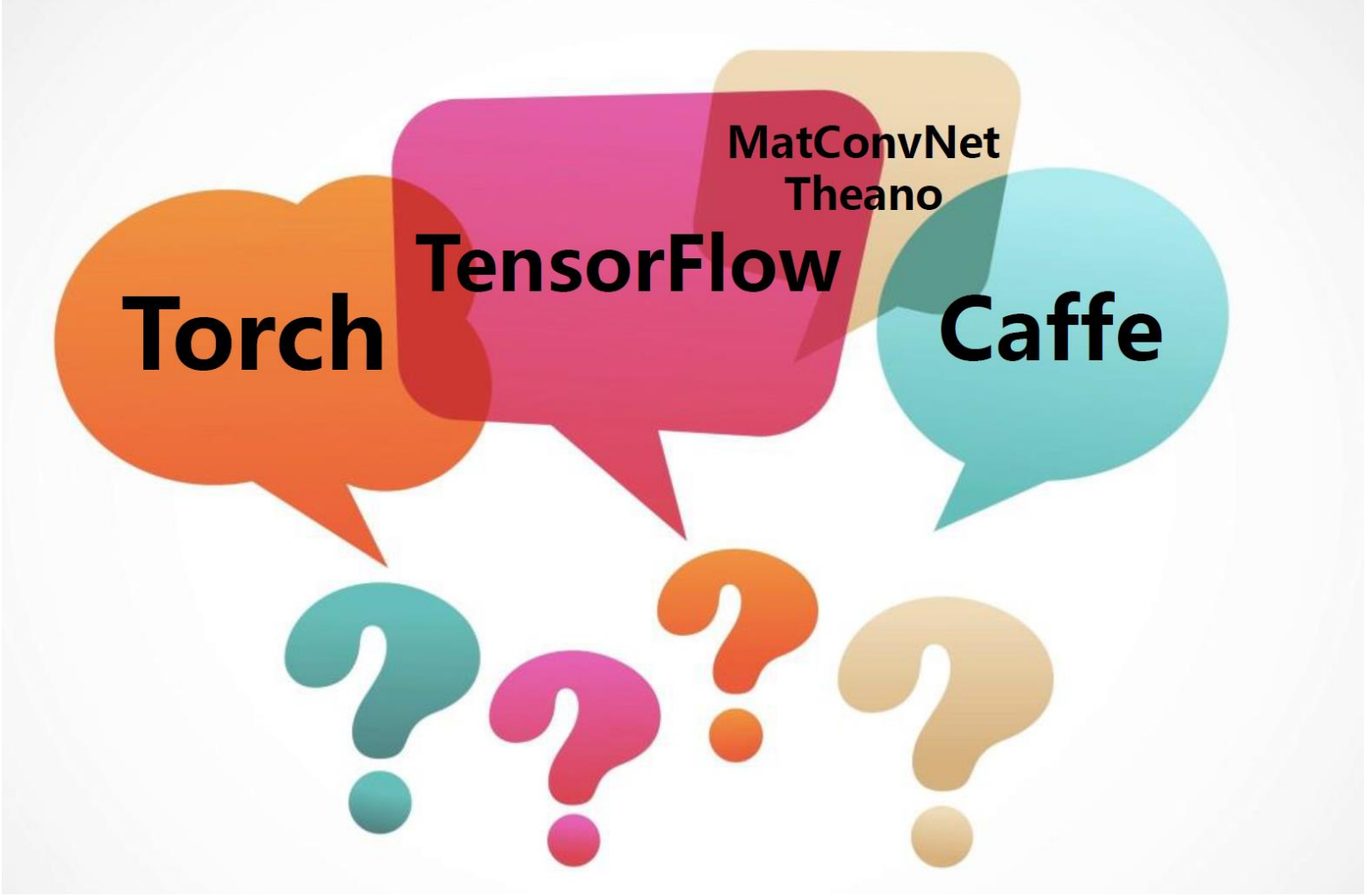
CS 6423
Scalable Computing for Data Analytics

Lecture 13: TensorFlow

Prof. Gregory Provan
Department of Computer Science
University College Cork

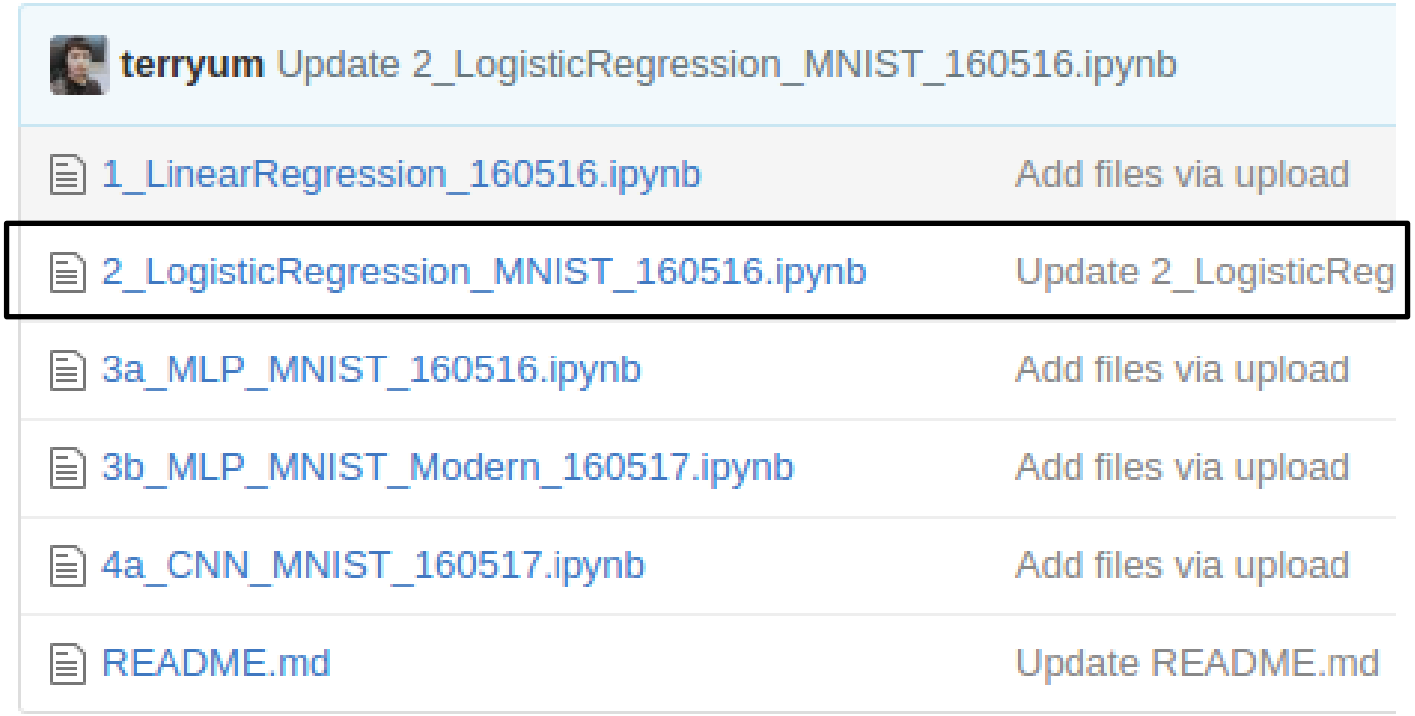
DEEP LEARNING LIBRARIES



BASIC WORKFLOW OF TF

1. Load data
2. Define the NN structure
3. Set optimization parameters
4. Run!

EXAMPLE 1



Google CoLab exercises

Other repositories:

https://github.com/terryum/TensorFlow_Exercises

1. LOAD DATA

https://github.com/terryum/TensorFlow_Exercises/blob/master/2_LogisticRegression_MNIST_160516.ipynb

2. Logistic Regression with MNIST data

In [1]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
#%matplotlib inline
```

Load MNIST data

In [2]:
```
mnist      = input_data.read_data_sets('data', one_hot=True)
X_train   = mnist.train.images
Y_train = mnist.train.labels
X_test    = mnist.test.images
Y_test  = mnist.test.labels
```

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

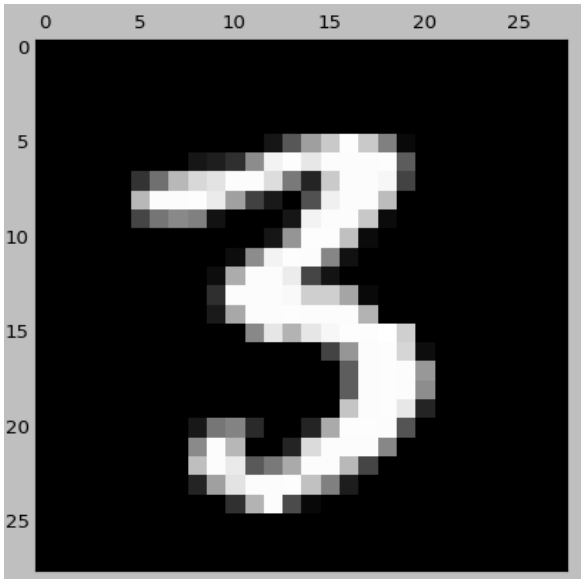
In [3]:
```
dimX = X_train.shape[1]
dimY = Y_train.shape[1]
nTrain = X_train.shape[0]
nTest = X_test.shape[0]
print ("Shape of (X_train, X_test, Y_train, Y_test)")
print (X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
Shape of (X_train, X_test, Y_train, Y_test)
((55000, 784), (10000, 784), (55000, 10), (10000, 10))
```

1. LOAD DATA

Plotan example image of MNIST data

In [4]:
```
myIdx = 36436   # any number
img   = np.reshape(X_train[myIdx, :], (28, 28)) # 28 * 28 = 784

plt.matshow(img, cmap=plt.get_cmap('gray'))
plt.show()
```



2. DEFINE THE NN STRUCTURE

Write a TF graph

In [5]:
```
X = tf.placeholder(tf.float32, [None, dimX], name="input")
Y= tf.placeholder(tf.float32, [None, dimY], name="output")
W = tf.Variable(tf.zeros([dimX, dimY]), name="weight")
b = tf.Variable(tf.zeros([dimY]), name="bias")
```

The output of the logic regression is $softmax(Wx + b)$

Note that the dimension of *Y_pred* is *(nBatch, dimY)*

In [6]:
```
Y_pred = tf.nn.softmax(tf.matmul(X, W) + b)
```

3. SET OPTIMIZATION PARAMETERS

In [7]:
```
loss = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(Y_pred), reduction_indices=1))
```

In [8]:
```
learning_rate = 0.005
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
training_epochs = 50
display_epoch = 5
batch_size = 100   # For each time, we will use 100 samples to update parameters
```

4. RUN

In [11]:
```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())

    for epoch in range(training_epochs):
        nBatch  = int(nTrain/batch_size)
        myIdx =  np.random.permutation(nTrain)
        for ii in range(nBatch):
            X_batch = X_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
            Y_batch = Y_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
            sess.run(optimizer, feed_dict={X:X_batch, Y:Y_batch})
```

```
(epoch 40)
[Loss / Training Accuracy] 0.3436 / 0.9047

(epoch 45)
[Loss / Training Accuracy] 0.3375 / 0.9066

(epoch 50)
[Loss / Training Accuracy] 0.3322 / 0.9078

[Test Accuracy]  0.9142
```

4. RUN (C.F.)

Compare prediction with the true value

argmax(X,1) returns the index of maximum value (which represents the label in this example) across the colums of the tensor *X*

In [9]:
```
correct_prediction = tf.equal(tf.argmax(Y_pred, 1), tf.argmax(Y, 1))
```

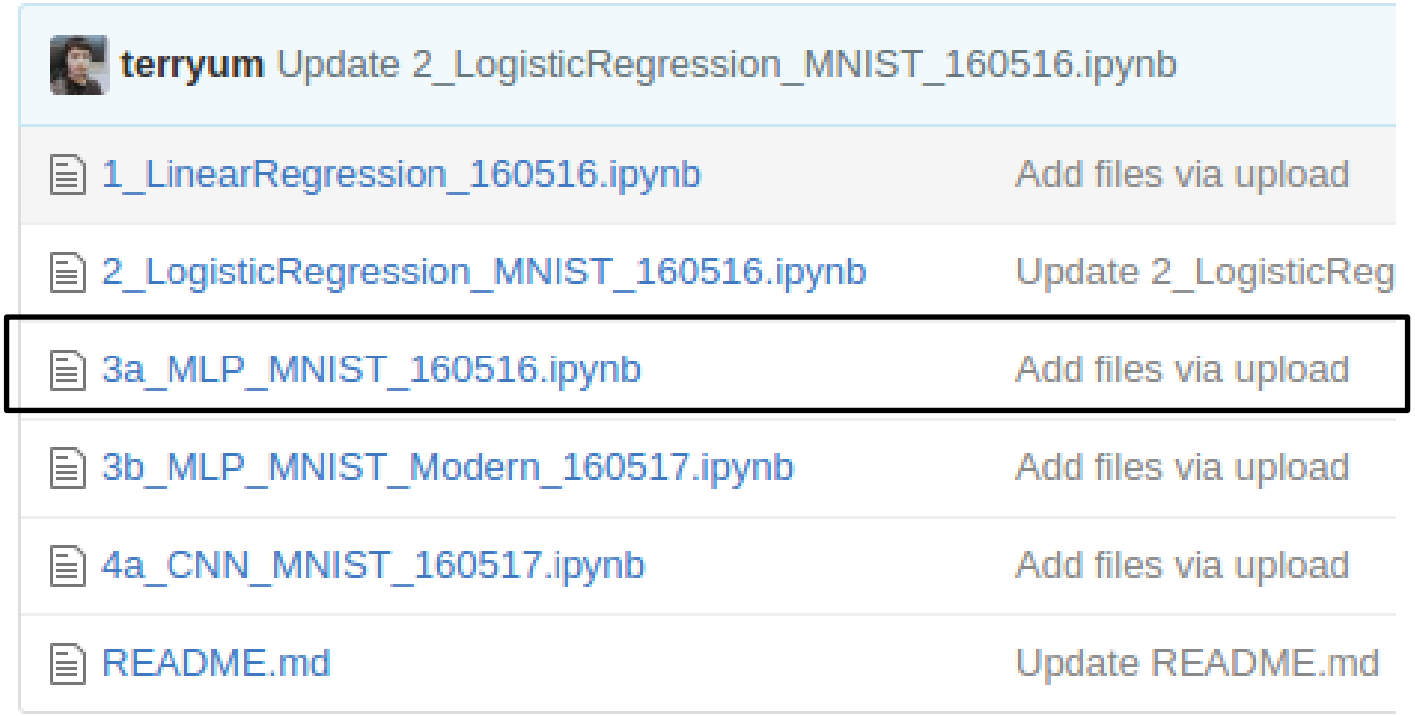
In [10]:
```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

Run the session

We use *with* for load a TF session

In [11]:
```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())

    for epoch in range(training_epochs):
        nBatch  = int(nTrain/batch_size)
        myIdx =  np.random.permutation(nTrain)
        for ii in range(nBatch):
            X_batch = X_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
            Y_batch = Y_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
            sess.run(optimizer, feed_dict={X:X_batch, Y:Y_batch})

        if (epoch+1) % display_epoch == 0:
            loss_temp = sess.run(loss, feed_dict={X: X_train, Y:Y_train})
            accuracy_temp = accuracy.eval({X: X_train, Y:Y_train})
            print "(epoch {})".format(epoch+1)
            print "[Loss / Training Accuracy] {:05.4f} / {:05.4f}".format(loss_temp, accuracy_temp)
            print " "

    print "[Test Accuracy] ",  accuracy.eval({X: X_test, Y: Y_test})
```

EXAMPLE 2



https://github.com/terryum/TensorFlow_Exercises

NEURAL NETWORK

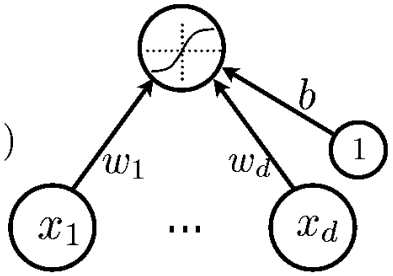
- **Basic NN structure**
  - Neuron pre-activation (or input activation):

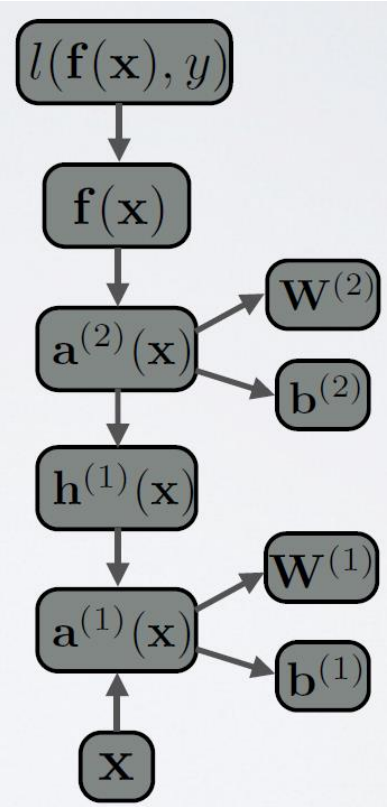
$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

  - Neuron (output) activation

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

  - $\mathbf{w}$ are the connection weights





- **Activation functions**

Name	Function $y=f(x)$	Derivative $\partial y / \partial x$
Logistic	$\frac{1}{1+e^{-x}}$	$y(1-y)$
Tanh	$Tanh(x)$	$1-y^2$
Gaussian	$e^{-x^2/2}$	$-xe^{-x^2/2}$
Linear	x	1

http://goo.gl/qMQk5H

1. LOAD DATA

https://github.com/terryum/TensorFlow_Exercises/blob/master/3a_MLP_MNIST_160516.ipynb

3. Multilayer Perceptron with MNIST data

In [1]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
#%matplotlib inline
```

Load MNIST data

In [2]:
```
mnist      = input_data.read_data_sets('data', one_hot=True)
X_train    = mnist.train.images
Y_train = mnist.train.labels
X_test     = mnist.test.images
Y_test   = mnist.test.labels
```

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

In [3]:
```
dimX = X_train.shape[1]
dimY = Y_train.shape[1]
nTrain = X_train.shape[0]
nTest = X_test.shape[0]
print ("Shape of (X_train, X_test, Y_train, Y_test)")
print (X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
Shape of (X_train, X_test, Y_train, Y_test)
((55000, 784), (10000, 784), (55000, 10), (10000, 10))
```

2. DEFINE THE NN STRUCTURE

Define my neural network structure

In [4]:
```
nLayer0 = dimX
nLayer1 = 256
nLayer2 = 256
nLayer3 =  dimY
sigma_init = 0.1   # For randomized initialization
```

In [7]:
```
W = {
    'W1': tf.Variable(tf.random_normal([nLayer0, nLayer1], stddev = sigma_init)),
    'W2': tf.Variable(tf.random_normal([nLayer1, nLayer2], stddev = sigma_init)),
    'W3': tf.Variable(tf.random_normal([nLayer2, nLayer3], stddev = sigma_init))
}
b = {
    'b1': tf.Variable(tf.random_normal([nLayer1])),
    'b2': tf.Variable(tf.random_normal([nLayer2])),
    'b3': tf.Variable(tf.random_normal([nLayer3]))
}
```

In [8]:
```
def model_myNN(_X, _W, _b):
    Layer1 = tf.nn.sigmoid(tf.add(tf.matmul(_X,_W['W1']), _b['b1']))
    Layer2 = tf.nn.sigmoid(tf.add(tf.matmul(Layer1,_W['W2']), _b['b2']))
    Layer3 = tf.add(tf.matmul(Layer2,_W['W3']), _b['b3'])
    #Layer3 = tf.nn.sigmoid(tf.add(tf.matmul(Layer2,_W['W3']), _b['b3']))
    return Layer3
```

In [9]:
```
X = tf.placeholder(tf.float32, [None, dimX], name="input")
Y= tf.placeholder(tf.float32, [None, dimY], name="output")
```

In [10]:
```
Y_pred = model_myNN(X, W, b)
```

3. SET OPTIMIZATION PARAMETERS

Define loss function, optimizer, measurer

We use *softmax_cross_entropy()* and *AdamOptimizer()*

In [9]:
```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(Y_pred, Y))
```

In [10]:
```
learning_rate = 0.001
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
#optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
training_epochs = 30
display_epoch = 5
batch_size = 100   # For each time, we will use 100 samples to update parameters
```

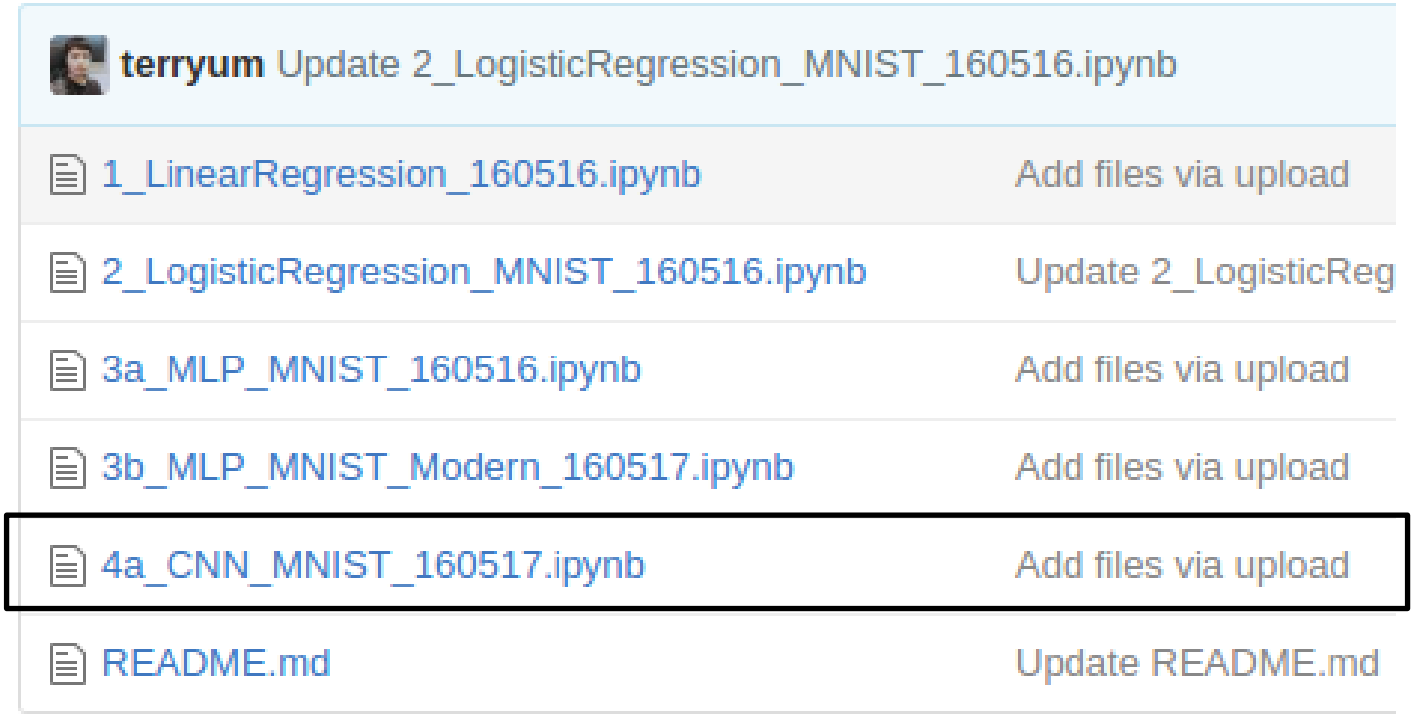
In [11]:
```
correct_prediction = tf.equal(tf.argmax(Y_pred, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

4. RUN

In [12]:
```
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())

    for epoch in range(training_epochs):
        nBatch  = int(nTrain/batch_size)
        myIdx =  np.random.permutation(nTrain)
        for ii in range(nBatch):
            X_batch = X_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
            Y_batch = Y_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
            #print X_batch.shape, Y_batch.shape
            sess.run(optimizer, feed_dict={X:X_batch, Y:Y_batch})

        if (epoch+1) % display_epoch == 0:
            loss_temp = sess.run(loss, feed_dict={X: X_train, Y:Y_train})
            accuracy_temp = accuracy.eval({X: X_train, Y:Y_train})
            print "(epoch {})".format(epoch+1)
            print "[Loss / Tranining Accuracy] {:05.4f} / {:05.4f}".format(loss_temp, accuracy_
temp)
            print " "

    print "[Test Accuracy] ",  accuracy.eval({X: X_test, Y: Y_test})
```

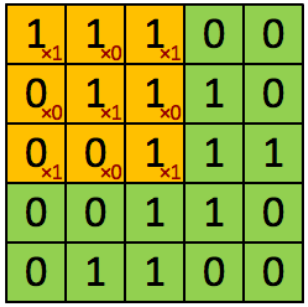
```
(epoch 45)
[Loss / Tranining Accuracy] 0.0000 / 1.0000

(epoch 50)
[Loss / Tranining Accuracy] 0.0002 / 1.0000

[Test Accuracy]  0.9799
```

EXAMPLE 3

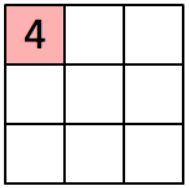


https://github.com/terryum/TensorFlow_Exercises

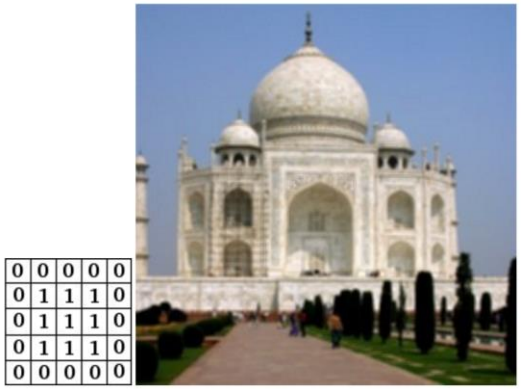
CONVOLUTION

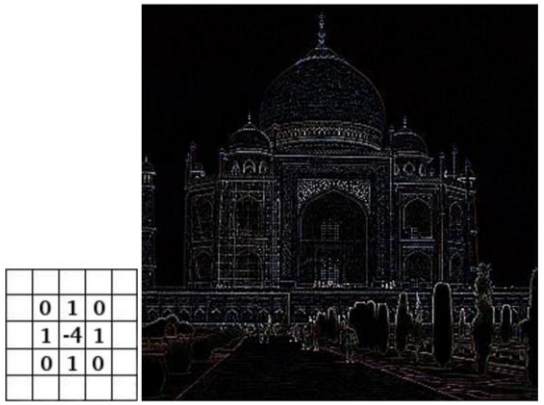


Image

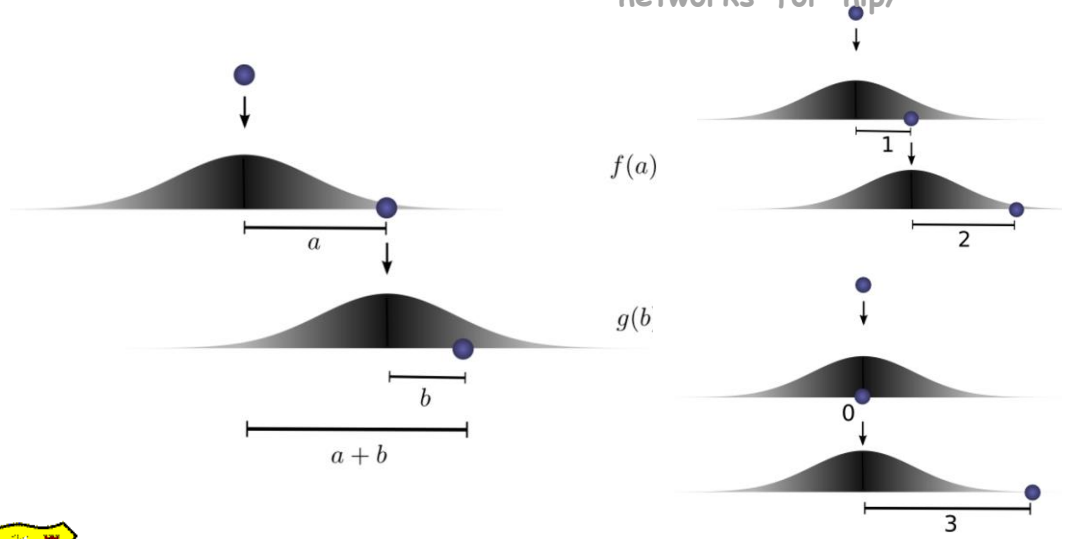


Convolved Feature





http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/



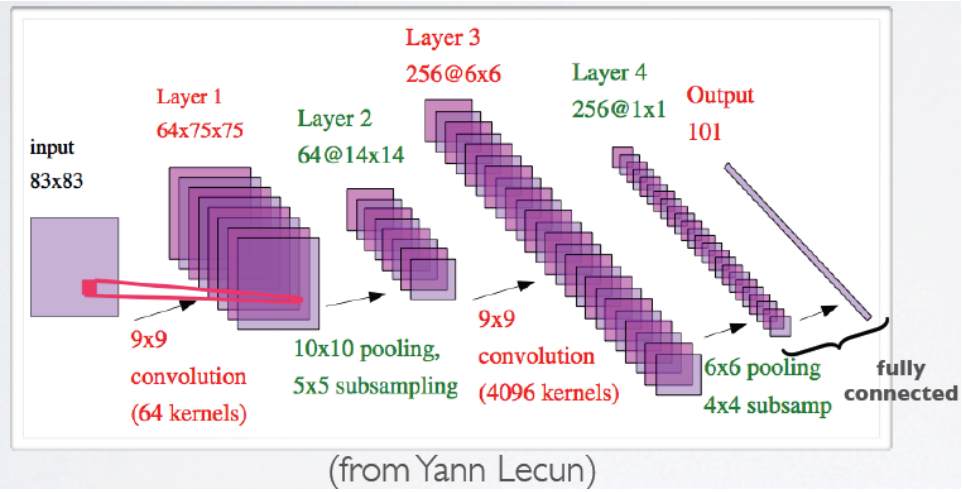
$$\ldots \quad f(0)\cdot g(3) \;+\; f(1)\cdot g(2) \;+\; f(2)\cdot g(1) \quad \ldots$$

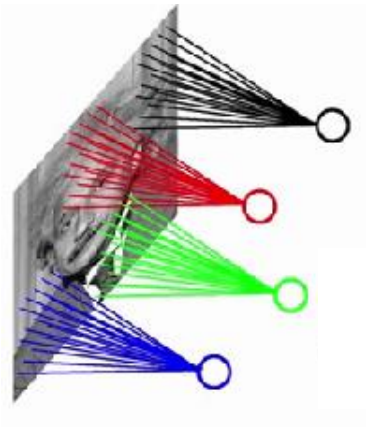
$$(f * g)(c) = \sum_{a+b=c} f(a) \cdot g(b)$$

$$(f * g)(c) = \sum_{a} f(a) \cdot g(c-a)$$

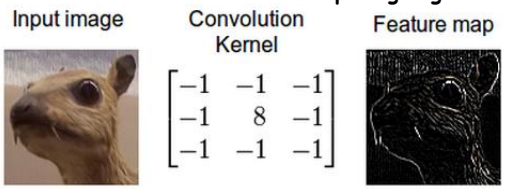
http://colah.github.io/posts/2014-07-Understanding-Convolutions/

CONVOLUTIONAL NN

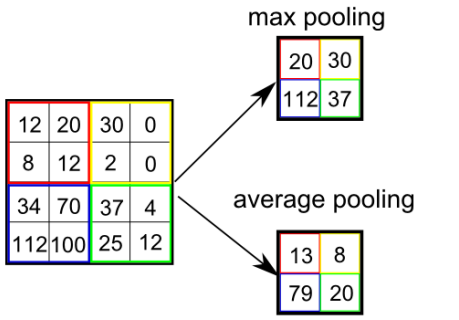




https://goo.gl/Xswsbd



- **How can we deal with real images which is much bigger than MNIST digit images?**
  - Use not fully-connected, but locally-connected NN
  - Use convolutions to get various feature maps
  - Abstract the results into higher layer by using pooling
  - Fine tune with fully-connected NN



http://goo.gl/5OR5oH

1. LOAD DATA

https://github.com/terryum/TensorFlow_Exercises/blob/master/4a_CNN_MNIST_160517.ipynb

4. Convolutional Neural Network with MNIST data

In [1]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
#%matplotlib inline
```

Load MNIST data

In [2]:
```
mnist      = input_data.read_data_sets('data', one_hot=True)
X_train   = mnist.train.images
Y_train = mnist.train.labels
X_test    = mnist.test.images
Y_test  = mnist.test.labels
```

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

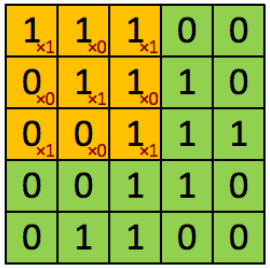
In [3]:
```
dimX = X_train.shape[1]
dimY = Y_train.shape[1]
nTrain = X_train.shape[0]
nTest = X_test.shape[0]
print ("Shape of (X_train, X_test, Y_train, Y_test)")
print (X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
Shape of (X_train, X_test, Y_train, Y_test)
((55000, 784), (10000, 784), (55000, 10), (10000, 10))
```

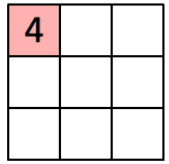
2. DEFINE THE NN STRUCTURE

Set parameters for my CNN structure

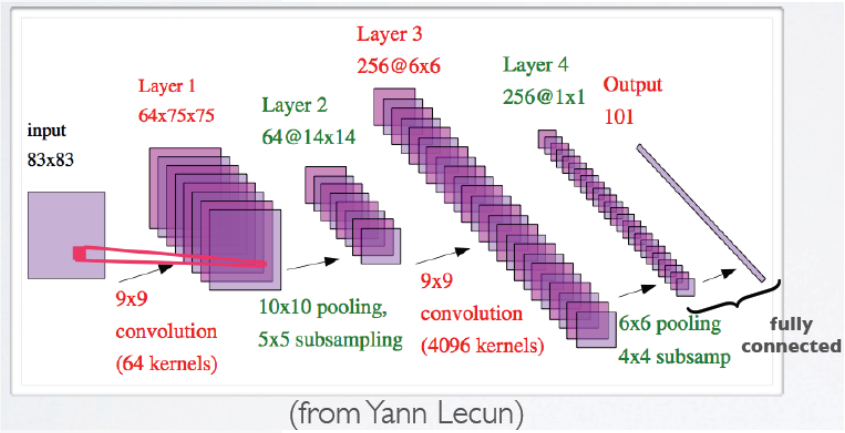
In [4]:
```
pp = {
    'nLayerIn': dimX,
    'nLayerOut':  dimY,
    'sigma_init': 0.1,
    'myDropProb': 0.7,

    'nWin_conv1': 3,
    'nStr_conv1': 1,
    'nPad_conv1': 'SAME', # or 'VALID'
    'nWin_pool1': 2,
    'nStr_pool1': 2,
    'nPad_pool1': 'SAME',
    'nFeat1': 64,

    'nWin_conv2': 3,
    'nStr_conv2': 1,
    'nPad_conv2': 'SAME',
    'nWin_pool2': 2,
    'nStr_pool2': 2,
    'nPad_pool2': 'SAME',
    'nFeat2': 128,

    'dimX_mat': 28,    # 28*28 = 784
    'nDimReduce': 7,   # dimX_mat/nWin_pool1/nWin_pool2
    'nFull': 1024
}
```



Image



Convolved Feature



(from Yann Lecun)

2. DEFINE THE NN STRUCTURE

In [5]:
```
W = {
    'W_conv1': tf.Variable(tf.truncated_normal([pp['nWin_conv1'], pp['nWin_conv1'], 1, pp['nFeat1']], stddev=pp['sig
    'W_conv2': tf.Variable(tf.truncated_normal([pp['nWin_conv2'], pp['nWin_conv2'], pp['nFeat1'], pp['nFeat2']], std
    'W_full': tf.Variable(tf.truncated_normal([pp['nDimReduce']*pp['nDimReduce']*pp['nFeat2'], pp['nFull']], stddev=
    'W_out': tf.Variable(tf.truncated_normal([pp['nFull'], pp['nLayerOut']], stddev=pp['sigma_init']))

}
b = {
    'b_conv1': tf.Variable(tf.truncated_normal([pp['nFeat1']], stddev=pp['sigma_init'])),
    'b_conv2': tf.Variable(tf.truncated_normal([pp['nFeat2']], stddev=pp['sigma_init'])),
    'b_full': tf.Variable(tf.truncated_normal([pp['nFull']], stddev=pp['sigma_init'])),
    'b_out': tf.Variable(tf.truncated_normal([pp['nLayerOut']], stddev=pp['sigma_init']))
}
```

In [6]:
```
def model_myCNN(_X, _W, _B, _dropout_prob, _pp):

    _X_mat = tf.reshape(_X, shape=[-1, _pp['dimX_mat'], _pp['dimX_mat'], 1])

    # L1: Convolution
    _L1_conv = tf.nn.relu(tf.nn.bias_add(
            tf.nn.conv2d(_X_mat, _W['W_conv1'], strides=[1, _pp['nStr_conv1'], _pp['nStr_conv1'], 1], padding=_pp['
            , _B['b_conv1']))
    _L1_pool = tf.nn.max_pool(_L1_conv, ksize=[1, _pp['nWin_pool1'], _pp['nWin_pool1'], 1], strides=[1, _pp['nStr_p
    _L1_pool2 = tf.nn.dropout(_L1_pool, _dropout_prob)

    # L2: Convolution
    _L2_conv = tf.nn.relu(tf.nn.bias_add(
            tf.nn.conv2d(_L1_pool2, _W['W_conv2'], strides=[1, _pp['nStr_conv2'], _pp['nStr_conv2'], 1], padding=_p
            , _B['b_conv2']))
    _L2_pool = tf.nn.max_pool(_L2_conv, ksize=[1, _pp['nWin_pool2'], _pp['nWin_pool2'], 1], strides=[1, _pp['nStr_p
    _L2_pool2 = tf.nn.dropout(_L2_pool, _dropout_prob)


    # L_full: Fully-connected
    _L2_pool2_vec = tf.reshape(_L2_pool2, [-1, _W['W_full'].get_shape().as_list()[0]])
    _L_full = tf.nn.relu(tf.add(tf.matmul(_L2_pool2_vec, _W['W_full']), _B['b_full']))
    _L_full2 = tf.nn.dropout(_L_full, _dropout_prob)

    # L_full: Output
    _L_out = tf.add(tf.matmul(_L_full2, _W['W_out']), _B['b_out'])

    # Return
    out = {
        'X_mat': _X_mat,
        'L1_conv': _L1_conv, 'L1_pool': _L1_pool, 'L1_pool2': _L1_pool2, # After dropout
        'L2_conv': _L2_conv, 'L2_pool': _L2_pool, 'L2_pool2': _L2_pool2,
        'L_full': _L_full, 'L_full2': _L_full2, 'L_out': _L_out
    }
    return out
```

3. SET OPTIMIZATION PARAMETERS

Define variables and optimizer

In [7]:
```
X = tf.placeholder(tf.float32, [None, dimX], name="input")
Y= tf.placeholder(tf.float32, [None, dimY], name="output")
dropout_prob = tf.placeholder(tf.float32, name="dropout")
```

In [8]:
```
Y_pred_all = model_myCNN(X, W, b, dropout_prob, pp)
Y_pred = Y_pred_all['L_out']
```

In [9]:
```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(Y_pred, Y))
learning_rate = 0.001
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
training_epochs = 5
display_epoch = 1
batch_size = 100   # For each time, we will use 100 samples to update ppeters
```

In [10]:
```
correct_prediction = tf.equal(tf.argmax(Y_pred, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

4. RUN

In [11]:
```
# Because of the memory allocation problem in evaluation
divide_train = 50;
divide_test = 10;
nTrainSub = (int)(nTrain/divide_train);
nTestSub = (int)(nTest/divide_test);
```

In [12]:
```
#with tf.Session() as sess:
sess = tf.InteractiveSession()
sess.run(tf.initialize_all_variables())

for epoch in range(training_epochs):
    nBatch  = int(nTrain/batch_size)
    #myIdx =  np.random.permutation(nTrain)
    for ii in range(nBatch):
        X_Batch, Y_Batch = mnist.train.next_batch(batch_size)
        #X_Batch = X_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
        #Y_Batch = Y_train[myIdx[ii*batch_size:(ii+1)*batch_size],:]
        sess.run(optimizer, feed_dict={X:X_Batch, Y:Y_Batch, dropout_prob:pp['myDropProb']})

    if (epoch+1) % display_epoch == 0:
        # Because of the memory allocation problem in evaluation
        loss_temp = accuracy_train_temp = accuracy_test_temp = 0
        for jj in range(divide_train):
            loss_temp += sess.run(loss, feed_dict={X: X_train[jj*divide_train:(jj+1)*divide_train,:], Y:Y_train[jj*
            accuracy_train_temp += accuracy.eval({X: X_train[jj*divide_train:(jj+1)*divide_train,:], Y:Y_train[jj*d
        for kk in range(divide_test):
            accuracy_test_temp += accuracy.eval({X: X_test[kk*divide_test:(kk+1)*divide_test,:], Y: Y_test[kk*divid

        print "(epoch {})".format(epoch+1)
        print "[Loss / Tranining Accuracy / Test Accuracy] {:05.4f} / {:05.4f} / {:05.4f}".format(loss_temp/divide_
        print " "

print "[Test Accuracy] {:05.4f}".format(accuracy_test_temp/divide_test)
```

```
(epoch 4)
[Loss / Tranining Accuracy / Test Accuracy] 0.0189 / 0.9940 / 1.0000

(epoch 5)
[Loss / Tranining Accuracy / Test Accuracy] 0.0155 / 0.9960 / 1.0000

[Test Accuracy] 1.0000
```

4. RUN (C.F.)

Let's see the learned features

In [13]:
```
nExample = 223

Y_pred_all = model_myCNN(X, W, b, dropout_prob, pp)
X_mat = sess.run(Y_pred_all['X_mat'], feed_dict={X: X_train[nExample-1:nExample, :]})
L1_conv   = sess.run(Y_pred_all['L1_conv'], feed_dict={X: X_train[nExample-1:nExample, :]})
```

In [14]:
```
# (nExample)th Input
plt.matshow(X_mat[0, :, :, 0], cmap=plt.get_cmap('gray'))
plt.colorbar()
plt.show()
```

In [15]:
```
# Features
nFeature = 40
plt.matshow(L1_conv[0, :, :, nFeature], cmap=plt.get_cmap('gray'))
plt.colorbar()
plt.show()
```

In [16]:
```
sess.close()
```