In [2]:

```
from __future__ import print_function
import torch

from mnist import MNIST
import numpy as np
# import matplotlib.pyplot as plt
# from scipy import linalg
```

In [3]:

```
def load_dataset():
    mndata = MNIST('/home/mumu/Desktop/CSE546/hw2/data/python-mnist/data')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0
    return X_test,labels_test,X_train,labels_train
```

In [4]:

```
def one_hot_encode(vector):
    n_classes = len(vector.unique())  # 1
    one_hot = torch.zeros((vector.shape[0], n_classes))\
        .type(torch.LongTensor)  # 2
    return one_hot\
        .scatter(1, vector.type(torch.LongTensor).unsqueeze(1), 1)  # 3
```

In [5]:

```
X_test,labels_test,X_train,labels_train=load_dataset()
y_train = torch.from_numpy(labels_train)
y_one_hot = one_hot_encode(y_train)
n,d = X_train.shape
X_train_tensor = torch.from_numpy(X_train).double()
W = torch.rand(d, 10, requires_grad=True).double()
```

In [ ]:

```
################################################################################
###########################
################################ Multinomial Logistic Regression ###############
###########################
################################################################################
###########################
```

In [34]:

```
epochs=100000
MLR_w_autograd = torch.zeros((d, 10), requires_grad=True)
step_size = 0.1
for epoch in range(epochs):
    y_hat = torch.matmul(X_train_tensor, MLR_w_autograd.double())
    # cross entropy combines softmax calculation with NLLLoss

    loss = torch.nn.functional.cross_entropy(y_hat, y_train.long())
    # computes derivatives of the loss with respect to W

    loss.backward()
    #print(torch.max(w_autograd.grad))
    MLR_w_autograd.data = MLR_w_autograd.data - step_size * MLR_w_autograd.grad

    if (torch.max(torch.abs((MLR_w_autograd.grad))) < 0.001):
        print(torch.max(MLR_w_autograd.grad))
        print(MLR_w_autograd)
        break
    if (epoch % 100 == 0):
        print("the max of the {} the iteration is {}".format(epoch,torch.max(torc
h.abs(MLR_w_autograd.grad))))

    MLR_w_autograd.grad.zero_()
```

```
the max of the 0 the iteration is 0.06373168528079987
the max of the 100 the iteration is 0.010296175256371498
the max of the 200 the iteration is 0.005794010125100613
the max of the 300 the iteration is 0.004077061545103788
the max of the 400 the iteration is 0.0031427759677171707
the max of the 500 the iteration is 0.002765779849141836
the max of the 600 the iteration is 0.0025410777889192104
the max of the 700 the iteration is 0.002359171863645315
the max of the 800 the iteration is 0.002207828452810645
the max of the 900 the iteration is 0.002079244237393141
the max of the 1000 the iteration is 0.00196815631352365
the max of the 1100 the iteration is 0.0018708654679358006
the max of the 1200 the iteration is 0.001784683670848608
the max of the 1300 the iteration is 0.00170760543551296
the max of the 1400 the iteration is 0.0016381009481847286
the max of the 1500 the iteration is 0.00157497962936759
the max of the 1600 the iteration is 0.0015172993298619986
the max of the 1700 the iteration is 0.0014643047470599413
the max of the 1800 the iteration is 0.0014153801603242755
the max of the 1900 the iteration is 0.001370019861496985
the max of the 2000 the iteration is 0.001327802543528378
the max of the 2100 the iteration is 0.0012883737217634916
the max of the 2200 the iteration is 0.0012514350237324834
the max of the 2300 the iteration is 0.0012167301028966904
the max of the 2400 the iteration is 0.0011840392835438251
the max of the 2500 the iteration is 0.0011531729251146317
the max of the 2600 the iteration is 0.0011239650193601847
the max of the 2700 the iteration is 0.0010962699307128787
the max of the 2800 the iteration is 0.001069961697794497
the max of the 2900 the iteration is 0.001044926350004971
the max of the 3000 the iteration is 0.0010210640029981732
tensor(0.0007)
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]], requires_grad=True)
```

In [44]:

```
############################## PREDICT FOR MULTINOMIAL REG ########################
###############
print(MLR_w_autograd.shape)
print(X_train_tensor.shape)
predict = torch.matmul(X_train_tensor.double(),MLR_w_autograd.double())
MLR_predicted_labels_train = torch.argmax(predict,axis=1)
print(labels_train.shape[0])
MLR_train_accuracy = float(sum(MLR_predicted_labels_train== torch.from_numpy(labe
ls_train))) / labels_train.shape[0]
print("\nFinal Train set  Accuracy for Multinomial Regression is : {}".format(MLR
_train_accuracy))
```

```
torch.Size([784, 10])
torch.Size([60000, 784])
60000

Final Train set  Accuracy for Multinomial Regression is : 0.9151833333
333333
```

In [45]:

```
############################## PREDICT FOR MULTINOMIAL REG ########################
###############
X_test_tensor = torch.from_numpy(X_test)
#print(MLR_w_autograd.shape)
#print(X_test_tensor.shape)
predict_test = torch.matmul(X_test_tensor.double(),MLR_w_autograd.double())
MLR_predicted_labels_test = torch.argmax(predict_test,axis=1)
#print(MLR_predicted_labels_test.shape)
MLR_train_accuracy_test = float(sum(MLR_predicted_labels_test== torch.from_numpy(
labels_test))) / labels_test.shape[0]
print("\nFinal test set Accuracy for Multinomial Regression is : {}".format(MLR_t
rain_accuracy_test))
```

```
Final test set Accuracy for Multinomial Regression is : 0.9182
```

In [37]:

```
#################################################################################
#
###################### MSE loss ##################################################
#
#################################################################################
#
```

In [38]:

```
epochs=10000
MSE_w_autograd = torch.zeros((d, 10), requires_grad=True)
step_size = 0.1
for epoch in range(epochs):
    y_hat = torch.matmul(X_train_tensor, MSE_w_autograd.double())
    # cross entropy combines softmax calculation with NLLoss

    loss = torch.nn.functional.mse_loss(y_hat.double(), y_one_hot.double())
    # computes derivatives of the loss with respect to W

    loss.backward()
    #print(torch.max(w_autograd.grad))
    MSE_w_autograd.data = MSE_w_autograd.data - step_size * MSE_w_autograd.grad
    if (torch.max(torch.abs((MSE_w_autograd.grad))) < 0.001):
        print(torch.max(MSE_w_autograd.grad))
        print(MSE_w_autograd)
        break
    if (epoch % 100 == 0):
        print("the max of the {} the iteration is {}".format(epoch,torch.max(torc
h.abs(MSE_w_autograd.grad))))

    MSE_w_autograd.grad.zero_()
```

```
the max of the 0 the iteration is 0.021706614643335342
the max of the 100 the iteration is 0.0017530462937429547
the max of the 200 the iteration is 0.0011112616630271077
tensor(0.0007)
tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]], requires_grad=True)
```

In [48]:

```
############################ PREDICT FOR MULTINOMIAL REG #######################
###############
print(MSE_w_autograd.shape)
print(X_train_tensor.shape)
predict = torch.matmul(X_train_tensor.double(),MSE_w_autograd.double())
MSE_predicted_labels_train = torch.argmax(predict,axis=1)
print(labels_train.shape[0])
MSE_test_accuracy = float(sum(MSE_predicted_labels_train== torch.from_numpy(label
s_train))) / labels_train.shape[0]
print("\nFinal Trainning set Accuracy for MSE is : {}".format(MSE_test_accuracy))
```

```
torch.Size([784, 10])
torch.Size([60000, 784])
60000

Final Trainning set Accuracy for MSE is : 0.8448666666666667
```

In [47]:

```
############################## PREDICT FOR MULTINOMIAL REG ########################
###############
predict_test = torch.matmul(X_test_tensor.double(),MSE_w_autograd.double())
MSE_predicted_labels_test = torch.argmax(predict_test,axis=1)
#print(MLR_predicted_labels_test.shape)
MSE_train_accuracy_test = float(sum(MSE_predicted_labels_test== torch.from_numpy(
labels_test))) / labels_test.shape[0]
print("\nFinal test set Accuracy for Least square is : {}".format(MSE_train_accur
acy_test))
```

```
Final test set Accuracy for Least square is : 0.8519
```

In [ ]: