Tanmay Bhatt

011499072

CMPE 258
Assignmnt - 5
Date: 04/08/2018

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import h5py
```

1. (70pts) Define functions

Please define the functions which are needed for CNN architectures. The following list is a suggestion. One-hot encoding
Activation forward (Relu, sigmoid)
Compute cost
Zero pad
Convolution with single step
Convolution forward (for all data)
Pooling forward (max, average)
Pooling backward (max, average)
Activation backward (Relu, sigmoid)
Convolution backward (for all data)
Forward propagation (including all steps)
Backward propagation (including all steps)
Parameter updating (Gradient descent or other optimization method)

In [2]:
```
def sigmoid(z):
    return 1/(1 + np.exp(-z))
```

In [3]:
```
def sigmoid_derivative(a):
    return a * (1-a)
```

In [4]:
```
def ReLU_Derivative(a):
    a[a<=0] = 0
    return a
```

In [5]:
```
def ReLU(z):
    return z * (z > 0)
```

In [6]:
```
def zero_pad(X,p):
    return np.pad(X, [(0,0), (p,p), (p,p), (0,0)], 'constant', constant_
values=(0))
```

In [7]:
```
def flatten_array(X):
    temp = []
    for item in X:
        temp.append(item.flatten())
    return np.asarray(temp)
```

In [8]:
```
def one_hot_encoding(mat):
    list_of_list = []
    for i in range(0,len(mat)):
        small_list = np.zeros(np.max(mat)+1)
        small_list[mat[i]] = 1
        list_of_list.append(small_list)
    result = np.array(list_of_list)
    return result
```

In [9]:
```
def conv_single_step(a_slice_prev, W, b):

    s = np.multiply(a_slice_prev, W) + b
    Z = np.sum(s)

    return Z
```

In [10]:
```
def conv_forward(A_prev, W, b, hparameters,activation="sigmoid"):


    (m, n_H_prev, n_W_prev, n_C_prev) = A_prev.shape

    (f, f, n_C_prev, n_C) = W.shape

    stride = hparameters['stride']
    pad = hparameters['padding']

    n_H = int((n_H_prev - f + 2 * pad) / stride) + 1
    n_W = int((n_W_prev - f + 2 * pad) / stride) + 1

    Z = np.zeros((m, n_H, n_W, n_C))

    A_prev_pad = zero_pad(A_prev, pad)

    for i in range(m):
        a_prev_pad = A_prev_pad[i]
        for h in range(n_H):
            for w in range(n_W):
                for c in range(n_C):

                    vert_start = h * stride
                    vert_end = vert_start + f
                    horiz_start = w * stride
                    horiz_end = horiz_start + f
                    a_slice_prev = a_prev_pad[vert_start:vert_end, horiz
_start:horiz_end, :]
                    Z[i, h, w, c] = conv_single_step(a_slice_prev, W[...
,c], b[...,c])

    cache = (A_prev, W, b, hparameters)

    if activation == "sigmoid":
        a = sigmoid(Z)
    elif activation == "ReLU":
        a = ReLU(Z)

    return a, cache
```

In [11]:
```
def conv_backward(dZ, cache):


    (A_prev, W, b, hparameters) = cache

    (m, n_H_prev, n_W_prev, n_C_prev) = A_prev.shape

    (f, f, n_C_prev, n_C) = W.shape

    stride = hparameters["stride"]
    pad = hparameters["padding"]

    (m, n_H, n_W, n_C) = dZ.shape

    dA_prev = np.zeros((m, n_H_prev, n_W_prev, n_C_prev))

    dW = np.zeros((f, f, n_C_prev, n_C))
    db = np.zeros((1, 1, 1, n_C))

    A_prev_pad = zero_pad(A_prev, pad)
    dA_prev_pad = zero_pad(dA_prev, pad)

    for i in range(m):

        a_prev_pad = A_prev_pad[i]
        da_prev_pad = dA_prev_pad[i]

        for h in range(n_H):
            for w in range(n_W):
                for c in range(n_C):

                    vert_start = h
                    vert_end = vert_start + f
                    horiz_start = w
                    horiz_end = horiz_start + f

                    a_slice = a_prev_pad[vert_start:vert_end, horiz_star
t:horiz_end, :]

                    da_prev_pad[vert_start:vert_end, horiz_start:horiz_e
nd, :] += W[:,:,:,c] * dZ[i, h, w, c]
                    dW[:,:,:,c] += a_slice * dZ[i, h, w, c]
                    db[:,:,:,c] += dZ[i, h, w, c]

    return dA_prev, dW, db
```

In [12]:
```
def pool_forward(A_prev, hparameters, mode = "max"):

    (m, n_H_prev, n_W_prev, n_C_prev) = A_prev.shape

    f = hparameters["f"]
    stride = hparameters["stride"]

    n_H = int(1 + (n_H_prev - f) / stride)
    n_W = int(1 + (n_W_prev - f) / stride)
    n_C = n_C_prev

    A = np.zeros((m, n_H, n_W, n_C))

    for i in range(m):
        for h in range(n_H):
            for w in range(n_W):
                for c in range (n_C):

                    vert_start = h * stride
                    vert_end = vert_start + f
                    horiz_start = w * stride
                    horiz_end = horiz_start + f

                    a_prev_slice = A_prev[i, vert_start:vert_end, horiz_
start:horiz_end, c]

                    if mode == "max":
                        A[i, h, w, c] = np.max(a_prev_slice)
                    elif mode == "average":
                        A[i, h, w, c] = np.mean(a_prev_slice)


    cache = (A_prev, hparameters)

    return A, cache
```

In [13]:
```
def pool_backward(dA, cache, mode = "max"):


    (A_prev, hparameters) = cache

    stride = hparameters["stride"]
    f = hparameters["f"]

    m, n_H_prev, n_W_prev, n_C_prev = A_prev.shape
    m, n_H, n_W, n_C = dA.shape

    dA_prev = np.zeros(A_prev.shape)

    for i in range(m):
        a_prev = A_prev[i]
        for h in range(n_H):
            for w in range(n_W):
                for c in range(n_C):
                    vert_start = h
                    vert_end = vert_start + f
                    horiz_start = w
                    horiz_end = horiz_start + f

                    if mode == "max":
                        a_prev_slice = a_prev[vert_start:vert_end, horiz
_start:horiz_end, c]
                        mask = a_prev_slice == np.max(a_prev_slice)
                        dA_prev[i, vert_start:vert_end, horiz_start:hori
z_end, c] += np.multiply(mask, dA[i, h, w, c])

                    elif mode == "average":
                        da = dA[i, h, w, c]
                        shape = (f, f)
                        (n_H, n_W) = shape
                        dA_prev[i, vert_start:vert_end, horiz_start:hori
z_end, c] += np.ones(shape) * da / (n_H * n_W)

    return dA_prev
```

In [14]:
```
def forward_pass(X_mat):

    global neural_dict

    W1 = neural_dict['Fc1']['W']
    W2 = neural_dict['Fc2']['W']
    B1 = neural_dict['Fc1']['B']
    B2 = neural_dict['Fc2']['B']

    z1 = np.dot(X_mat,W1.T).T + B1
    a1 = ReLU(z1)
    z2 = (np.dot(W2, a1)+ B2)
    a2 = sigmoid(z2).T
    neural_dict['Fc1']['a'] = a1
    neural_dict['Fc2']['a'] = a2

    return a2
```

In [15]:
```
def backward_pass(f1):
    m = X_train.shape[0]
    y = Y_train_onehot
    global neural_dict

    W1 = neural_dict['Fc1']['W']
    W2 = neural_dict['Fc2']['W']
    B1 = neural_dict['Fc1']['B']
    B2 = neural_dict['Fc2']['B']
    a1 = neural_dict['Fc1']['a']
    a2 = neural_dict['Fc2']['a']

    dl_dz2 = a2-y
    dl_dw2 = np.dot(dl_dz2.T, a1.T)/m
    dl_db2 = np.sum(dl_dz2.T, axis=1, keepdims=True)/m
    dl_da1 = np.dot(dl_dz2, W2)
    dl_dz1 = np.multiply(dl_da1.T,ReLU_Derivative(a1))
    dl_dw1 = np.dot(dl_dz1, f1)/m
    dl_db1 = np.sum(dl_dz1, axis=1, keepdims=True)/m

    return dl_dw1, dl_dw2, dl_db1, dl_db2, dl_dz1
```

In [16]:
```
def calculate_loss():
    global neural_dict
    a = neural_dict['Fc2']['a']
    y = Y_train_onehot
    return (np.multiply(y,np.log(a)) + np.multiply((1-y),np.log(1-a)))
```

In [17]:
```
def calculate_cost():
    m = X_train.shape[0]
    cost = -np.sum(calculate_loss())
    return cost/m
```

In [18]:
```
def gradient_descent(X_train,learning_rate,iterations=1):

    global plot_object
    global neural_dict

    all_costs = []
    count = 0
    a1,c1 = conv_forward(X_train, neural_dict['Conv1']['W'], neural_dict
['Conv1']['B'], neural_dict['Conv1']['params'],"ReLU")

    P1,cp1 = pool_forward(a1, neural_dict['Pool1']['params'], mode = "ma
x")

    a2,c2 = conv_forward(P1, neural_dict['Conv2']['W'], neural_dict['Con
v2']['B'], neural_dict['Conv2']['params'],"ReLU")

    P2,cp2 = pool_forward(a2,  neural_dict['Pool2']['params'], mode = "a
verage")

    f1 = flatten_array(P2)

    forward_pass(f1)

    new_cost = calculate_cost()
    current_cost = float("inf")
    while new_cost <= current_cost and count < iterations:
        print "Iteration : ", count
        all_costs.append(new_cost)

        dl_dw1, dl_dw2, dl_db1, dl_db2, dl_dz1 = backward_pass(f1)

        df1 = np.dot(dl_dz1.T,neural_dict['Fc1']['W'])


        neural_dict['Fc1']['W'] = neural_dict['Fc1']['W'] - (learning_ra
te * dl_dw1)
        neural_dict['Fc2']['W'] = neural_dict['Fc2']['W'] - (learning_ra
te * dl_dw2)
        neural_dict['Fc1']['B'] = neural_dict['Fc1']['B'] - (learning_ra
te * dl_db1)
        neural_dict['Fc2']['B'] = neural_dict['Fc2']['B'] - (learning_ra
te * dl_db2)


        dA2 = df1.reshape(P2.shape)
        dA2 = pool_backward(dA2, cp2, mode = "average")
        dz2 = np.multiply(dA2,ReLU_Derivative(a2))
        dA1, dW2, dB2 = conv_backward(dz2, c2)
        dA1 = pool_backward(dA1, cp1, mode = "max")
        dz1 = np.multiply(dA1,ReLU_Derivative(a1))
        dA0, dW1, dB1 = conv_backward(dz1, c1)


        neural_dict['Conv1']['W'] = neural_dict['Conv1']['W'] - (learnin
g_rate * dW1)
        neural_dict['Conv2']['W'] = neural_dict['Conv2']['W'] - (learnin
```

```
g_rate * dW2)
        neural_dict['Conv1']['B'] = neural_dict['Conv1']['B'] - (learnin
g_rate * dB1)
        neural_dict['Conv2']['B'] = neural_dict['Conv2']['B'] - (learnin
g_rate * dB2)

        current_cost = new_cost

        a1,c1 = conv_forward(X_train, neural_dict['Conv1']['W'], neural_
dict['Conv1']['B'], neural_dict['Conv1']['params'],"ReLU")

        P1,cp1 = pool_forward(a1, neural_dict['Pool1']['params'], mode =
 "max")

        a2,c2 = conv_forward(P1, neural_dict['Conv2']['W'], neural_dict[
'Conv2']['B'], neural_dict['Conv2']['params'],"ReLU")

        P2,cp2 = pool_forward(a2,  neural_dict['Pool2']['params'], mode
= "average")

        f1 = flatten_array(P2)

        forward_pass(f1)

        new_cost = calculate_cost()
        print new_cost
        count +=1
    plot_object[learning_rate] = all_costs
    print "Final cost : ",
    print new_cost
    print "Iternations : %d" % count
```

2. Load data

Using Jupyter notebook, load the data.

In [19]:
```
X_train = np.load("./ex5_train_x.npy")
Y_train = np.load("./ex5_train_y.npy")
print 'train x shape :', X_train.shape
print 'train y shape :', Y_train.shape
```

```
train x shape : (1020, 64, 64, 3)
train y shape : (1020,)
```

3. (10pts) Initialize parameters (Weights, bias for each layer)

Please initialize weight coefficients and bias terms for each layer. Please make sure the size (dimension) of each Weights and bias. Please consider optimum initialization method depending on Activation function. You may use your trained weights and bias. In this case, please make sure to submit the trained weights and bias as one separate file (para_yourFirstName_LastnName)

In [20]:
```
Y_train_onehot = one_hot_encoding(Y_train)
```

In [21]:
```
X_train = X_train/255.0
```

In [22]:
```
hidden_neurons = 108
output_neurons = 6
np.random.seed(1)

W1_fc = []
for i in range(0,hidden_neurons):
    sampl = np.random.uniform(low=0, high=1, size=(1296)) * 0.01
    W1_fc.append(sampl)
W2_fc = []
for i in range(0,output_neurons):
    sampl = np.random.uniform(low=0, high=1, size=(hidden_neurons)) * 0.
01
    W2_fc.append(sampl)
B1_fc = []
for i in range(0,hidden_neurons):
    B1_fc.append([0])
B2_fc = []
for i in range(0,output_neurons):
    B2_fc.append([0])

W1_fc = np.array(W1_fc)
W2_fc = np.array(W2_fc)
B1_fc = np.array(B1_fc)
B2_fc = np.array(B2_fc)
```

In [23]:
```
np.random.seed(1)
neural_dict = {

    'Conv1':{
        'W' : np.random.rand(4,4,3,8) * 0.01,
        'B' : np.random.rand(1,1,1,8) * 0.01,
        'params' : {
            'padding' : 1,
            'stride' : 2
        }
    },
    'Pool1' : {
        'params' : {
            'f' : 5,
            'stride' : 1
        }
    },
    'Conv2' : {

        'W' : np.random.rand(4,4,8,16) * 0.01,
        'B' : np.random.rand(1,1,1,16) * 0.01,
        'params' : {
            'padding' : 0,
            'stride' : 2
        }
    },
    'Pool2' : {
        'params' : {
            'f' : 5,
            'stride' : 1
        }
    },
    'Fc1' : {
        'W' : W1_fc,
        'B' : B1_fc
    },
    'Fc2' : {
        'W' : W2_fc,
        'B' : B2_fc
    }
}
```

In [24]:
```
plot_object = {}
start = datetime.now()
gradient_descent(X_train,1,iterations=7)
end = datetime.now()
plot_object
print "Time taken : ", (end-start)
```

```
Iteration :  0
3.04980915232
Iteration :  1
2.93061792507
Iteration :  2
2.85571357037
Iteration :  3
2.80729839931
Iteration :  4
2.7752704456
Iteration :  5
2.75367706657
Iteration :  6
2.73889074499
Final cost :  2.73889074499
Iternations : 7
Time taken :  1:02:40.312724
```

In [25]:
```
print "Final Cost :", calculate_cost()
```

```
Final Cost : 2.73889074499
```

4. (20pts) Optimization of Convolution Neural Network model

Please build your model with forward propagation procedure and backward propagation procedure. Please print out the size (dimension) of each layer (C1, P1, C2, P2, F3, F4, F5) Please print your CNN architecture model as the above table. Please optimize your model using a learning rate and number of iteration. Please print out cost with number of iteration. It may take long time to calculate. You may limit the number of iteration less than 10.

In [26]:
```
print X_train.shape
print neural_dict['Conv1']['W']
print neural_dict['Conv1']['B']
print neural_dict['Conv2']['W']
print neural_dict['Conv2']['B']
```

```
(1020, 64, 64, 3)
[[[[  4.17022005e-03   7.20324493e-03   1.14374817e-06   3.02332573e-03
      1.46755891e-03   9.23385948e-04   1.86260211e-03   3.45560727e-0
3]
   [  3.96767474e-03   5.38816734e-03   4.19194514e-03   6.85219500e-03
      2.04452250e-03   8.78117436e-03   2.73875932e-04   6.70467510e-0
3]
   [  4.17304802e-03   5.58689828e-03   1.40386939e-03   1.98101489e-03
      8.00744569e-03   9.68261576e-03   3.13424178e-03   6.92322616e-0
3]]

  [[  8.76389152e-03   8.94606664e-03   8.50442114e-04   3.90547832e-04
      1.69830420e-03   8.78142503e-03   9.83468338e-04   4.21107625e-0
3]
   [  9.57889530e-03   5.33165285e-03   6.91877114e-03   3.15515631e-03
      6.86500928e-03   8.34625672e-03   1.82882773e-04   7.50144315e-0
3]
   [  9.88861089e-03   7.48165654e-03   2.80443992e-03   7.89279328e-03
      1.03226007e-03   4.47893526e-03   9.08595503e-03   2.93614148e-0
3]]

  [[  2.87775339e-03   1.30028572e-03   1.93669579e-04   6.78835533e-03
      2.11628116e-03   2.65546659e-03   4.91573159e-03   5.33625451e-0
4]
   [  5.74117605e-03   1.46728575e-03   5.89305537e-03   6.99758360e-03
      1.02334429e-03   4.14055988e-03   6.94400158e-03   4.14179270e-0
3]
   [  4.99534589e-04   5.35896406e-03   6.63794645e-03   5.14889112e-03
      9.44594756e-03   5.86555041e-03   9.03401915e-03   1.37474704e-0
3]]

  [[  1.39276347e-03   8.07391289e-03   3.97676837e-03   1.65354197e-03
      9.27508580e-03   3.47765860e-03   7.50812103e-03   7.25997985e-0
3]
   [  8.83306091e-03   6.23672207e-03   7.50942434e-03   3.48898342e-03
      2.69927892e-03   8.95886218e-03   4.28091190e-03   9.64840047e-0
3]
   [  6.63441498e-03   6.21695720e-03   1.14745973e-03   9.49489259e-03
      4.49912133e-03   5.78389614e-03   4.08136803e-03   2.37026980e-0
3]]]


 [[[  9.03379521e-03   5.73679487e-03   2.87032703e-05   6.17144914e-03
      3.26644902e-03   5.27058102e-03   8.85942099e-03   3.57269760e-0
3]
   [  9.08535151e-03   6.23360116e-03   1.58212428e-04   9.29437234e-03
      6.90896918e-03   9.97322850e-03   1.72340508e-03   1.37135750e-0
3]
   [  9.32595463e-03   6.96818161e-03   6.60001727e-04   7.55463053e-03
      7.53876188e-03   9.23024536e-03   7.11524759e-03   1.24270962e-0
3]]

  [[  1.98801338e-04   2.62109869e-04   2.83064880e-04   2.46211068e-03
      8.60027949e-03   5.38831064e-03   5.52821979e-03   8.42030892e-0
3]
   [  1.24173315e-03   2.79183679e-03   5.85759271e-03   9.69595748e-03
      5.61030219e-03   1.86472894e-04   8.00632673e-03   2.32974274e-0
```

```
3]
   [  8.07105196e-03   3.87860644e-03   8.63541855e-03   7.47121643e-03
      5.56240234e-03   1.36455226e-03   5.99176895e-04   1.21343456e-0
3]]

  [[  4.45518785e-04   1.07494129e-03   2.25709339e-03   7.12988980e-03
      5.59716982e-03   1.25559802e-04   7.19742797e-04   9.67276330e-0
3]
   [  5.68100462e-03   2.03293235e-03   2.52325745e-03   7.43825854e-03
      1.95429481e-03   5.81358927e-03   9.70019989e-03   8.46828801e-0
3]
   [  2.39847759e-03   4.93769714e-03   6.19955718e-03   8.28980900e-03
      1.56791395e-03   1.85762022e-04   7.00221437e-04   4.86345111e-0
3]]

  [[  6.06329462e-03   5.68851437e-03   3.17362409e-03   9.88616154e-03
      5.79745219e-03   3.80141173e-03   5.50948219e-03   7.45334431e-0
3]
   [  6.69232893e-03   2.64919558e-03   6.63348344e-04   3.70084198e-03
      6.29717507e-03   2.10174010e-03   7.52755554e-03   6.65364814e-0
4]
   [  2.60315099e-03   8.04754564e-03   1.93434283e-03   6.39460881e-03
      5.24670309e-03   9.24807970e-03   2.63296770e-03   6.59610907e-0
4]]]


 [[[  7.35065963e-03   7.72178030e-03   9.07815853e-03   9.31972069e-03
      1.39515730e-04   2.34362086e-03   6.16778357e-03   9.49016321e-0
3]
   [  9.50176119e-03   5.56653188e-03   9.15606350e-03   6.41566209e-03
      3.90007714e-03   4.85990667e-03   6.04310483e-03   5.49547922e-0
3]
   [  9.26181427e-03   9.18733436e-03   3.94875613e-03   9.63262528e-03
      1.73955667e-03   1.26329519e-03   1.35079158e-03   5.05662166e-0
3]]

  [[  2.15248053e-04   9.47970211e-03   8.27115471e-03   1.50189807e-04
      1.76196256e-03   3.32063574e-03   1.30996845e-03   8.09490692e-0
3]
   [  3.44736653e-03   9.40107482e-03   5.82014180e-03   8.78831984e-03
      8.44734445e-03   9.05392319e-03   4.59880266e-03   5.46346816e-0
3]
   [  7.98603591e-03   2.85718852e-03   4.90253523e-03   5.99110308e-03
      1.55332756e-04   5.93481408e-03   4.33676349e-03   8.07360529e-0
3]]

  [[  3.15244803e-03   8.92888709e-03   5.77857215e-03   1.84010202e-03
      7.87929234e-03   6.12031177e-03   5.39092721e-04   4.20193680e-0
3]
   [  6.79068837e-03   9.18601778e-03   4.02024891e-06   9.76759149e-03
      3.76580315e-03   9.73783538e-03   6.04716101e-03   8.28845808e-0
3]
   [  5.74711505e-03   6.28076198e-03   2.85576282e-03   5.86833341e-03
      7.50021764e-03   8.58313836e-03   7.55082188e-03   6.98057248e-0
3]]

  [[  8.64479430e-03   3.22680997e-03   6.70788791e-03   4.50873936e-03
```

```
      3.82102752e-03   4.10811350e-03   4.01479583e-03   3.17383946e-0
3]
   [  6.21919368e-03   4.30247271e-03   9.73802078e-03   6.77800891e-03
      1.98569888e-03   4.26701009e-03   3.43346240e-03   7.97638804e-0
3]
   [  8.79998289e-03   9.03841956e-03   6.62719812e-03   2.70208262e-03
      2.52366702e-03   8.54897943e-03   5.27714646e-03   8.02161084e-0
3]]]


 [[[  5.72488517e-03   7.33142525e-03   5.19011627e-03   7.70883911e-03
      5.68857991e-03   4.65709879e-03   3.42688908e-03   6.82093484e-0
4]
   [  3.77924179e-03   7.96260777e-04   9.82817114e-03   1.81612851e-03
      8.11858698e-03   8.74961645e-03   6.88413252e-03   5.69494413e-0
3]
   [  1.60971437e-03   4.66880023e-03   3.45172051e-03   2.25039958e-03
      5.92511869e-03   3.12269838e-03   9.16305553e-03   9.09635525e-0
3]]

  [[  2.57118294e-03   1.10891301e-03   1.92962732e-03   4.99584171e-03
      7.28585668e-03   2.08194438e-03   2.48033558e-03   8.51671875e-0
3]
   [  4.15848718e-03   6.16685067e-03   2.33666139e-03   1.01967259e-03
      5.15857017e-03   4.77140987e-03   1.52671644e-03   6.21806232e-0
3]
   [  5.44010119e-03   6.54137347e-03   1.44545540e-03   7.51527817e-03
      2.22049140e-03   5.19351824e-03   7.85296028e-03   2.23304280e-0
4]]

  [[  3.24362460e-03   8.72922376e-03   8.44709608e-03   5.38440593e-03
      8.66608274e-03   9.49805991e-03   8.26406998e-03   8.54115444e-0
3]
   [  9.87434018e-04   6.51304332e-03   7.03516988e-03   6.10240813e-03
      7.99615262e-03   3.45712199e-04   7.70238735e-03   7.31728601e-0
3]
   [  2.59698393e-03   2.57069299e-03   6.32303317e-03   3.45297462e-03
      7.96588678e-03   4.46146232e-03   7.82749415e-03   9.90471784e-0
3]]

  [[  3.00248340e-03   1.43005828e-03   9.01308436e-03   5.41559379e-03
      9.74740371e-03   6.36604400e-03   9.93913025e-03   5.46070804e-0
3]
   [  5.26425934e-03   1.35427903e-03   3.55705171e-03   2.62185673e-04
      1.60395180e-03   7.45637193e-03   3.03996899e-04   3.66543097e-0
3]
   [  8.62346253e-03   6.92677718e-03   6.90942142e-03   1.88636801e-03
      4.41904281e-03   5.81577407e-03   9.89751708e-03   2.03906225e-0
3]]]]
[[[[ 0.00247733  0.00262173  0.00750172  0.00456975  0.00056929  0.0050
8516
     0.0021196   0.00798604]]]]
[[[[-174.77122893 -206.79435571 -296.349182   ..., -315.64853902
    -300.34403038 -412.22091151]
   [-181.56354367 -214.82458443 -307.86810817 ..., -327.92194927
    -312.01538026 -428.23930632]
   [-282.95146506 -334.79966938 -479.81280825 ..., -511.10676204
```

```
  -486.2741557  -667.43579753]
 ...,
 [-241.71834241 -286.00575907 -409.8761726  ..., -436.59701636
  -415.40607112 -570.15504561]
 [-159.2912497  -188.4782318  -270.10473527 ..., -287.69860208
  -273.74428275 -375.71369372]
 [-314.48322533 -372.10737025 -533.27504257 ..., -568.05958898
  -540.46663117 -741.80395539]]

[[-174.89666044 -206.93996909 -296.5607396  ..., -315.87342857
  -300.55624115 -412.51074519]
 [-181.68958285 -214.97686569 -308.08670854 ..., -328.14606162
  -312.2327253  -428.53142932]
 [-283.04864647 -334.91177973 -479.97637186 ..., -511.27007739
  -486.44408321 -667.65156506]
 ...,
 [-241.8450721  -286.15230793 -410.09751414 ..., -436.81867883
  -415.62327573 -570.43513383]
 [-159.41159348 -188.61354257 -270.29664996 ..., -287.89226667
  -273.9380254  -375.9680145 ]
 [-314.6002542  -372.24644508 -533.4736925  ..., -568.26512931
  -540.67123426 -742.08315896]]

[[-174.99089369 -207.04702536 -296.71386693 ..., -316.02259835
  -300.71176377 -412.70904138]
 [-181.78536964 -215.0779091  -308.22687388 ..., -328.29749312
  -312.37589201 -428.72605171]
 [-283.11476403 -334.98559693 -480.0756759  ..., -511.37998724
  -486.55373839 -667.79874479]
 ...,
 [-241.93443512 -286.25122678 -410.23526416 ..., -436.96625089
  -415.76520396 -570.62605267]
 [-159.49047499 -188.69486648 -270.41617919 ..., -288.01591944
  -274.06231651 -376.13578362]
 [-314.6881756  -372.33209271 -533.60756094 ..., -568.39238213
  -540.79738192 -742.25479717]]

[[-175.0392095  -207.09776467 -296.78275471 ..., -316.08697028
  -300.77867661 -412.79870199]
 [-181.82969914 -215.12719854 -308.29919472 ..., -328.3514688
  -312.45053425 -428.81404364]
 [-283.14278296 -335.01664779 -480.12111533 ..., -511.41324754
  -486.58649584 -667.84469112]
 ...,
 [-241.97311007 -286.29917575 -410.30125143 ..., -437.01427016
  -415.83042927 -570.71118231]
 [-159.52009837 -188.73697315 -270.46657399 ..., -288.06041279
  -274.10906838 -376.19884615]
 [-314.7197462  -372.37595145 -533.66028693 ..., -568.43403838
  -540.85556722 -742.31094179]]]


[[[-174.21517545 -206.13366601 -295.40167523 ..., -314.63495166
   -299.38574286 -410.900572  ]
  [-180.99077954 -214.14597213 -306.89265772 ..., -326.87067188
   -311.02287865 -426.86849746]
  [-282.48138928 -334.23179945 -479.0007992  ..., -510.2346016
```

```
  -485.46041785 -666.30303774]
 ...,
 [-241.15603846 -285.33688799 -408.92087701 ..., -435.56418669
  -414.43400187 -568.80738033]
 [-158.76475817 -187.84425586 -269.20049015 ..., -286.72497723
  -272.83353666 -374.45438432]
 [-313.9213961  -371.4363157  -532.31814593 ..., -567.01435844
  -539.49486419 -740.46474765]]

[[-174.29080548 -206.21209814 -295.51369901 ..., -314.74533651
  -299.49253878 -411.04762388]
 [-181.05573793 -214.22622601 -307.00544798 ..., -326.98507663
  -311.14126238 -427.01923932]
 [-282.52555854 -334.28737843 -479.0789434  ..., -510.30611963
  -485.53406253 -666.39885142]
 ...,
 [-241.22103141 -285.41418106 -409.03142907 ..., -435.66575719
  -414.53955118 -568.94992556]
 [-158.82115353 -187.91569126 -269.29612342 ..., -286.81386951
  -272.92124423 -374.56745543]
 [-313.98202799 -371.50269623 -532.4101564  ..., -567.11165552
  -539.5882858  -740.58786869]]

[[-174.3058431  -206.23148586 -295.54105683 ..., -314.76213819
  -299.51431564 -411.06154577]
 [-181.08081847 -214.23918673 -307.02478669 ..., -326.99124884
  -311.15432713 -427.04664186]
 [-282.52753386 -334.29113588 -479.07060622 ..., -510.29597641
  -485.53382652 -666.38577695]
 ...,
 [-241.24070013 -285.42852825 -409.04370941 ..., -435.67354541
  -414.55447706 -568.95366774]
 [-158.83153007 -187.91949941 -269.29477941 ..., -286.80308594
  -272.92308114 -374.56088509]
 [-313.9952387  -371.51141052 -532.42050223 ..., -567.11325698
  -539.59920798 -740.595028  ]]

[[-174.27092978 -206.17272484 -295.45481886 ..., -314.66476433
  -299.43562961 -410.93941216]
 [-181.04249619 -214.1919709  -306.94306502 ..., -326.89758126
  -311.07541846 -426.92151705]
 [-282.48651921 -334.22701007 -478.98556324 ..., -510.19649034
  -485.44456536 -666.26409304]
 ...,
 [-241.19116955 -285.36229744 -408.94855977 ..., -435.57046808
  -414.46476033 -568.81996772]
 [-158.78240907 -187.86127818 -269.20597387 ..., -286.69866997
  -272.82423507 -374.42118953]
 [-313.94874257 -371.45479834 -532.32947378 ..., -567.00407038
  -539.50903411 -740.45397558]]]


[[[-173.58786168 -205.38683895 -294.34918702 ..., -313.49558541
   -298.31378725 -409.41586205]
  [-180.34018254 -213.37619767 -305.78869163 ..., -325.6872778
   -309.91087456 -425.34569887]
  [-281.94589914 -333.60311931 -478.09298296 ..., -509.25458467
```

```
  -484.53429055 -665.04179712]
 ...,
 [-240.51608892 -284.58221157 -407.84064678 ..., -434.40629553
  -413.34024537 -567.29988773]
 [-158.1619283  -187.14259987 -268.18384378 ..., -285.63293112
  -271.80324194 -373.03756581]
 [-313.28607727 -370.68529251 -531.23091962 ..., -565.85652394
  -538.39912465 -738.95846814]]

[[-173.58211243 -205.36927852 -294.31151988 ..., -313.4534902
  -298.27683694 -409.36408837]
 [-180.33568762 -213.36056155 -305.76645067 ..., -325.65010387
  -309.88721359 -425.29240681]
 [-281.9250617  -333.57042238 -478.05371975 ..., -509.20039969
  -484.49573728 -664.96545372]
 ...,
 [-240.50849739 -284.55662233 -407.80833321 ..., -434.35098564
  -413.30042683 -567.23601597]
 [-158.14873357 -187.1163991  -268.14774233 ..., -285.57428075
  -271.75082845 -372.96723476]
 [-313.26764781 -370.65925565 -531.1953004  ..., -565.80586617
  -538.3603721  -738.88904132]]

[[-173.50842453 -205.2730686  -294.16556427 ..., -313.28090181
  -298.1324279  -409.15013077]
 [-180.25651358 -213.26170678 -305.62036221 ..., -325.47969176
  -309.73476617 -425.07521873]
 [-281.85464196 -333.47339161 -477.90614847 ..., -509.03732355
  -484.34846369 -664.75882434]
 ...,
 [-240.42628033 -284.4545446  -407.65008564 ..., -434.17643003
  -413.14285329 -567.01170951]
 [-158.07152452 -187.00505533 -267.98729151 ..., -285.39598427
  -271.5964414  -372.73284718]
 [-313.18602027 -370.55066944 -531.04314545 ..., -565.63021911
  -538.19931332 -738.66899231]]

[[-173.35644522 -205.09637068 -293.90361665 ..., -312.98871291
  -297.8557835  -408.76215912]
 [-180.11548893 -213.08516678 -305.3573614  ..., -325.18431251
  -309.46426814 -424.69866467]
 [-281.71479279 -333.31489977 -477.66412242 ..., -508.77139301
  -484.10255169 -664.40433374]
 ...,
 [-240.26853977 -284.26930025 -407.36998014 ..., -433.86768742
  -412.85954852 -566.60320749]
 [-157.92419521 -186.81934266 -267.72104167 ..., -285.10473739
  -271.32538442 -372.34924293]
 [-313.04284761 -370.36948128 -530.77804846 ..., -565.341056
  -537.92393962 -738.28402567]]]


[[[-172.91151704 -204.58371659 -293.18892537 ..., -312.26557976
   -297.14301623 -407.81300349]
  [-179.64574515 -212.55252413 -304.60847487 ..., -324.41702258
   -308.70937476 -423.68696679]
  [-281.36396224 -332.91007709 -477.10666379 ..., -508.20338557
```

```
   -483.54725827 -663.66860493]
  ...,
  [-239.83178868 -283.76627429 -406.66463134 ..., -433.14535198
   -412.14715136 -565.66112497]
  [-157.52058504 -186.36976295 -267.08321803 ..., -284.45458324
   -270.6884265  -371.49212633]
  [-312.59886435 -369.86619417 -530.0628522  ..., -564.61017742
   -537.21059173 -737.32878504]]

 [[-172.80533541 -204.44580176 -292.9913087  ..., -312.03315225
   -296.93600747 -407.51494659]
  [-179.5383485  -212.41487751 -304.40853634 ..., -324.18745356
   -308.50729198 -423.39059394]
  [-281.26461526 -332.78525796 -476.92392073 ..., -507.99089844
   -483.34942485 -663.38939735]
  ...,
  [-239.72121203 -283.6211646  -406.45743564 ..., -432.90967877
   -411.93181886 -565.3552676 ]
  [-157.40706911 -186.23033105 -266.87590422 ..., -284.22053642
   -270.47214182 -371.19259738]
  [-312.48556024 -369.73095651 -529.86099226 ..., -564.37389594
   -537.00666897 -737.0252044 ]]

 [[-172.61024826 -204.20694193 -292.63175403 ..., -311.6362418
   -296.57582723 -407.00356193]
  [-179.34217142 -212.17529745 -304.05178703 ..., -323.7977058
   -308.1426477  -422.88932647]
  [-281.09005442 -332.57427628 -476.6070446  ..., -507.64152138
   -483.02661762 -662.93941272]
  ...,
  [-239.51345608 -283.37208102 -406.08845057 ..., -432.49758586
   -411.55581702 -564.82478299]
  [-157.21144301 -185.98538255 -266.52381851 ..., -283.83384205
   -270.11021984 -370.69388067]
  [-312.29313125 -369.4878011  -529.51148892 ..., -563.98843618
   -536.64941106 -736.52412619]]

 [[-172.33706839 -203.87252106 -292.14233331 ..., -311.09551345
   -296.0681472  -406.30042491]
  [-179.06930988 -211.84253894 -303.55526769 ..., -323.25835516
   -307.6403746  -422.1854501 ]
  [-280.84486972 -332.27260896 -476.17083865 ..., -507.17069784
   -482.58957116 -662.32137217]
  ...,
  [-239.22979622 -283.01793023 -405.58149118 ..., -431.94191319
   -411.03435389 -564.09398733]
  [-156.94546069 -185.66109273 -266.04128089 ..., -283.30415656
   -269.62332716 -369.99911412]
  [-312.02518732 -369.15309862 -529.03106662 ..., -563.45339986
   -536.15829097 -735.82795527]]]]
[[[[ -520.44581561  -615.81425729  -882.56805956 -2127.75978639
    -1324.82556665 -1765.48294358  -796.50051318  -792.23557526
    -1514.58597596 -1997.8138618  -1423.40806342 -2244.69446896
    -1762.37460372  -940.21688516  -894.48014454 -1227.73914213]]]]
```

In [31]:

```
file_obj = h5py.File("Weights_A5_Tanmay_Bhatt.hdf5","w")
```

In [32]:
```
dataset = file_obj.create_dataset("C1W", data=neural_dict['Conv1']['W'])
dataset = file_obj.create_dataset("C1B", data=neural_dict['Conv1']['B'])
dataset = file_obj.create_dataset("C2W", data=neural_dict['Conv2']['W'])
dataset = file_obj.create_dataset("C2B", data=neural_dict['Conv2']['B'])
dataset = file_obj.create_dataset("FC1W", data=neural_dict['Fc1']['W'])
dataset = file_obj.create_dataset("FC1B", data=neural_dict['Fc1']['B'])
dataset = file_obj.create_dataset("FC2W", data=neural_dict['Fc2']['W'])
dataset = file_obj.create_dataset("FC2B", data=neural_dict['Fc2']['B'])
file_obj.close()
```

In [33]:
```
def test_foward_pass(Y_train,C1W,C1B,C2W,C2B,FC1W,FC1B,FC2W,FC2B):


    def test_calculate_loss(y,a):

        return (np.multiply(y,np.log(a)) + np.multiply((1-y),np.log(1-a
)))

    def test_calculate_cost(y,a):
        m = X_train.shape[0]
        cost = -np.sum(test_calculate_loss(y,a))
        return cost/m


    def test_one_hot_encoding(mat):
        list_of_list = []
        for i in range(0,len(mat)):
            small_list = np.zeros(np.max(mat)+1)
            small_list[mat[i]] = 1
            list_of_list.append(small_list)
        result = np.array(list_of_list)
        return result

    neural_dict['Conv1']['W'] = C1W
    neural_dict['Conv1']['B'] = C1B
    neural_dict['Conv2']['W'] = C2W
    neural_dict['Conv2']['B'] = C2B
    neural_dict['Fc1']['W'] = FC1W
    neural_dict['Fc1']['B'] = FC1B
    neural_dict['Fc2']['W'] = FC2W
    neural_dict['Fc2']['B'] = FC2B


    neural_dict['Conv1']['params'] = {
            'padding' : 1,
            'stride' : 2
    }
    neural_dict['Pool1']['params'] = {
            'f' : 5,
            'stride' : 1
    }
    neural_dict['Conv2']['params'] = {
            'padding' : 0,
            'stride' : 2
    }
    neural_dict['Pool2']['params'] = {
            'f' : 5,
            'stride' : 1
    }

    ac1,c1 = conv_forward(X_train, neural_dict['Conv1']['W'], neural_dic
t['Conv1']['B'], neural_dict['Conv1']['params'],"ReLU")
    P1,cp1 = pool_forward(ac1, neural_dict['Pool1']['params'], mode = "m
ax")
    ac2,c2 = conv_forward(P1, neural_dict['Conv2']['W'], neural_dict['Co
nv2']['B'], neural_dict['Conv2']['params'],"ReLU")
```

```
    P2,cp2 = pool_forward(ac2,  neural_dict['Pool2']['params'], mode = 
"average")

    f1 = []
    for item in P2:
        f1.append(item.flatten())
    f1 = np.asarray(f1)

    W1 = neural_dict['Fc1']['W']
    W2 = neural_dict['Fc2']['W']
    B1 = neural_dict['Fc1']['B']
    B2 = neural_dict['Fc2']['B']

    z1 = np.dot(f1,W1.T).T + B1
    a1 = ReLU(z1)
    z2 = (np.dot(W2, a1)+ B2)
    a2 = sigmoid(z2).T

    Y_train_onehot = test_one_hot_encoding(Y_train)

    print "Training data shape : ", X_train.shape
    print "Convolution 1 shape : ", ac1.shape
    print "Pooling 1 shape : ",P1.shape
    print "Convolution 2 shape : ", ac2.shape
    print "Pooling 2 shape : ",P2.shape
    print "Flatten 1 shape : ",f1.shape
    print "Fully Connected 1 shape : ",a1.shape
    print "Fully Connected 2 shape : ",a2.shape

    print neural_dict['Conv1']['params']
    print neural_dict['Conv2']['params']
    print neural_dict['Pool1']['params']
    print neural_dict['Pool2']['params']
    print "Cost is :" , test_calculate_cost(Y_train_onehot,a2)
```

In [34]:
```
file_obj = h5py.File("Weights_A5_Tanmay_Bhatt.hdf5","r")
```

In [35]:
```
C1W = np.array(file_obj['C1W'])
C1B = np.array(file_obj['C1B'])
C2W = np.array(file_obj['C2W'])
C2B = np.array(file_obj['C2B'])
FC1W = np.array(file_obj['FC1W'])
FC1B = np.array(file_obj['FC1B'])
FC2W = np.array(file_obj['FC2W'])
FC2B = np.array(file_obj['FC2B'])
test_foward_pass(Y_train,C1W,C1B,C2W,C2B,FC1W,FC1B,FC2W,FC2B)
```

```
Training data shape :  (1020, 64, 64, 3)
Convolution 1 shape :  (1020, 32, 32, 8)
Pooling 1 shape :  (1020, 28, 28, 8)
Convolution 2 shape :  (1020, 13, 13, 16)
Pooling 2 shape :  (1020, 9, 9, 16)
Flatten 1 shape :  (1020, 1296)
Fully Connected 1 shape :  (108, 1020)
Fully Connected 2 shape :  (1020, 6)
{'padding': 1, 'stride': 2}
{'padding': 0, 'stride': 2}
{'stride': 1, 'f': 5}
{'stride': 1, 'f': 5}
Cost is : 2.73889074499
```