mp1_main

September 1, 2019

In [1]:
```
import numpy as np
import h5py
import time
import copy
from random import randint
from utils import *
```

In [2]:
```
#load MNIST data

MNIST_data = h5py.File('MNISTdata.hdf5', 'r')
x_train = np.float32(MNIST_data['x_train'][:] )
y_train = np.int32(np.array(MNIST_data['y_train'][:,0]))
x_test = np.float32( MNIST_data['x_test'][:] )
y_test = np.int32( np.array( MNIST_data['y_test'][:,0] ) )
MNIST_data.close()

x_train, x_test = x_train.T, x_test.T
y_train, y_test = np.eye(10)[y_train].T, np.eye(10)[y_test].T
```

In [3]:
```
#Implementation of stochastic gradient descent algorithm

num_inputs = 28*28
num_hiddens = 128
num_outputs = 10

model = {}
model['W'] = np.random.randn(num_hiddens,num_inputs)
model['b1'] = np.random.randn(num_hiddens)
model['C'] = np.random.randn(num_outputs, num_hiddens)
model['b2'] = np.random.randn(num_outputs)
grads = copy.deepcopy(model)

print (x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(784, 60000) (10, 60000) (784, 10000) (10, 10000)
```

In [4]:
```
# Training Process
```

```
import time
time1 = time.time()
LR = .1
num_epochs = 40
for epochs in range(num_epochs):
        #Learning rate schedule
        if (epochs > 10):
                LR = 0.01
        if (epochs > 20):
                LR = 0.001
        if (epochs > 30):
                LR = 0.0001
        total_correct = 0
        for n in range(x_train.shape[1] // 5):
                y = y_train[:, 5*n: 5*n+5]
                x = x_train[:, 5*n: 5*n+5]
                cache = feed_forward(x, model)
                prediction = np.argmax(cache['R'], axis=0)
                total_correct += np.sum(prediction == np.argmax(y, axis=0))
                grads = back_propagate(x, y, model, cache)
                model['W'] = model['W'] - LR * grads['W']
                model['C'] = model['C'] - LR * grads['C']
                model['b1'] = model['b1'] - LR * grads['b1']
                model['b2'] = model['b2'] - LR * grads['b2']
        print(total_correct/np.float(x_train.shape[1]))
time2 = time.time()
print(time2-time1)
```

```
0.8822333333333333
0.9392333333333334
0.9545
0.9628
0.9689833333333333
0.97345
0.9787
0.9817166666666667
0.9827166666666667
0.98395
0.9877
0.99075
0.99455
0.9958
0.9967166666666667
0.9974
0.9977833333333334
0.9980166666666667
0.9982833333333333
0.99835
```

```
0.9985
0.9986333333333334
0.9988
0.9988333333333334
0.99885
0.9988666666666667
0.9988833333333333
0.9988833333333333
0.9988833333333333
0.9988833333333333
0.9988833333333333
0.9989333333333333
0.9989333333333333
0.9989333333333333
0.9989333333333333
0.99895
0.99895
0.99895
0.99895
0.99895
335.02962589263916
```

In [5]:
```
#Testing Data

        total_correct = 0
        for n in range( x_test.shape[1] ):
                y = np.expand_dims(y_test[:, n], axis=1)
                x = np.expand_dims(x_test[:, n], axis=1)
                cache = feed_forward(x, model)
                prediction = np.argmax(cache['R'])
                if (prediction == np.argmax(y)):
                        total_correct += 1
        print(total_correct/np.float(x_test.shape[1]) )
```

```
0.9709
```

In [ ]: