NN

February 2, 2017

In [1]:
```python
import numpy as np
import random
from mnist import MNIST
import matplotlib.pyplot as plt
```

In [2]:
```python
mndata = MNIST('../HW1/Dataset')
trainX = np.array(mndata.load_training()[0])[:50000]
trainY = np.array(mndata.load_training()[1])[:50000]
```

In [3]:
```python
valIndices = np.random.choice(len(trainX), 2000)
nonValIndices = [x for x in range(len(trainX)) if x not in valIndices]

valX = trainX[valIndices]
valY = trainY[valIndices]

trainX = trainX[nonValIndices]
trainY = trainY[nonValIndices]

testX = np.array(mndata.load_testing()[0])[:1000]
testY = np.array(mndata.load_testing()[1])[:1000]

def feat(data,i):
    return data[i].tolist()

def oneHot(clas, noOfClasses):
    feat = np.zeros(noOfClasses)
    feat[clas] = 1;
    return feat

trnX = np.array([feat(trainX,i) for i in range(trainX.shape[0])])/256.0
trnY = np.array([oneHot(trainY[i], 10) for i in range(trainX.shape[0])])

tstX = np.array([feat(testX,i) for i in range(testX.shape[0])])/256.0
tstY = np.array([oneHot(testY[i], 10) for i in range(testX.shape[0])])

valX = np.array([feat(valX,i) for i in range(valX.shape[0])])/256.0
valY = np.array([oneHot(valY[i], 10) for i in range(valX.shape[0])])
```

In [58]:
```
def sigmoid(x):
    return 1.0/(1+np.exp(-1*x))

def lecun(x):
    return 1.7159*np.tanh(2.0*x/3)

def grad(x):
    return x*(1-x)

def gradLecun(x):
    t = 2.0*x/3
    return 1.7159*(1-t**2)

def softmax(x):
    x = np.exp(x)
    x = x/x.sum(axis=1)[:, None]
    return x

maxValAcc = 0
lr = 0.0001
trnAcc = []
valAcc = []
tstAcc = []
lst = []

# randomly initialize our weights with mean 0
n_hid_1 = 100
fan_in = 1.0/np.sqrt(784)
# print fan_in
W1 = fan_in*(2*np.random.random((784,n_hid_1)) - 1)
# print W1
bias = fan_in*(2*np.random.random((n_hid_1)) - 1)
fan_in_h = 1.0/np.sqrt(n_hid_1)
W2 = fan_in_h*(2*np.random.random((n_hid_1,10)) - 1)
bias2 = fan_in_h*(2*np.random.random((10)) - 1)

for j in xrange(300):
    indices = np.random.choice(len(trnX), 1000)
    tempY = trnY[indices]
    tempX = trnX[indices]
    A1 = np.dot(tempX, W1) + bias
    l1 = lecun(A1)

    A2 = np.dot(l1, W2) + bias2
    l2 = softmax(A2)

    # Errors in output layer
    d2 = (l2 - tempY)
```

```
        dbias2 = np.sum(d2, axis = 0)

        # Delta of W2
        dW2 = np.dot(l1.T, d2)

        # Errors in 1st hidden layer
        d1 = np.dot(d2, W2.T)*gradLecun(l1)
        dbias = np.sum(d1, axis = 0)

        # Delta W2
        dW1 = np.dot(tempX.T, d1)

        W2 -= lr*dW2
        bias2 -= lr*dbias2
        W1 -= lr*dW1
        bias -= lr*dbias

        prediction = softmax(np.dot(lecun(np.dot(valX, W1)+bias ), W2)+bias2)
        correct = [1 if a == b else 0 for (a, b) in zip(np.argmax(valY, axis =
        valAcc.append(np.sum(correct)*100.0/len(valX))

        prediction = softmax(np.dot(lecun(np.dot(trnX, W1)+bias ), W2)+bias2)
        correct = [1 if a == b else 0 for (a, b) in zip(np.argmax(trnY, axis =
        trnAcc.append(np.sum(correct)*100.0/len(trnX))

        prediction = softmax(np.dot(lecun(np.dot(tstX, W1)+bias ), W2)+bias2)
        correct = [1 if a == b else 0 for (a, b) in zip(np.argmax(tstY, axis =
        tstAcc.append(np.sum(correct)*100.0/len(tstX))

#       if(valAcc[-1] > maxValAcc):
        print tstAcc[-1]
#           maxValAcc = valAcc[-1]
```

```
29.5
36.95
47.8
54.5
60.05
62.05
64.6
66.1
68.0
69.05
70.05
70.35
70.25
71.85
```

72.05
72.0
71.8
72.15
72.8
73.15
73.15
73.85
73.85
74.3
75.6
76.7
77.15
76.7
77.1
76.9
77.5
78.0
78.3
78.3
78.75
79.15
79.6
80.25
80.45
80.7
80.85
80.9
81.0
81.0
81.5
82.05
82.05
81.85
82.1
82.35
82.55
82.4
82.6
82.45
82.55
82.85
83.1
83.25
83.25
83.65
83.6
83.4

83.9
84.15
83.85
84.05
84.6
84.55
84.4
84.15
84.4
84.8
84.9
84.95
84.9
84.7
84.8
84.85
85.1
85.05
84.85
84.95
85.2
85.05
85.4
85.35
85.4
85.2
85.55
85.65
85.6
85.7
85.7
85.8
85.8
85.8
85.7
85.75
85.65
85.7
85.75
85.95
85.95
86.0
86.05
86.1
86.25
86.1
86.3
85.85

85.95
85.85
85.95
86.05
86.25
86.45
86.35
86.4
86.4
86.25
86.5
86.6
86.15
86.5
86.35
86.7
86.55
86.6
86.55
86.55
86.75
86.7
86.7
86.55
86.35
86.65
86.75
86.9
86.8
86.8
86.7
86.85
86.75
86.9
86.9
86.85
86.65
86.6
86.6
87.0
86.95
87.0
87.0
86.85
86.75
87.0
86.95
87.1

86.75
86.85
86.85
86.7
86.75
86.7
86.95
87.05
87.2
87.05
87.1
87.1
87.1
87.1
87.1
87.05
87.05
87.1
87.2
87.25
87.25
87.2
87.35
87.4
87.3
87.4
87.35
87.35
87.45
87.55
87.55
87.55
87.35
87.45
87.35
87.4
87.4
87.45
87.4
87.35
87.55
87.5
87.5
87.45
87.5
87.6
87.45
87.45

87.45
87.6
87.6
87.6
87.35
87.4
87.45
87.45
87.55
87.55
87.5
87.6
87.5
87.65
87.7
87.5
87.65
87.3
87.5
87.4
87.55
87.55
87.9
87.6
87.6
87.55
87.55
87.6
87.4
87.65
87.7
87.85
87.65
87.9
87.9
87.75
87.6
87.75
87.95
87.85
87.95
87.9
87.8
87.75
87.8
88.0
88.1
88.0

88.1
88.0
88.0
88.3
87.9
88.0
88.1
87.95
87.85
87.9
87.85
87.7
87.8
87.9
87.85
88.15
88.2
88.35
87.95
87.95
88.2
87.85
87.9
88.1
88.15
87.85
88.1
88.2
88.15
87.9
88.15
88.15
88.05
88.05
88.15
87.95
88.25
88.1
88.4
88.25
88.3
88.25
88.2
88.2
88.25
88.15

```
In [59]: plt.plot([x+1 for x in range(len(trnAcc))], trnAcc, label = 'Training Accu
         plt.plot([x+1 for x in range(len(valAcc))], valAcc, label = 'Validation Ac
         plt.plot([x+1 for x in range(len(tstAcc))], tstAcc, label = 'Testing Accur
         plt.xlabel("Iterations")
         plt.ylabel("Accuracy")
         plt.legend(loc='lower right', shadow=True)
         plt.show()
```

```
In [ ]: # Gradient Checking
```

```
In [44]: indices = np.random.choice(len(trnX), 3)
         t = trnY[indices]
         d = trnX[indices]
         epsilon = 0.01
         for i,j in np.ndindex(W2.shape):
             e = np.zeros(W2.shape)
             e[i, j] = 0.01
             A1 = np.dot(d, W1) + bias
             l1 = sigmoid(A1)
             A2 = np.dot(l1, W2+e) + bias2
             l2 = softmax(A2)
             # Errors in output layer
             d2 = (l2 - t)
             dbias2 = np.sum(d2, axis = 0)
             # Delta of W2
             dW2 = np.dot(l1.T, d2)
             # Errors in 1st hidden layer
             d1 = np.dot(d2, W2.T+e.T)*grad(l1)
             dbias = np.sum(d1, axis = 0)
             # Delta W2
             dW1 = np.dot(d.T, d1)
             ErrPlus = error(t, l2)

             A1 = np.dot(d, W1) + bias
             l1 = sigmoid(A1)
             A2 = np.dot(l1, W2-e) + bias2
             l2 = softmax(A2)
             # Errors in output layer
             d2 = (l2 - t)
             dbias2 = np.sum(d2, axis = 0)
             # Delta of W2
             dW2 = np.dot(l1.T, d2)
             # Errors in 1st hidden layer
             d1 = np.dot(d2, W2.T-e.T)*grad(l1)
             dbias = np.sum(d1, axis = 0)
             # Delta W2
             dW1 = np.dot(d.T, d1)
```

```
            ErrMinus = error(t, l2)

            A1 = np.dot(d, W1) + bias
            l1 = sigmoid(A1)
            A2 = np.dot(l1, W2) + bias2
            l2 = softmax(A2)
            # Errors in output layer
            d2 = (l2 - t)
            dbias2 = np.sum(d2, axis = 0)
            # Delta of W2
            dW2 = np.dot(l1.T, d2)
            # Errors in 1st hidden layer
            d1 = np.dot(d2, W2.T)*grad(l1)
            dbias = np.sum(d1, axis = 0)
            # Delta W2
            dW1 = np.dot(d.T, d1)

            s.append(dW2[i, j] - (ErrPlus - ErrMinus)/0.02))
```

```
In [11]: def error(trnY, y):
             err = 0
             for i in range(len(trnY)):
                 for k in range(10):
                     if(y[i, k] < 0.00001):
                         y[i, k] = 0.00001
                     elif(y[i, k] > 0.99999):
                         y[i, k] = 0.99999
                     err += trnY[i, k]*np.log(y[i, k])
             err=-1*err/len(trnY)
             return err
```

```
In [45]: print "Average difference = ", np.mean(np.abs(s))
         print "Max difference = ", np.max(np.abs(s))
         print "Min difference = ", np.min(np.abs(s))
```

```
Average difference =  7.46216183773e-05
Max difference =  0.000620140298849
Min difference =  5.36629046164e-09
```

```
In [46]: print np.mean(np.abs(s))
```

```
7.46216183773e-05
```

```
In [43]: s = []
```

```
In [ ]:
```