%04d' % (epoch + 1),
          'cost =', '{:.9f}'.format(avg_cost))

   print("Learning finished")

   # Test the model using test sets
   print("Accuracy: ", accuracy.eval(session=sess,
          feed_dict={X: mnist.test.images, Y: mnist.test.labels}))

   # Get one and predict
   r = random.randint(0, mnist.test.num_examples - 1)
   print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r+1], 1)))
    rint("Prediction: ", sess.run(tf.argmax(hypothesis, 1),
          feed_dict={X: mnist.test.images[r:r+1]}))

    lt.imshow(mnist.test.images[r:r+1].reshape(28, 28), cmap='Greys',
             interpolation='nearest')
   plt.show()
```

ex10_2.ipynb (Notes)

Softmax classifier for MNIST

```
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
train data 라인 수?  55000
test data 라인 수?  10000
Epoch: 0001 cost = 2.894389681
Epoch: 0002 cost = 1.080790195
Epoch: 0003 cost = 0.865806108
Epoch: 0004 cost = 0.760368443
Epoch: 0005 cost = 0.693575929
Epoch: 0006 cost = 0.646156849
Epoch: 0007 cost = 0.609966863
Epoch: 0008 cost = 0.580936535
Epoch: 0009 cost = 0.556751010
Epoch: 0010 cost = 0.536568020
Epoch: 0011 cost = 0.518954003
Epoch: 0012 cost = 0.503147648
Epoch: 0013 cost = 0.489707803
Epoch: 0014 cost = 0.477875153
Epoch: 0015 cost = 0.466704109
Learning finished
Accuracy:  0.891
Label:  [6]
Prediction:  [6]
```



NN for MNIST

ex10_3.ipynb (1/2)

NN for MNIST (ReLu)

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
tf.set_random_seed(777)  # reproducibility
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers (3단으로 구성)
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.nn.softmax(tf.matmul(L2, W3) + b3)

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

ex10_3.ipynb (2/2)

NN for MNIST (ReLu)

```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

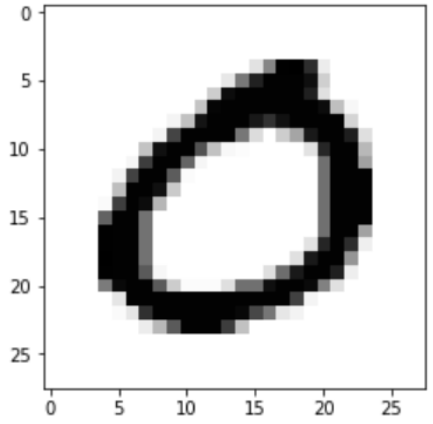
ex10_3.ipynb (Notes)

NN for MNIST (ReLu)

Softmax classifier for MNIST

```
Learning Finished!
Accuracy: 0.891
Label:  [6]
Prediction:  [6]
```

```
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
Epoch: 0001 cost = 157.733483412
Epoch: 0002 cost = 39.615378581
Epoch: 0003 cost = 24.751927294
Epoch: 0004 cost = 17.287412878
Epoch: 0005 cost = 12.414808827
Epoch: 0006 cost = 9.144174436
Epoch: 0007 cost = 6.906447330
Epoch: 0008 cost = 5.139310880
Epoch: 0009 cost = 3.913037534
Epoch: 0010 cost = 2.887293900
Epoch: 0011 cost = 2.229416456
Epoch: 0012 cost = 1.662174887
Epoch: 0013 cost = 1.319727440
Epoch: 0014 cost = 0.981397172
Epoch: 0015 cost = 0.756970026
Learning Finished!
Accuracy: 0.9409
Label:  [0]
Prediction:  [0]
```



Xavier for MNIST

ex10_4.ipynb (1/2)

Xavier for MNIST

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
tf.set_random_seed(777)  # reproducibility
tf.reset_default_graph() # 여러 번 실행할 경우, 주피터 노트북 환경에서는 리셋이 필요함
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
W1 = tf.get_variable("W1", shape=[784, 256],initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[256, 256],initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[256, 10],initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.nn.softmax(tf.matmul(L2, W3) + b3)

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

ex10_4.ipynb (2/2)

Xavier for MNIST

```python
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy (Xavier) :', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

ex10_4.ipynb (Notes)

Softmax classifier for MNIST

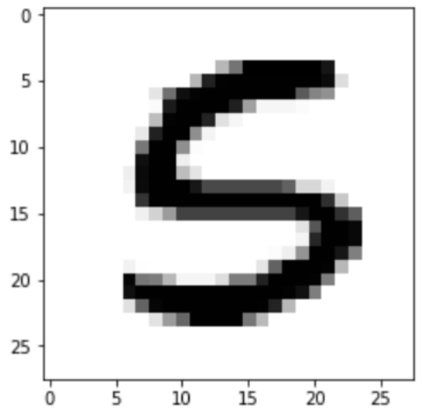
```
Learning Finished!
Accuracy: 0.891
Label:  [6]
Prediction:  [6]
```

NN for MNIST(ReLu)

```
Epoch: 0001 cost = 157.733483412
Epoch: 0002 cost = 39.615378581
Epoch: 0003 cost = 24.751927294
Epoch: 0004 cost = 17.287412878
Epoch: 0005 cost = 12.414808827
Epoch: 0006 cost = 9.144174436
Epoch: 0007 cost = 6.906447330
Epoch: 0008 cost = 5.139310880
Epoch: 0009 cost = 3.913037534
Epoch: 0010 cost = 2.887293900
Epoch: 0011 cost = 2.229416456
Epoch: 0012 cost = 1.662174887
Epoch: 0013 cost = 1.319727440
Epoch: 0014 cost = 0.981397172
Epoch: 0015 cost = 0.756970026
Learning Finished!
Accuracy: 0.9409
Label:  [0]
Prediction:  [0]
```

Xavier for MNIST

```
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
Epoch: 0001 cost = 0.288715202
Epoch: 0002 cost = 0.108830303
Epoch: 0003 cost = 0.070484336
Epoch: 0004 cost = 0.050766690
Epoch: 0005 cost = 0.037596774
Epoch: 0006 cost = 0.028400295
Epoch: 0007 cost = 0.025659041
Epoch: 0008 cost = 0.017986738
Epoch: 0009 cost = 0.015344570
Epoch: 0010 cost = 0.016166556
Epoch: 0011 cost = 0.013296543
Epoch: 0012 cost = 0.008216371
Epoch: 0013 cost = 0.012122193
Epoch: 0014 cost = 0.009940228
Epoch: 0015 cost = 0.009169801
Learning Finished!
Accuracy (Xavier) : 0.9803
Label:  [5]
Prediction:  [5]
```



Deep NN for MNIST

ex10_5.ipynb (1/3)

Deep NN for MNIST

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set random seed(777)  # reproducibility
tf.reset_default_graph() # 여러 번 실행할 경우, 주피터 노트북 환경에서는 리셋이 필요함

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])
```

ex10_5.ipynb (2/3)

Deep NN for MNIST

```
# weights & bias for nn layers (5단으로 구성)
# http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
W1 = tf.get_variable("W1", shape=[784, 512],initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[512, 512],initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[512, 512],initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)

W4 = tf.get_variable("W4", shape=[512, 512],initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)

W5 = tf.get_variable("W5", shape=[512, 10],initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis  = tf.nn.softmax(tf.matmul(L4, W5) + b5)

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

ex10_5.ipynb (3/3)

Deep NN for MNIST

```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

ex10_5.ipynb (Notes)

Deep NN for MNIST

Softmax classifier for MNIST

```
Learning Finished!
Accuracy: 0.891
Label:  [6]
Prediction:  [6]
```

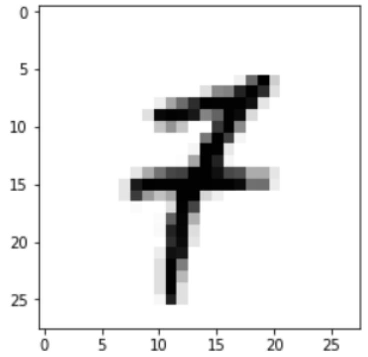
NN for MNIST(ReLu)

```
Learning Finished!
Accuracy: 0.9409
Label:  [0]
Prediction:  [0]
```

Xavier for MNIST

```
Learning Finished!
Accuracy (Xavier) : 0.9803
Label:  [5]
Prediction:  [5]
```

```
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
Epoch: 0001 cost = 0.293043509
Epoch: 0002 cost = 0.101662572
Epoch: 0003 cost = 0.069667759
Epoch: 0004 cost = 0.051410287
Epoch: 0005 cost = 0.041484293
Epoch: 0006 cost = 0.035476658
Epoch: 0007 cost = 0.030571641
Epoch: 0008 cost = 0.025109763
Epoch: 0009 cost = 0.023301485
Epoch: 0010 cost = 0.020631982
Epoch: 0011 cost = 0.020126167
Epoch: 0012 cost = 0.016526681
Epoch: 0013 cost = 0.017930981
Epoch: 0014 cost = 0.014518772
Epoch: 0015 cost = 0.013578182
Learning Finished!
Accuracy (Deep NN) : 0.9746
Label:  [7]
Prediction:  [7]
```



Over fitting ??
→ Dropout 필요..

Dropout for MNIST

ex10_6.ipynb (1/3)

Dropout for MNIST

```
import tensorflow as tf
import random
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
tf.set random seed(777)  # reproducibility
tf.reset_default_graph() # 여러 번 실행할 경우, 주피터 노트북 환경에서는 리셋이 필요함

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
total_batch = int(mnist.train.num_examples / batch_size)

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# dropout (keep_prob) rate  0.7 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)
```

ex10_6.ipynb (2/3)

Dropout for MNIST

```
# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
W1 = tf.get_variable("W1", shape=[784, 512],initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

W2 = tf.get_variable("W2", shape=[512, 512],initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)

W3 = tf.get_variable("W3", shape=[512, 512],initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)

W4 = tf.get_variable("W4", shape=[512, 512],initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

W5 = tf.get_variable("W5", shape=[512, 10],initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.nn.softmax(tf.matmul(L4, W5) + b5)

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

ex10_6.ipynb (3/3)

Dropout for MNIST

```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7} # training data에서 유지할 비율을 입력
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy (Dropout):', sess.run(accuracy, feed_dict={X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
                                                                                  # test : test data에서 100% 사용해야 함

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))
                                                                                  # test : test data에서 100% 사용해야 함
plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

ex10_6.ipynb (Notes)

Softmax classifier for MNIST

```
Learning Finished!
Accuracy: 0.891
Label:  [6]
Prediction:  [6]
```

NN for MNIST(ReLu)

```
Learning Finished!
Accuracy: 0.9409
Label:  [0]
Prediction:  [0]
```

Xavier for MNIST

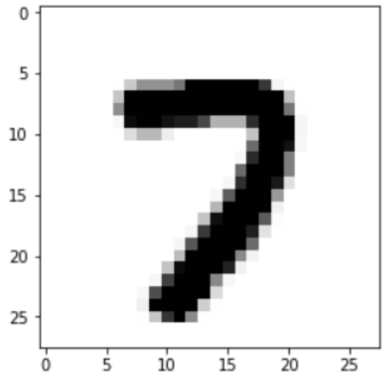
```
Learning Finished!
Accuracy (Xavier) : 0.9803
Label:  [5]
Prediction:  [5]
```

Deep NN for MNIST

```
Learning Finished!
Accuracy (Deep NN) : 0.9746
Label:  [7]
Prediction:  [7]
```

Dropout for MNIST

```
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
Epoch: 0001 cost = 0.456952580
Epoch: 0002 cost = 0.169334593
Epoch: 0003 cost = 0.130340332
Epoch: 0004 cost = 0.108162031
Epoch: 0005 cost = 0.092491395
Epoch: 0006 cost = 0.083232253
Epoch: 0007 cost = 0.073448378
Epoch: 0008 cost = 0.067860136
Epoch: 0009 cost = 0.065150652
Epoch: 0010 cost = 0.057729495
Epoch: 0011 cost = 0.056347424
Epoch: 0012 cost = 0.054186034
Epoch: 0013 cost = 0.048335011
Epoch: 0014 cost = 0.051020966
Epoch: 0015 cost = 0.044033949
Learning Finished!
Accuracy (Dropout): 0.9835
Label:  [7]
Prediction:  [7]
```



Summary

- Softmax
- Neural Nets
- Xavier initialization
- Deep Neural Nets with Dropout
- Adam and other optimizers