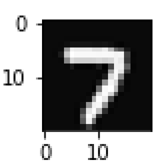
In [19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import defaultdict
import random
%matplotlib inline
```

In [20]:
```
Mnist_df = pd.read_csv("ex3_train.csv")
# Alternative way of doing -----Mnist_df_train_data = Mnist_df.as_matrix
(columns=Mnist_df.columns[0:400])
Mnist_df_train_data = Mnist_df.values
Mnist_df_train_label = Mnist_df.as_matrix(columns=Mnist_df.columns[400:4
01])
#Mnist_df_train_data_with_bias = np.c_[np.ones((Mnist_df_train_data.shap
e[0] ,1)),Mnist_df_train_data]## Add 1 for the bias
```

In [21]:
```
Mnist_df_test = pd.read_csv("ex3_test.csv")
Mnist_df_test_data = Mnist_df_test.values
Mnist_df_test_label = Mnist_df_test.as_matrix(columns=Mnist_df.columns[4
00:401])
#Mnist_df_test_data_with_bias = np.c_[np.ones((Mnist_df_test_data.shape
[0] ,1)),Mnist_df_test_data]## Add 1 for the bias
```

In [22]:
```
fig = plt.figure(figsize=(1,1))
plt.imshow(np.reshape(Mnist_df_train_data[:,:-1][2],[20,20]).T,cmap='gra
y')
print (Mnist_df_train_data[2][-1])
```

```
7.0
```



In [23]:
```
## Label set - containing unique labels
Mnist_df_train_label_uq = set(Mnist_df.values[:,400].astype(int))
```

In [24]:
```
## Being used for this assignment
def label_dict_gen(label_set): ## Useful for scaling if  huge number of
 labels (both number and string)
    label_dict = {}
    count = 0
    for label in label_set:
        label_dict[label]=count
        count +=1
    return label_dict
```

In [25]:
```
def sigmoid(data):
    return 1/(1+np.exp(-data))
```

In [26]:
```
def softmax_prediction(data):
    result= np.exp(data) / np.sum(np.exp(data), axis=1, keepdims=True)
    return result ## The max proability per coloumn(each coloumn represe
nts a record , and
                                        ## The complete result matrix t
o calculate the loss
```

In [27]:
```
def forward_prop(data,weights_list):
    hidden_input = np.dot(data[:,:-1],weights_list[0])+weights_list[1]
    hidden_output = sigmoid(hidden_input)
    #hidden_output = np.c_[np.ones((hidden_output.shape[0] ,1)),hidden_o
utput]## Add 1 for the bias
    output_before_softmax = np.dot(hidden_output,weights_list[2])+weight
s_list[3]
    softmax_output = softmax_prediction(output_before_softmax)
    return softmax_output
```

In [28]:
```
def Backward_prop(data,initial_weights,label_dict,learning_rate):
    hidden_input = np.dot(data[:,:-1],initial_weights[0])+initial_weight
s[1]
    hidden_output = sigmoid(hidden_input)
    #hidden_output = np.c_[np.ones((hidden_output.shape[0] ,1)),hidden_o
utput]## Add 1 for the bias
    output_before_softmax = np.dot(hidden_output,initial_weights[2])+ini
tial_weights[3]
    softmax_output = softmax_prediction(output_before_softmax)

    label_general = [label_dict[label] for label in data[:,-1]]
    label_general = np.asarray(label_general)

    #print ("label",label_general)
    ## Andrej Karpathy Course calcualting the product of softmax with au
toencodin to give the cross-entropy array
    corect_logprobs = -np.log(softmax_output[range(softmax_output.shape[
0]),label_general])
    ## Sum the individual terms of output of autoencoding snd softmax in
 the above step and average it
    loss = np.sum(corect_logprobs)/softmax_output.shape[0]

  # compute the gradient on scores
    dsoftmax = softmax_output
    dsoftmax[range(softmax_output.shape[0]),label_general] -= 1 ## Very
 elegant only the element with true label is updated
    dsoftmax /= softmax_output.shape[0]

  # backpropate the gradient to the 2nd layer weight
    dW1 = np.dot(hidden_output.T, dsoftmax)
    db1 = np.sum(dsoftmax, axis=0, keepdims=True)

  # perform a parameter update
    gradient_descent(initial_weights[2],dW1,learning_rate)
    gradient_descent(initial_weights[3],db1,learning_rate)


    dlast = np.dot(dsoftmax,initial_weights[2].T)
    dsigmoid = np.dot(hidden_output,(1-hidden_output).T)
    dhidden = np.dot(dsigmoid,dlast)

  # Backpropagate the gradient to 1st layer weights
    dW0 = np.dot(data[:,:-1].T,dhidden)
    db0 = np.sum(dhidden, axis=0, keepdims=True)

  # perform a parameter update
    gradient_descent(initial_weights[0],dW0,learning_rate)
    gradient_descent(initial_weights[1],db0,learning_rate)
    return loss
```

In [29]:
```
def gradient_descent(weights,gradient,learning_rate):
    weights += -learning_rate * gradient
```

In [41]:
```
def SGD(training_data, epochs, mini_batch_size,initial_weights,Loss_grap
h,learning_rate = .01):
        n = training_data.shape[0]
        print ("For Learning Rate" ,learning_rate )
        for epoch in range(epochs):
            if mini_batch_size<n :
                np.random.shuffle(training_data)
            mini_batches = [training_data[k:k+mini_batch_size]for k in r
ange(0, n, mini_batch_size)]
            #print (len(mini_batches))
            for mini_batch in mini_batches:
                ## Loss gets updated with each batch and after the loop
 ends , the loss reflects the loss at end of 1 epoch
                Loss = Backward_prop(mini_batch,initial_weights,label_di
ct,learning_rate)
                #print (mini_batch[:,-1])
                #break
            #break
            if epoch % 1000== 0:
                Loss_graph[learning_rate].append([Loss,epoch])
                print ("iteration %d: loss %f" % (epoch, Loss))
                softmax_output = forward_prop(training_data,initial_weig
hts)
                prediction = np.argmax(softmax_output, axis=1)
                label_general = [label_dict_gen(Mnist_df_train_label_uq)
[label] for label in training_data[:,-1]]
                label_general = np.asarray(label_general)
                print(prediction,label_general)
                accuracy = np.mean(prediction == label_general)
                print ('training accuracy: %.4f' % accuracy)
                if accuracy>0.95  :
                    break
```

In [52]:
```
h = 25 # size of hidden layer
features = 400 # dimensionality
classes = 10 # number of classes
label_dict = label_dict_gen(Mnist_df_train_label_uq)
learning_rate_list = [.0001,.001,.01,]
```

In [ ]:
```
## Doing this for Full Batch
Loss_graph_AllData = defaultdict(list)
batch_size = 3500
epochs_full = 100000
Weights_basedOn_Learning_Rate_fullbatch = defaultdict(list)
## Doing this for Full Batch
for learning_rate in learning_rate_list:
    W0 = 0.01 * np.random.randn(features,h)
    b0 = np.zeros((1,h))
    W1 = 0.01 * np.random.randn(h,classes)
    b1 = np.zeros((1,classes))
    initial_weights = [W0,b0,W1,b1]
    SGD(Mnist_df_train_data, epochs, batch_size,initial_weights,Loss_gra
ph_AllData,learning_rate)
    Weights_basedOn_Learning_Rate_fullbatch[learning_rate] = initial_wei
ghts
```

In [55]:
```
## Doing this for mini Batch
Loss_graph_stochastic = defaultdict(list)
batch_size = 128
epochs_stochastic = 50000
Weights_basedOn_Learning_Rate = defaultdict(list)
for learning_rate in learning_rate_list:
    W0_mini = 0.01 * np.random.randn(features,h)
    b0_mini = np.zeros((1,h))
    W1_mini = 0.01 * np.random.randn(h,classes)
    b1_mini = np.zeros((1,classes))
    initial_weights_mini = [W0_mini,b0_mini,W1_mini,b1_mini]
    SGD(Mnist_df_train_data, epochs, batch_size,initial_weights_mini,Los
s_graph_stochastic,learning_rate)
    Weights_basedOn_Learning_Rate[learning_rate]=initial_weights_mini
```

```
For Learning Rate 0.0001
iteration 0: loss 2.300610
[8 0 8 ... 8 8 8] [2 4 5 ... 0 6 8]
training accuracy: 0.1217
iteration 1000: loss 2.288943
[2 2 2 ... 2 1 2] [9 4 7 ... 0 7 6]
training accuracy: 0.1657
iteration 2000: loss 2.271760
[2 0 7 ... 2 7 1] [4 0 7 ... 3 9 1]
training accuracy: 0.4703
iteration 3000: loss 2.238190
[7 1 1 ... 8 8 3] [7 5 1 ... 3 6 3]
training accuracy: 0.6231
iteration 4000: loss 2.212560
[1 2 2 ... 3 1 2] [8 9 7 ... 3 3 2]
training accuracy: 0.6926
iteration 5000: loss 2.182745
[3 1 8 ... 8 5 6] [5 5 8 ... 8 5 6]
training accuracy: 0.7274
iteration 6000: loss 2.148194
[4 0 2 ... 9 4 4] [4 0 2 ... 9 4 4]
training accuracy: 0.7520
iteration 7000: loss 2.108868
[0 8 9 ... 5 1 4] [9 8 9 ... 5 1 4]
training accuracy: 0.7731
iteration 8000: loss 2.078668
[0 3 2 ... 2 0 5] [0 3 2 ... 2 5 5]
training accuracy: 0.7814
iteration 9000: loss 2.075206
[4 7 1 ... 6 3 7] [4 7 1 ... 6 5 7]
training accuracy: 0.8006
iteration 10000: loss 2.014276
[7 1 9 ... 0 5 7] [7 1 3 ... 0 5 4]
training accuracy: 0.8123
iteration 11000: loss 1.992171
[3 3 1 ... 6 5 1] [3 8 7 ... 6 5 1]
training accuracy: 0.8257
iteration 12000: loss 1.962493
[6 1 7 ... 7 1 7] [6 1 7 ... 7 1 7]
training accuracy: 0.8369
iteration 13000: loss 1.938630
[6 3 7 ... 4 2 6] [8 3 7 ... 4 2 6]
training accuracy: 0.8463
iteration 14000: loss 1.813187
[6 7 6 ... 8 7 2] [6 7 6 ... 8 7 2]
training accuracy: 0.8517
iteration 15000: loss 1.851538
[2 3 5 ... 4 5 0] [2 3 5 ... 4 5 0]
training accuracy: 0.8597
iteration 16000: loss 1.797495
[0 5 7 ... 6 3 0] [0 5 9 ... 6 3 0]
training accuracy: 0.8640
iteration 17000: loss 1.811212
[2 3 6 ... 7 7 8] [2 3 2 ... 7 7 8]
training accuracy: 0.8663
iteration 18000: loss 1.809428
[7 7 0 ... 6 4 0] [9 7 0 ... 6 8 0]
```

```
training accuracy: 0.8706
iteration 19000: loss 1.769475
[6 1 6 ... 5 9 0] [6 1 6 ... 5 9 0]
training accuracy: 0.8763
iteration 20000: loss 1.659262
[7 6 4 ... 3 6 6] [7 6 4 ... 3 6 5]
training accuracy: 0.8794
iteration 21000: loss 1.650470
[1 1 2 ... 2 2 7] [1 1 2 ... 2 2 7]
training accuracy: 0.8760
iteration 22000: loss 1.687235
[1 2 6 ... 1 7 9] [1 2 6 ... 1 7 9]
training accuracy: 0.8857
iteration 23000: loss 1.592273
[5 6 7 ... 1 9 0] [5 6 7 ... 1 9 0]
training accuracy: 0.8846
iteration 24000: loss 1.556882
[6 1 3 ... 7 1 6] [6 1 3 ... 7 1 6]
training accuracy: 0.8846
iteration 25000: loss 1.594105
[9 1 8 ... 7 8 3] [9 1 1 ... 7 8 3]
training accuracy: 0.8831
iteration 26000: loss 1.521400
[0 8 3 ... 5 6 0] [0 8 3 ... 5 6 0]
training accuracy: 0.8826
iteration 27000: loss 1.443801
[7 6 2 ... 5 6 4] [7 6 2 ... 5 6 4]
training accuracy: 0.8857
iteration 28000: loss 1.505679
[2 7 6 ... 5 8 7] [2 7 6 ... 5 8 7]
training accuracy: 0.8909
iteration 29000: loss 1.490657
[6 4 7 ... 7 2 6] [3 4 9 ... 9 2 6]
training accuracy: 0.8903
iteration 30000: loss 1.489010
[1 8 9 ... 8 8 5] [1 8 9 ... 8 8 9]
training accuracy: 0.8906
iteration 31000: loss 1.454264
[1 8 3 ... 3 7 1] [8 4 3 ... 3 7 1]
training accuracy: 0.8900
iteration 32000: loss 1.376461
[7 9 0 ... 5 5 1] [7 9 0 ... 5 5 1]
training accuracy: 0.8903
iteration 33000: loss 1.446167
[7 0 9 ... 4 8 8] [7 7 9 ... 4 8 8]
training accuracy: 0.8966
iteration 34000: loss 1.340245
[4 1 4 ... 4 2 0] [4 1 4 ... 4 2 0]
training accuracy: 0.8946
iteration 35000: loss 1.179666
[4 8 6 ... 9 2 7] [7 3 6 ... 9 2 7]
training accuracy: 0.8946
iteration 36000: loss 1.232528
[3 1 0 ... 6 3 0] [3 1 0 ... 6 3 0]
training accuracy: 0.8974
iteration 37000: loss 1.245304
[5 7 3 ... 6 8 6] [5 7 3 ... 6 8 0]
```

```
training accuracy: 0.8943
iteration 38000: loss 1.248249
[8 2 1 ... 3 0 6] [8 2 8 ... 3 0 6]
training accuracy: 0.8963
iteration 39000: loss 1.239411
[1 1 0 ... 8 2 7] [1 3 0 ... 8 2 7]
training accuracy: 0.8931
iteration 40000: loss 1.288391
[4 0 2 ... 8 3 7] [4 0 2 ... 3 3 7]
training accuracy: 0.8934
iteration 41000: loss 1.221365
[1 0 1 ... 4 3 9] [1 0 1 ... 4 3 5]
training accuracy: 0.9014
iteration 42000: loss 1.226447
[1 0 2 ... 7 0 0] [1 0 2 ... 7 0 0]
training accuracy: 0.8951
iteration 43000: loss 1.130120
[8 5 3 ... 4 9 5] [8 5 5 ... 4 9 5]
training accuracy: 0.8920
iteration 44000: loss 1.191690
[3 1 3 ... 8 7 1] [3 1 3 ... 8 7 1]
training accuracy: 0.8974
iteration 45000: loss 1.301709
[7 4 6 ... 0 3 9] [7 4 6 ... 0 5 9]
training accuracy: 0.8997
iteration 46000: loss 1.035133
[9 5 4 ... 5 5 6] [9 5 4 ... 5 6 6]
training accuracy: 0.8966
iteration 47000: loss 1.126442
[4 5 5 ... 2 8 6] [4 5 5 ... 3 2 0]
training accuracy: 0.9026
iteration 48000: loss 1.136659
[5 7 7 ... 8 8 3] [5 2 9 ... 8 8 8]
training accuracy: 0.9011
iteration 49000: loss 1.078776
[7 4 5 ... 5 6 6] [7 4 5 ... 5 6 6]
training accuracy: 0.9009
For Learning Rate 0.001
iteration 0: loss 2.299999
[6 3 3 ... 6 3 3] [7 5 6 ... 7 7 4]
training accuracy: 0.1106
iteration 1000: loss 2.070524
[6 8 5 ... 5 8 3] [6 8 5 ... 5 2 3]
training accuracy: 0.8194
iteration 2000: loss 1.790182
[7 7 4 ... 3 2 8] [7 7 4 ... 3 2 8]
training accuracy: 0.8069
iteration 3000: loss 1.761631
[8 7 8 ... 3 7 6] [1 7 8 ... 3 7 6]
training accuracy: 0.8143
```

```
/anaconda2/envs/carnd-term1/lib/python3.5/site-packages/ipykernel_launc
her.py:2: RuntimeWarning: overflow encountered in exp
```

```
iteration 4000: loss 2.432586
[3 1 2 ... 1 2 3] [5 0 2 ... 0 4 1]
training accuracy: 0.0746
iteration 5000: loss 2.073489
[7 8 1 ... 1 0 7] [4 6 8 ... 9 4 4]
training accuracy: 0.2711
iteration 6000: loss 2.162143
[1 1 1 ... 1 7 6] [6 1 8 ... 3 7 6]
training accuracy: 0.2674
iteration 7000: loss 2.283029
[8 8 8 ... 8 8 8] [9 3 8 ... 6 3 8]
training accuracy: 0.1306
iteration 8000: loss 2.298436
[2 2 2 ... 2 2 2] [8 3 8 ... 0 2 5]
training accuracy: 0.1149
iteration 9000: loss 2.298798
[2 2 2 ... 2 2 2] [0 9 1 ... 5 9 9]
training accuracy: 0.1114
iteration 10000: loss 2.297395
[2 2 2 ... 2 2 2] [2 7 4 ... 3 8 8]
training accuracy: 0.1091
iteration 11000: loss 2.300249
[2 4 2 ... 2 2 2] [5 6 1 ... 3 0 1]
training accuracy: 0.1114
iteration 12000: loss 2.289105
[2 2 2 ... 2 2 2] [4 5 4 ... 3 4 3]
training accuracy: 0.1114
iteration 13000: loss 2.301732
[2 2 2 ... 2 2 2] [2 2 6 ... 9 1 9]
training accuracy: 0.1109
iteration 14000: loss 2.302177
[2 2 2 ... 2 2 2] [1 8 0 ... 6 8 2]
training accuracy: 0.1106
iteration 15000: loss 2.303720
[2 2 2 ... 2 2 2] [8 7 7 ... 2 2 6]
training accuracy: 0.1100
iteration 16000: loss 2.300738
[2 2 2 ... 2 2 2] [3 2 1 ... 2 7 1]
training accuracy: 0.1100
iteration 17000: loss 2.303087
[2 2 2 ... 2 2 2] [3 5 9 ... 4 5 2]
training accuracy: 0.1097
iteration 18000: loss 2.304931
[2 2 2 ... 2 2 2] [5 6 8 ... 2 8 9]
training accuracy: 0.1097
iteration 19000: loss 2.301713
[2 2 2 ... 2 2 2] [7 1 0 ... 5 6 5]
training accuracy: 0.1094
iteration 20000: loss 2.292002
[2 2 2 ... 2 2 2] [0 8 6 ... 3 8 0]
training accuracy: 0.1094
iteration 21000: loss 2.299076
[2 2 2 ... 2 2 2] [3 2 5 ... 9 1 7]
training accuracy: 0.1091
iteration 22000: loss 2.301300
[2 2 2 ... 2 2 2] [7 4 3 ... 6 4 0]
training accuracy: 0.1089
```

```
iteration 23000: loss 2.300188
[2 2 2 ... 2 2 2] [7 7 8 ... 8 7 1]
training accuracy: 0.1089
iteration 24000: loss 2.303070
[2 2 2 ... 2 2 2] [4 5 8 ... 8 8 6]
training accuracy: 0.1089
iteration 25000: loss 2.300116
[2 2 2 ... 7 2 2] [3 4 9 ... 7 8 7]
training accuracy: 0.1086
iteration 26000: loss 2.297523
[2 2 2 ... 2 2 2] [4 1 3 ... 7 9 2]
training accuracy: 0.1086
iteration 27000: loss 2.285814
[2 2 2 ... 4 2 2] [0 0 1 ... 4 0 3]
training accuracy: 0.1083
iteration 28000: loss 2.301269
[2 4 2 ... 2 2 2] [5 7 3 ... 8 7 1]
training accuracy: 0.1083
iteration 29000: loss 2.306978
[2 2 2 ... 2 2 2] [8 2 2 ... 4 4 1]
training accuracy: 0.1080
iteration 30000: loss 2.296904
[2 2 2 ... 2 2 2] [8 2 0 ... 0 5 7]
training accuracy: 0.1080
iteration 31000: loss 2.274892
[2 2 2 ... 2 2 2] [9 4 3 ... 7 0 8]
training accuracy: 0.1080
iteration 32000: loss 2.299925
[2 2 2 ... 2 2 2] [8 7 5 ... 1 4 0]
training accuracy: 0.1077
iteration 33000: loss 2.303482
[2 2 2 ... 2 2 2] [3 2 4 ... 8 7 6]
training accuracy: 0.1074
iteration 34000: loss 2.301212
[2 2 2 ... 2 2 2] [2 7 3 ... 4 1 5]
training accuracy: 0.1074
iteration 35000: loss 2.295649
[2 2 2 ... 2 2 2] [5 6 1 ... 6 2 5]
training accuracy: 0.1074
iteration 36000: loss 2.304037
[2 2 2 ... 2 2 2] [3 9 0 ... 1 7 5]
training accuracy: 0.1074
iteration 37000: loss 2.296303
[2 2 2 ... 2 2 2] [5 1 9 ... 5 1 0]
training accuracy: 0.1074
iteration 38000: loss 2.274619
[2 2 2 ... 2 2 2] [8 0 8 ... 3 6 5]
training accuracy: 0.1074
iteration 39000: loss 2.302146
[2 2 2 ... 2 2 2] [4 5 8 ... 1 0 0]
training accuracy: 0.1074
iteration 40000: loss 2.301978
[2 2 2 ... 2 2 2] [5 2 7 ... 1 3 9]
training accuracy: 0.1074
iteration 41000: loss 2.303168
[2 2 2 ... 2 2 2] [9 0 0 ... 9 7 3]
training accuracy: 0.1074
```

```
iteration 42000: loss 2.288534
[2 2 2 ... 2 2 2] [0 9 0 ... 7 8 8]
training accuracy: 0.1074
iteration 43000: loss 2.299991
[2 2 2 ... 2 2 2] [6 8 6 ... 3 5 2]
training accuracy: 0.1074
iteration 44000: loss 2.299325
[2 2 2 ... 2 2 2] [5 1 5 ... 5 8 8]
training accuracy: 0.1074
iteration 45000: loss 2.300710
[2 2 2 ... 2 2 2] [6 7 9 ... 1 4 1]
training accuracy: 0.1074
iteration 46000: loss 2.302301
[2 2 2 ... 2 2 2] [1 6 1 ... 3 1 9]
training accuracy: 0.1074
iteration 47000: loss 2.296906
[2 2 2 ... 2 2 2] [9 3 5 ... 2 6 3]
training accuracy: 0.1074
iteration 48000: loss 2.261811
[2 2 2 ... 2 2 2] [6 4 3 ... 4 8 0]
training accuracy: 0.1074
iteration 49000: loss 2.296937
[2 2 2 ... 2 2 2] [5 6 2 ... 5 6 8]
training accuracy: 0.1074
For Learning Rate 0.01
iteration 0: loss 2.301318
[1 1 1 ... 1 1 1] [4 1 4 ... 8 3 6]
training accuracy: 0.0780
iteration 1000: loss 2.258899
[2 2 2 ... 2 2 2] [6 8 9 ... 4 2 9]
training accuracy: 0.3129
iteration 2000: loss 2.216418
[2 2 2 ... 1 2 2] [3 9 9 ... 1 9 8]
training accuracy: 0.3246
iteration 3000: loss 2.201752
[2 2 2 ... 2 2 2] [4 0 3 ... 9 7 6]
training accuracy: 0.3137
iteration 4000: loss 2.120478
[2 2 2 ... 2 2 2] [8 4 4 ... 3 7 4]
training accuracy: 0.3260
iteration 5000: loss 2.112424
[2 2 1 ... 2 2 2] [9 6 1 ... 6 9 2]
training accuracy: 0.3534
iteration 6000: loss 2.120081
[7 2 2 ... 2 2 2] [7 2 3 ... 5 8 3]
training accuracy: 0.3214
iteration 7000: loss 2.082266
[2 2 2 ... 2 2 2] [4 9 7 ... 3 3 9]
training accuracy: 0.3377
iteration 8000: loss 2.154797
[2 2 2 ... 0 2 2] [3 8 3 ... 0 5 9]
training accuracy: 0.3191
iteration 9000: loss 2.124166
[2 2 2 ... 7 2 2] [6 0 0 ... 4 2 8]
training accuracy: 0.3329
iteration 10000: loss 2.116152
[2 2 2 ... 2 2 2] [2 6 3 ... 7 9 8]
```

```
training accuracy: 0.3049
iteration 11000: loss 2.202798
[2 2 2 ... 2 2 2] [0 3 8 ... 5 5 1]
training accuracy: 0.3017
iteration 12000: loss 1.966240
[0 1 2 ... 2 2 2] [0 1 8 ... 8 2 4]
training accuracy: 0.2740
iteration 13000: loss 2.019243
[2 2 2 ... 2 2 2] [3 2 2 ... 0 5 9]
training accuracy: 0.2774
iteration 14000: loss 2.029714
[2 2 6 ... 1 2 2] [5 7 6 ... 1 4 0]
training accuracy: 0.2774
iteration 15000: loss 2.106681
[2 2 2 ... 2 2 2] [0 5 9 ... 8 7 1]
training accuracy: 0.2780
iteration 16000: loss 2.107191
[1 2 2 ... 2 9 1] [1 1 2 ... 9 9 1]
training accuracy: 0.2749
iteration 17000: loss 2.248408
[2 2 2 ... 2 2 2] [6 8 6 ... 4 5 6]
training accuracy: 0.2749
iteration 18000: loss 2.084739
[2 2 2 ... 2 2 2] [4 5 2 ... 7 5 7]
training accuracy: 0.2706
iteration 19000: loss 2.173236
[2 2 2 ... 0 2 2] [5 0 2 ... 0 2 8]
training accuracy: 0.2649
iteration 20000: loss 2.000395
[2 7 2 ... 2 2 2] [3 7 3 ... 3 2 0]
training accuracy: 0.2751
iteration 21000: loss 2.073317
[2 2 2 ... 1 2 2] [5 0 6 ... 1 9 2]
training accuracy: 0.2583
iteration 22000: loss 2.169138
[2 2 9 ... 7 2 2] [9 2 9 ... 9 4 9]
training accuracy: 0.2586
iteration 23000: loss 2.116471
[2 2 0 ... 2 2 2] [7 2 0 ... 0 0 0]
training accuracy: 0.2577
iteration 24000: loss 1.865224
[2 5 1 ... 2 2 2] [2 5 1 ... 2 2 8]
training accuracy: 0.2531
iteration 25000: loss 2.213770
[2 2 2 ... 2 2 2] [1 7 4 ... 4 4 9]
training accuracy: 0.2509
iteration 26000: loss 2.080843
[2 2 2 ... 2 2 2] [6 8 9 ... 3 2 6]
training accuracy: 0.2477
iteration 27000: loss 2.047627
[2 2 1 ... 2 1 2] [3 7 1 ... 8 1 3]
training accuracy: 0.2563
iteration 28000: loss 2.165384
[2 2 2 ... 2 2 2] [4 1 2 ... 0 8 0]
training accuracy: 0.2574
iteration 29000: loss 2.162018
[2 2 2 ... 2 2 2] [1 4 5 ... 3 0 4]
```

```
training accuracy: 0.2440
iteration 30000: loss 2.186545
[2 2 2 ... 2 2 2] [0 9 9 ... 3 9 4]
training accuracy: 0.2400
iteration 31000: loss 2.180792
[2 2 2 ... 2 2 2] [8 8 8 ... 3 1 8]
training accuracy: 0.2331
iteration 32000: loss 2.176088
[2 2 2 ... 2 2 2] [2 1 6 ... 3 8 9]
training accuracy: 0.2423
iteration 33000: loss 2.184098
[2 2 2 ... 2 2 2] [1 9 3 ... 0 5 4]
training accuracy: 0.2391
iteration 34000: loss 2.036627
[2 2 2 ... 2 2 2] [5 9 6 ... 9 0 3]
training accuracy: 0.2411
iteration 35000: loss 2.109721
[2 2 2 ... 2 9 9] [3 0 4 ... 5 9 7]
training accuracy: 0.2351
iteration 36000: loss 2.157980
[5 2 2 ... 2 2 2] [5 4 6 ... 8 8 8]
training accuracy: 0.2169
iteration 37000: loss 2.166953
[2 2 2 ... 2 2 2] [2 2 3 ... 9 8 4]
training accuracy: 0.2151
iteration 38000: loss 2.140944
[1 2 2 ... 2 2 1] [1 8 0 ... 6 0 1]
training accuracy: 0.2069
iteration 39000: loss 2.035397
[2 2 2 ... 2 2 2] [7 9 9 ... 0 7 8]
training accuracy: 0.2086
iteration 40000: loss 2.131570
[2 2 2 ... 2 2 1] [1 6 8 ... 3 8 1]
training accuracy: 0.2077
iteration 41000: loss 2.117688
[2 2 2 ... 2 2 2] [6 4 5 ... 1 2 5]
training accuracy: 0.1957
iteration 42000: loss 2.122781
[2 2 2 ... 2 2 2] [2 2 9 ... 6 9 1]
training accuracy: 0.2069
iteration 43000: loss 2.162555
[2 2 2 ... 2 2 2] [0 2 4 ... 2 9 8]
training accuracy: 0.2077
iteration 44000: loss 2.013427
[2 2 2 ... 2 2 2] [4 0 9 ... 9 3 9]
training accuracy: 0.2083
iteration 45000: loss 1.922124
[2 2 2 ... 2 2 2] [0 2 4 ... 9 7 7]
training accuracy: 0.2126
iteration 46000: loss 2.149168
[2 2 7 ... 2 1 2] [3 8 7 ... 8 1 8]
training accuracy: 0.2114
iteration 47000: loss 2.082649
[2 2 2 ... 1 2 2] [4 8 8 ... 1 3 8]
training accuracy: 0.2043
iteration 48000: loss 2.192511
[2 2 2 ... 0 1 1] [8 0 1 ... 0 1 1]
```

```
training accuracy: 0.2134
iteration 49000: loss 2.114785
[2 2 2 ... 2 2 2] [9 5 6 ... 8 3 4]
training accuracy: 0.2134
```

In [77]:

```
softmax_output_train = forward_prop(Mnist_df_train_data,Weights_basedOn_Learning_Rate[.0001])
label_general_train = [label_dict[label] for label in Mnist_df_train_data[:,-1]]
label_general_train = np.asarray(label_general_train)
prediction_train = np.argmax(softmax_output_train, axis=1)
print (prediction_train,label_general_train)
print ('Testing_accuracy: %.2f' % (np.mean(prediction_train == label_general_train)))
```

```
[5 3 7 ... 0 2 0] [5 3 7 ... 0 2 0]
Testing_accuracy: 0.91
```

In [65]:

```
softmax_output_test = forward_prop(Mnist_df_test_data,Weights_basedOn_Learning_Rate[.0001])
label_general_test = [label_dict[label] for label in Mnist_df_test_data[:,-1]]
label_general_test = np.asarray(label_general_test)
prediction_test = np.argmax(softmax_output_test, axis=1)
print (prediction_test,label_general_test)
print ('Testing_accuracy: %.2f' % (np.mean(prediction_test == label_general_test)))
```

```
[9 5 5 ... 1 7 0] [9 6 5 ... 1 7 0]
Testing_accuracy: 0.86
```

In [76]:
```
fig = plt.figure(figsize=(10,8))
plt.ylabel('Loss')
plt.xlabel('Iteration')
for key, value in Loss_graph_stochastic.items():
    arrayvalue = np.asarray(value)
    plt.plot(arrayvalue[:,1],arrayvalue[:,0],'-+',label="Learning Rate %
 0.4f" %key)
    legend = plt.legend(loc='upper right', shadow=True)
```