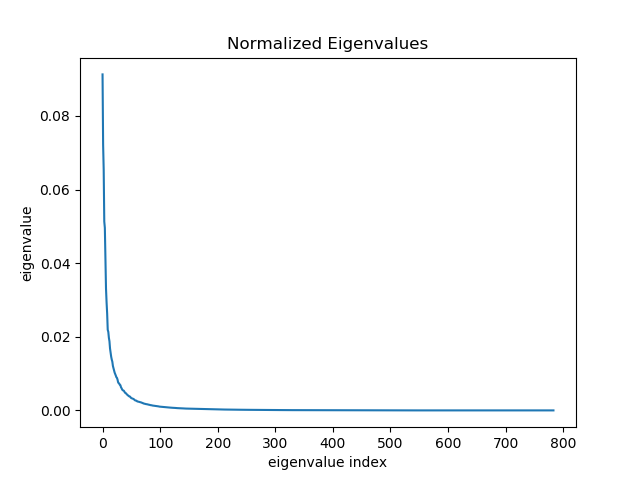
CMSC422

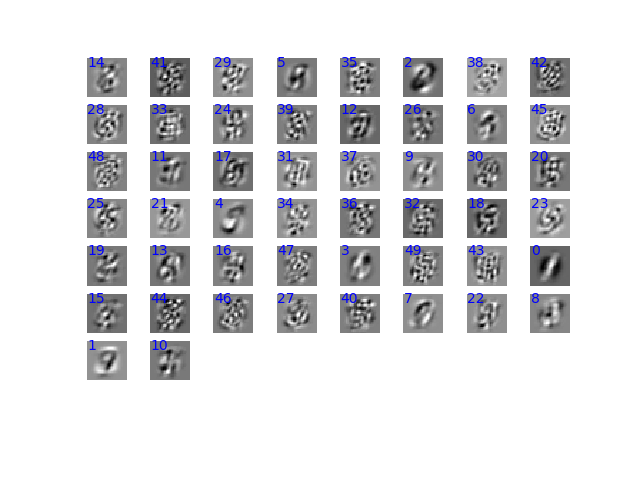
Project3 Writeup

Qpca2

We have to include 81 eigenvectors to account for 90% of the variance and 135 eigenvectors to account for 95%.



Qpca3



Most of these don’t look like digits, but a few of them seem to resemble digits. This result is expected, because the top eigenvectors account for most of the variance in the data, so we extract information that is only a partial description of the data. Therefore, the images don’t look like digits.

Qsr1

1. The dimension of W is the number of classes (rows) by the number of features of x (columns). The dimension of X is the number of features of x (rows) by the number of samples (columns). The dimension of WX is number of classes (rows) by the number of samples (columns).

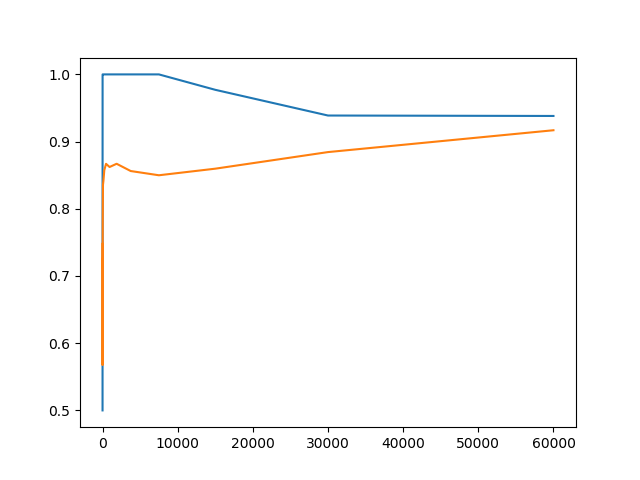
Qsr3

1. Assume the maximum one in W\_X is called WM, in the formula, exp(WM) would be divided by both numerator and denominator, which can be cancelled out in the calculation. As a result, the process of subtracting WM would not influence the probabilities.

= = P[y=i]

1. When calculating exponentials, the value of exponentials grows very fast, which is called “blow up of exponentials”. By subtracting the maximum of W\_X, the indices are limited no greater than 0, and all the exponentials are no greater than 1, which guarantees that no “blow up of exponentials” could happen.

Qsr4



Blue: trainAcc

Yellow: testAcc

There is an overfitting at the beginning since the training accuracy is greater than the test accuracy. When the data size increases, the difference between training accuracy and test accuracy gets smaller.

Qnn1.3

activation: Relu

loss function: SquaredLoss

final accuracy: dev\_acc:0.96150

Qnn1.4

When all the weights are initialized to 0, every hidden unit will get 0 signal, no matter what the actual input is. During forward propagation each unit in hidden layer gets signal:

enter image description here

If all the weights are the same, the units in hidden layer will be the same too. As a solution, initializing the weights with small random numbers can solve this problem.

Qnn2

(3)

def update(self, grad\_Ws, grad\_bs, learning\_rate):

# Update the weights and biases

num\_layers = len(grad\_Ws)

ws = self.weights

bs = self.biases

for idx in range(num\_layers):

ws[idx] -= (grad\_Ws[idx] \* learning\_rate/(idx+1))

bs[idx] -= (grad\_bs[idx] \* learning\_rate/(idx+1))

self.weights = ws

self.biases = bs

return

No, this new method has no obvious advantage comparing with the given one. For different layers, I tried to change the learning rate by taking the index into consideration, so that the deeper the layer, the less the values would change. I thought in this way the performance would be better, but the result doesn’t show any significant advantage.

Original function’s result:

activation:Relu

loss function:SquaredLoss

Layer 1 w:(256, 784) b:(256, 1)

Layer 2 w:(256, 256) b:(256, 1)

Layer 3 w:(10, 256) b:(10, 1)

Epoch 1/20 loss:1.20831 dev\_acc:0.92460

Epoch 2/20 loss:0.63683 dev\_acc:0.93970

Epoch 3/20 loss:0.66635 dev\_acc:0.94450

Epoch 4/20 loss:0.57863 dev\_acc:0.95140

Epoch 5/20 loss:0.50285 dev\_acc:0.95400

Epoch 6/20 loss:0.58866 dev\_acc:0.95710

Epoch 7/20 loss:0.66927 dev\_acc:0.96090

Epoch 8/20 loss:0.39660 dev\_acc:0.96290

Epoch 9/20 loss:0.46738 dev\_acc:0.96360

Epoch 10/20 loss:0.43955 dev\_acc:0.96490

Epoch 11/20 loss:0.36301 dev\_acc:0.96600

Epoch 12/20 loss:0.35435 dev\_acc:0.96640

Epoch 13/20 loss:0.24128 dev\_acc:0.96680

Epoch 14/20 loss:0.47372 dev\_acc:0.96920

Epoch 15/20 loss:0.37755 dev\_acc:0.97010

Epoch 16/20 loss:0.34421 dev\_acc:0.96940

Epoch 17/20 loss:0.34435 dev\_acc:0.97050

Epoch 18/20 loss:0.26925 dev\_acc:0.97070

Epoch 19/20 loss:0.30662 dev\_acc:0.97110

Epoch 20/20 loss:0.43017 dev\_acc:0.97180

My function’s result:

activation:Relu

loss function:SquaredLoss

Layer 1 w:(256, 784) b:(256, 1)

Layer 2 w:(256, 256) b:(256, 1)

Layer 3 w:(10, 256) b:(10, 1)

Epoch 1/20 loss:1.07321 dev\_acc:0.92860

Epoch 2/20 loss:0.90858 dev\_acc:0.94470

Epoch 3/20 loss:0.74531 dev\_acc:0.95260

Epoch 4/20 loss:0.71737 dev\_acc:0.95730

Epoch 5/20 loss:0.51703 dev\_acc:0.95800

Epoch 6/20 loss:0.51655 dev\_acc:0.96020

Epoch 7/20 loss:0.60157 dev\_acc:0.96290

Epoch 8/20 loss:0.66921 dev\_acc:0.96480

Epoch 9/20 loss:0.40088 dev\_acc:0.96490

Epoch 10/20 loss:0.50334 dev\_acc:0.96710

Epoch 11/20 loss:0.50731 dev\_acc:0.96690

Epoch 12/20 loss:0.58830 dev\_acc:0.96840

Epoch 13/20 loss:0.36116 dev\_acc:0.96980

Epoch 14/20 loss:0.40988 dev\_acc:0.96970

Epoch 15/20 loss:0.41188 dev\_acc:0.97100

Epoch 16/20 loss:0.57620 dev\_acc:0.97050

Epoch 17/20 loss:0.34612 dev\_acc:0.97110

Epoch 18/20 loss:0.43331 dev\_acc:0.97160

Epoch 19/20 loss:0.49971 dev\_acc:0.97210

Epoch 20/20 loss:0.33779 dev\_acc:0.97200