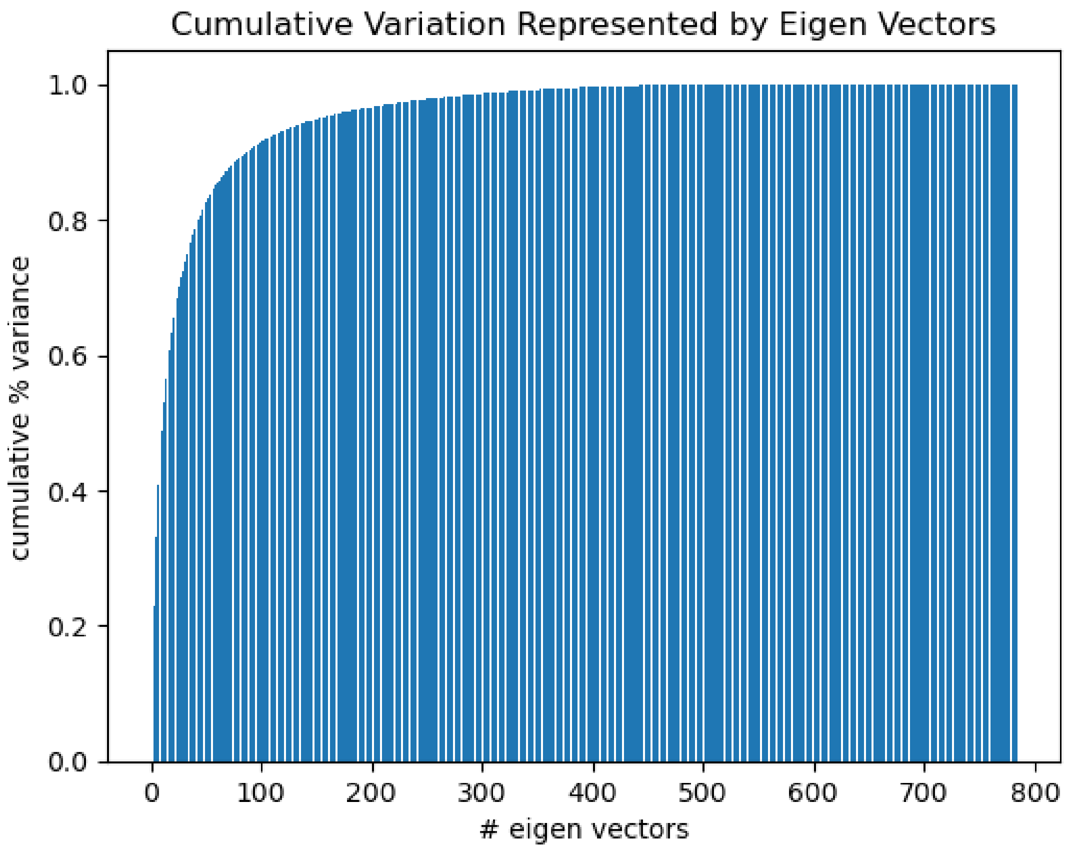
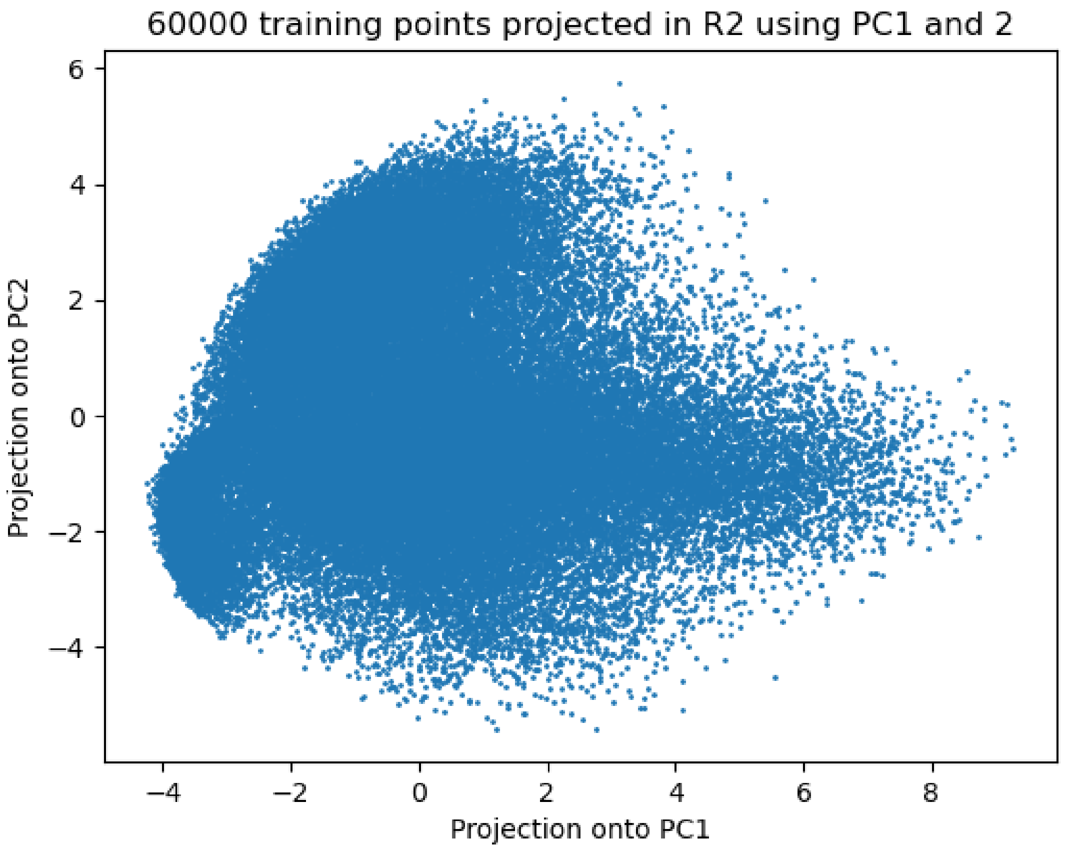
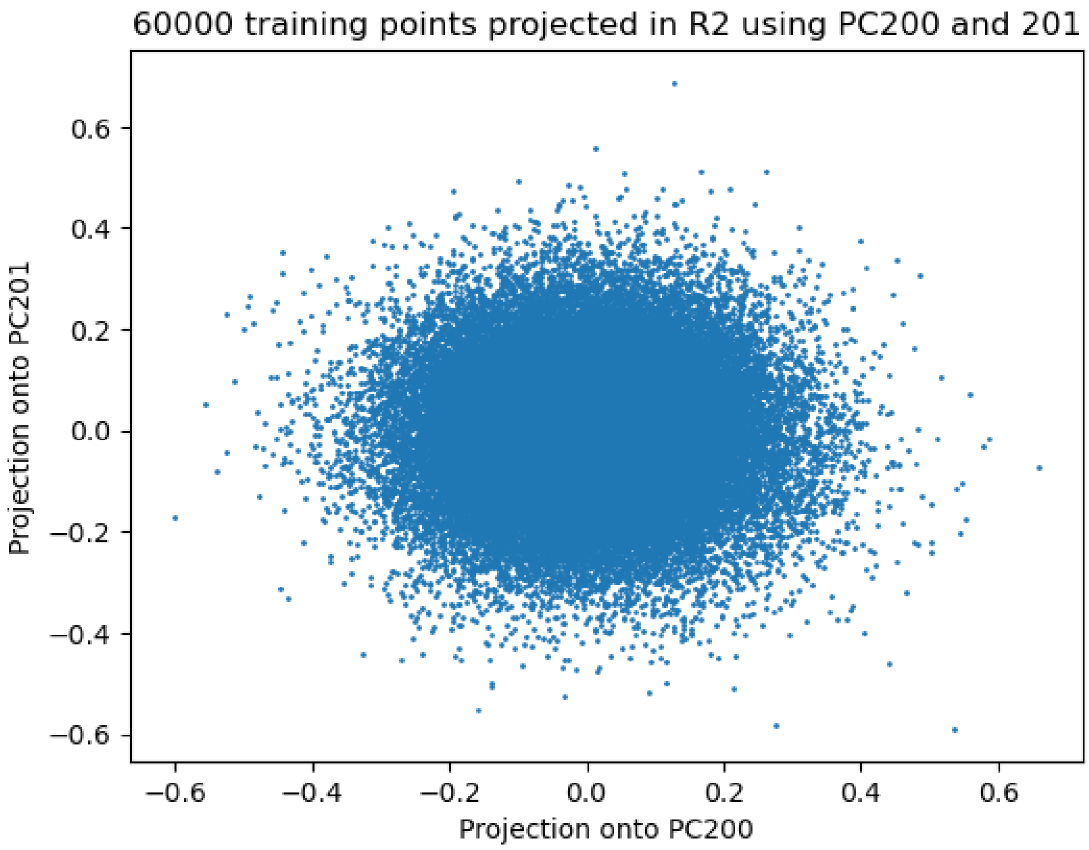
# Qnn2_extracredit

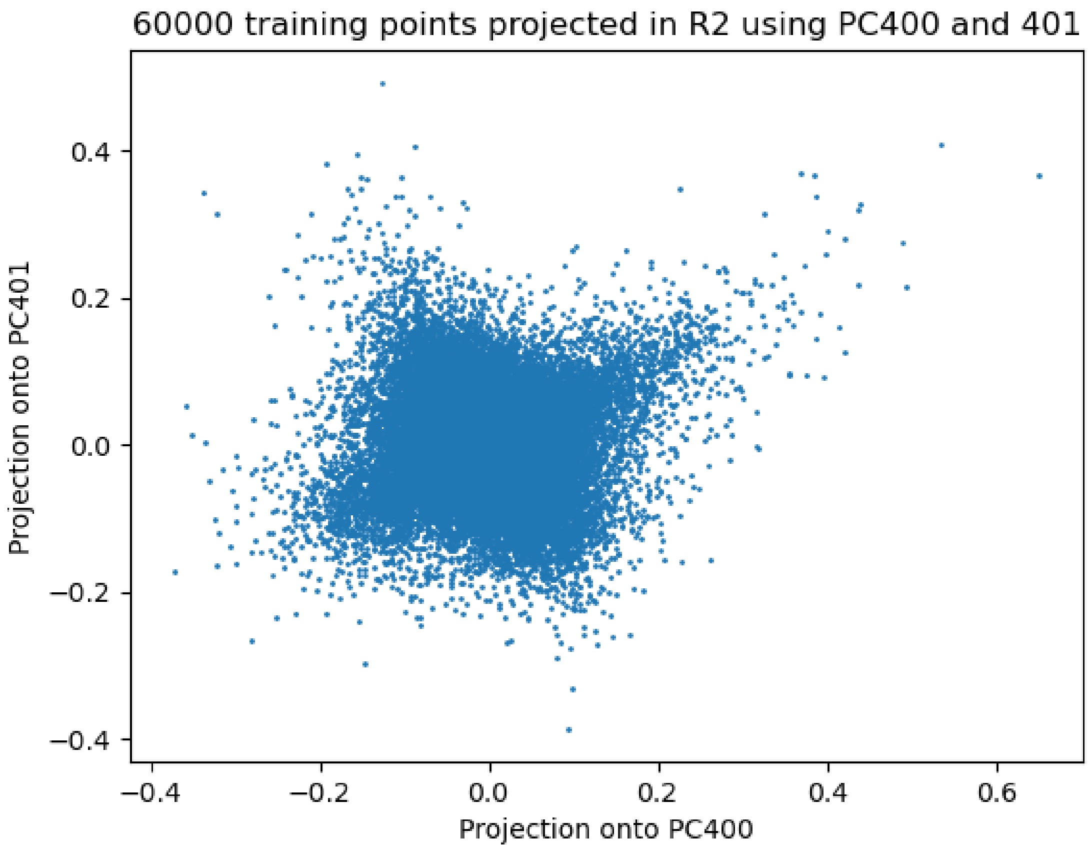
December 10, 2023

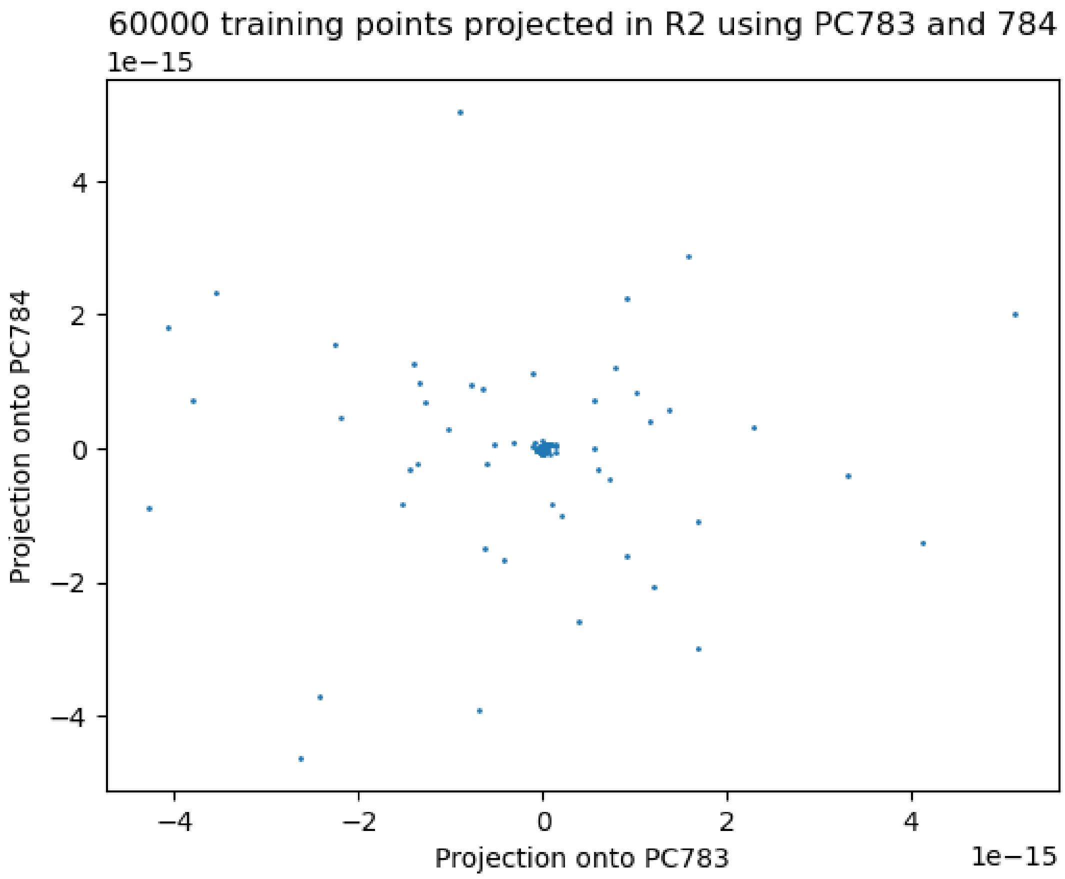
[2]:
```
from sklearn.decomposition import PCA
from matplotlib.pyplot import *
from timeit import default_timer as timer

import numpy as np
from nn import NN
from nn import Relu, Linear, SquaredLoss, CELoss
from utils import data_loader, acc, save_plot, loadMNIST, onehot

# Several passes of the training data
def train(model, training_data, dev_data, learning_rate, batch_size, max_epoch):
    X_train, Y_train = training_data['X'], training_data['Y']
    X_dev, Y_dev = dev_data['X'], dev_data['Y']
    for i in range(max_epoch):
        for X,Y in data_loader(X_train, Y_train, batch_size=batch_size,␣
 ↪shuffle=True):
            training_loss, grad_Ws, grad_bs = model.compute_gradients(X, Y)
            model.update(grad_Ws, grad_bs, learning_rate)
        dev_acc = acc(model.predict(X_dev), Y_dev)
        print("Epoch {: >3d}/{}\tloss:{:.5f}\tdev_acc:{:.5f}".
 ↪format(i+1,max_epoch,training_loss, dev_acc))
    return model

# One pass of the training data
def train_1pass(model, training_data, dev_data, learning_rate, batch_size,␣
 ↪print_every=100, plot_every=10):
    X_train, Y_train = training_data['X'], training_data['Y']
    X_dev, Y_dev = dev_data['X'], dev_data['Y']

    num_samples = 0
    print_loss_total = 0
    plot_loss_total = 0

    plot_losses = []
    plot_num_samples = []
    for idx, (X,Y) in enumerate(data_loader(X_train, Y_train,␣
 ↪batch_size=batch_size, shuffle=True),1):
        training_loss, grad_Ws, grad_bs = model.compute_gradients(X, Y)
        model.update(grad_Ws, grad_bs, learning_rate)
        num_samples += Y.shape[1]
        print_loss_total += training_loss
        plot_loss_total += training_loss

        if idx % print_every == 0:
            dev_acc = acc(model.predict(X_dev), Y_dev)
            print_loss_avg = print_loss_total/print_every
            print_loss_total = 0
            print("#Samples {: >5d}\tloss:{:.5f}\tdev_acc:{:.5f}".
 ↪format(num_samples, print_loss_avg, dev_acc))
        if idx % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_loss_total = 0
            plot_losses.append(plot_loss_avg)
            plot_num_samples.append(num_samples)

    return model, {"losses":plot_losses, "num_samples":plot_num_samples}

if __name__ == "__main__":
    x_train, label_train = loadMNIST('data/train-images.idx3-ubyte', 'data/
 ↪train-labels.idx1-ubyte')
    x_test, label_test = loadMNIST('data/t10k-images.idx3-ubyte', 'data/
 ↪t10k-labels.idx1-ubyte')
    y_train = onehot(label_train)
    y_test = onehot(label_test)

    dr = PCA()
    #no need to scale data since the features (i.e. pixels) are on the same␣
 ↪scale
    #need to transpose x since pca requires (N x D)
    full_train = dr.fit_transform(x_train.T)
    #must transform x_test into same space/dimensions as x_train
    #note we cannot perform pca on x_train + x_test together since
    #we are not allowed to use information from x_test during training
    #also we do not perform pca on x_test separated as we would get
    #different principal components (i.e. a projection into a different space)
    #so we use the principal components from x_train
    full_test = dr.transform(x_test.T)

    #get cumulative variance explained
    variances = np.cumsum(dr.explained_variance_ratio_)
    bar(range(1,785), height=variances)
    title("Cumulative Variation Represented by Eigen Vectors")
    xlabel("# eigen vectors")
    ylabel("cumulative % variance")
    show()

    #plot projections onto different pairs of principle components
    components = [(1,2), (200, 201), (400, 401), (783,784)]
    for c1,c2 in components:
        scatter(full_train[:,c1-1], full_train[:,c2-1],s=1)
        title(f"{full_train.shape[0]} training points projected in R2 using␣
↪PC{c1} and {c2}")
        xlabel(f"Projection onto PC{c1}")
        ylabel(f"Projection onto PC{c2}")
        show()


    #what proportion of variance should the reduced data maintain
    proportions = [0.10, 0.25, 0.50, 0.80, 0.90, 0.99]
    dr_xtrain = []
    dr_xtest = []
    for p in proportions:
        #num of eigen vectors necessary to explain at least p of the variance
        evs = np.argmax(variances >= p) + 1
        #must take transpose since PCA requires different format of data than NN
        dr_xtrain.append(full_train[:, :evs].T)
        dr_xtest.append(full_test[:, :evs].T)

    lr = 1e-2
    max_epoch = 20
    batch_size = 128

    #track train + prediction times
    train_times = []
    test_times = []
    #track test accuracies
    accuracies = []
    for x_train,x_test in zip(dr_xtrain,dr_xtest):
        #y data can be left untouched, don't need to map into predictor space
        training_data = {"X":x_train, "Y":y_train}
        dev_data = {"X":x_test, "Y":y_test}

        model = NN(Relu(), SquaredLoss(), hidden_layers=[256, 256],␣
↪input_d=x_train.shape[0], output_d=10)
        model.print_model()

        # model, plot_dict = train_1pass(model, training_data, dev_data, lr,␣
↪batch_size)
        # save_plot(plot_dict["num_samples"], plot_dict["losses"])
        start_train = timer()
        model = train(model, training_data, dev_data, lr, batch_size, max_epoch)
        end_train = timer()

        start_test = timer()
        accuracy = acc(model.predict(x_test), y_test)
        end_test = timer()

        train_times.append((end_train-start_train))
        test_times.append((end_test-start_test))
        accuracies.append(accuracy)

plot(dimensions, train_times, marker='o')
title("NN Train Time vs Dimension")
xlabel("Dimension")
ylabel("Time in Seconds")
show()
plot(dimensions, test_times, marker='o')
title("NN Test Time vs Dimension")
xlabel("Dimension")
ylabel("Time in Seconds")
show()
plot(dimensions, accuracies, marker='o')
title("NN Accuracy on Test Set vs Dimension")
xlabel("Dimension")
ylabel("Accuracy")
show()
plot(proportions, accuracies, marker='o')
title("NN Accuracy on Test Set vs Proportion of Variance")
xlabel("Proportion of Variance")
ylabel("Accuracy")
show()
```


Cumulative Variation Represented by Eigen Vectors
cumulative % variance
# eigen vectors
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500
600
700
800

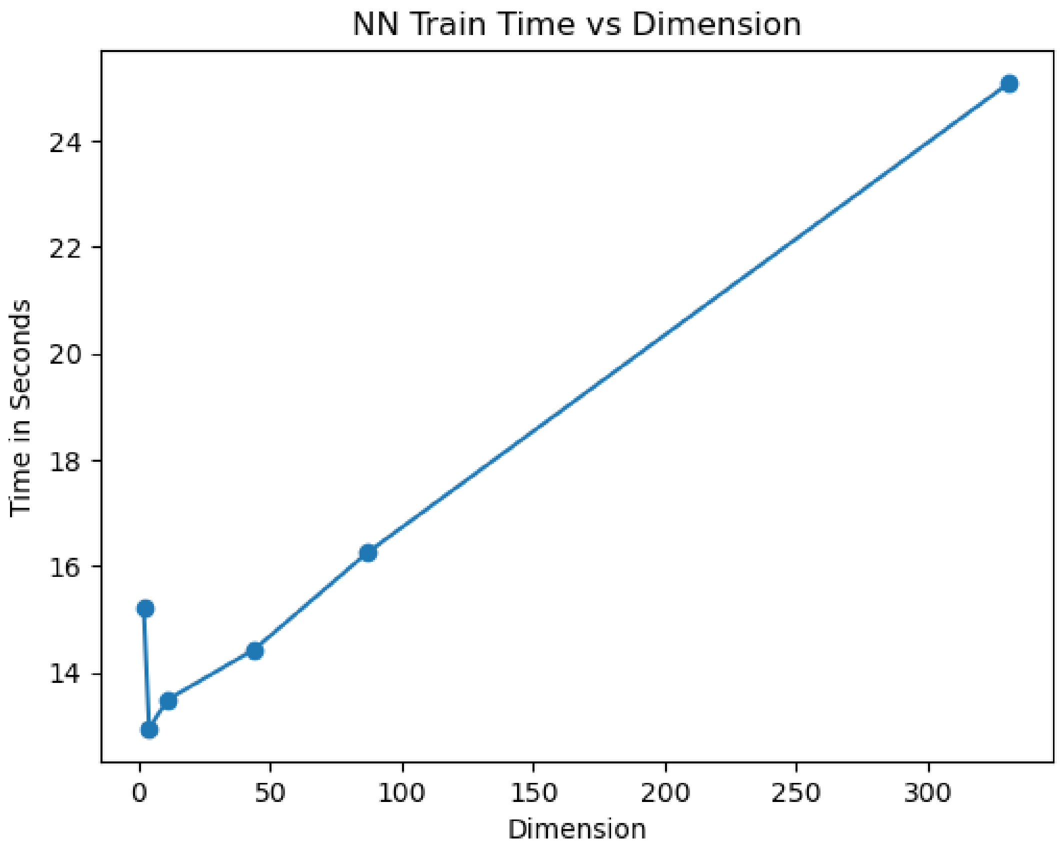

60000 training points projected in R2 using PC1 and 2
Projection onto PC1
Projection onto PC2
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8

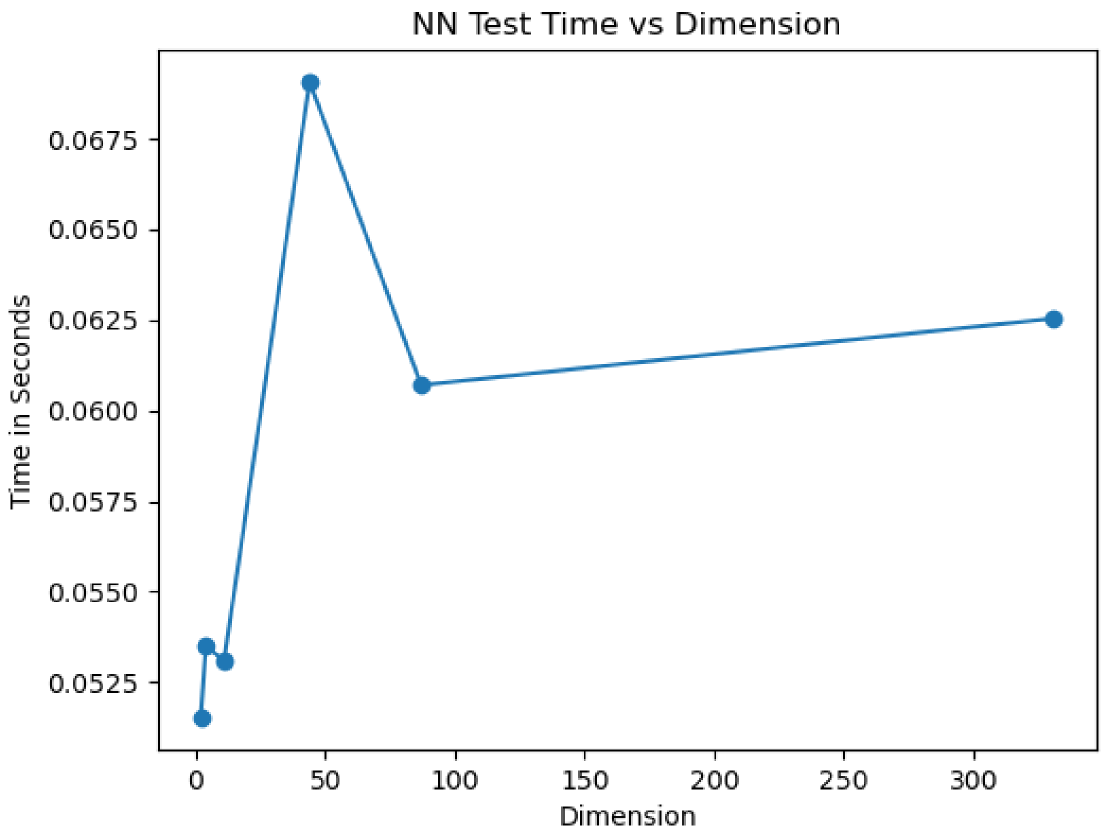

60000 training points projected in R2 using PC200 and 201
Projection onto PC201
Projection onto PC200
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

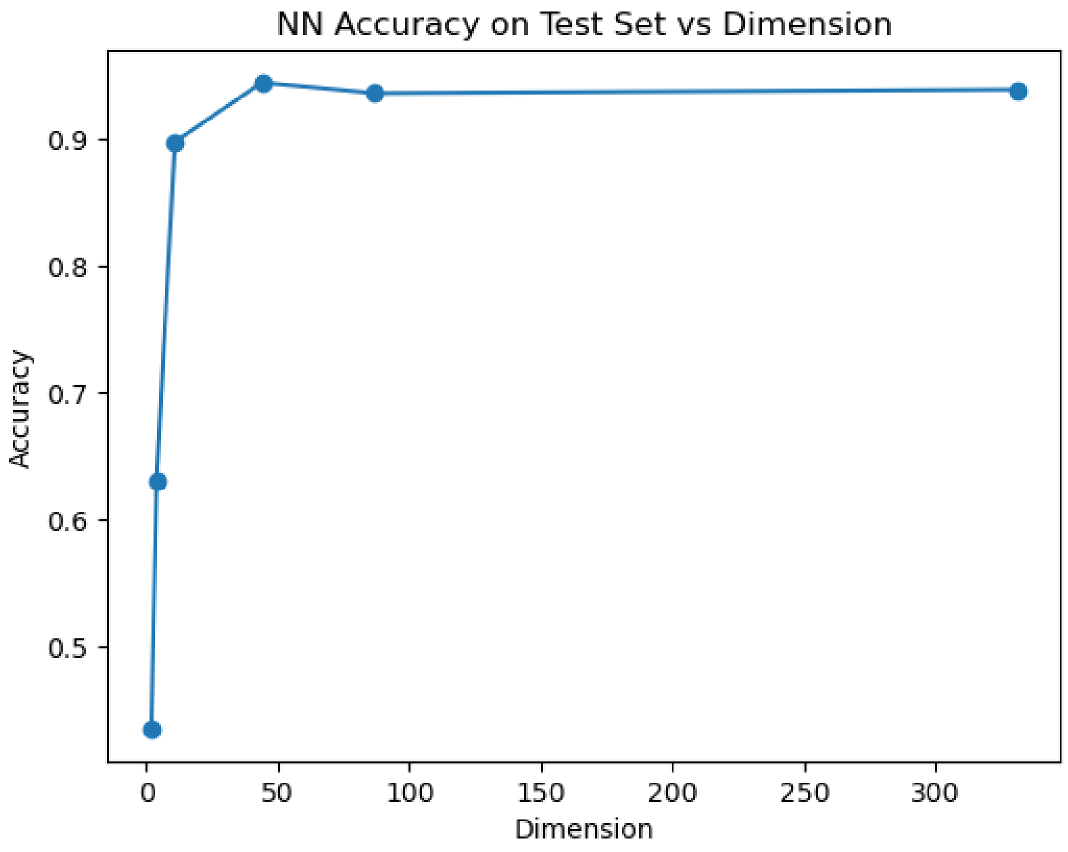

60000 training points projected in R2 using PC400 and 401
Projection onto PC401
Projection onto PC400
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6

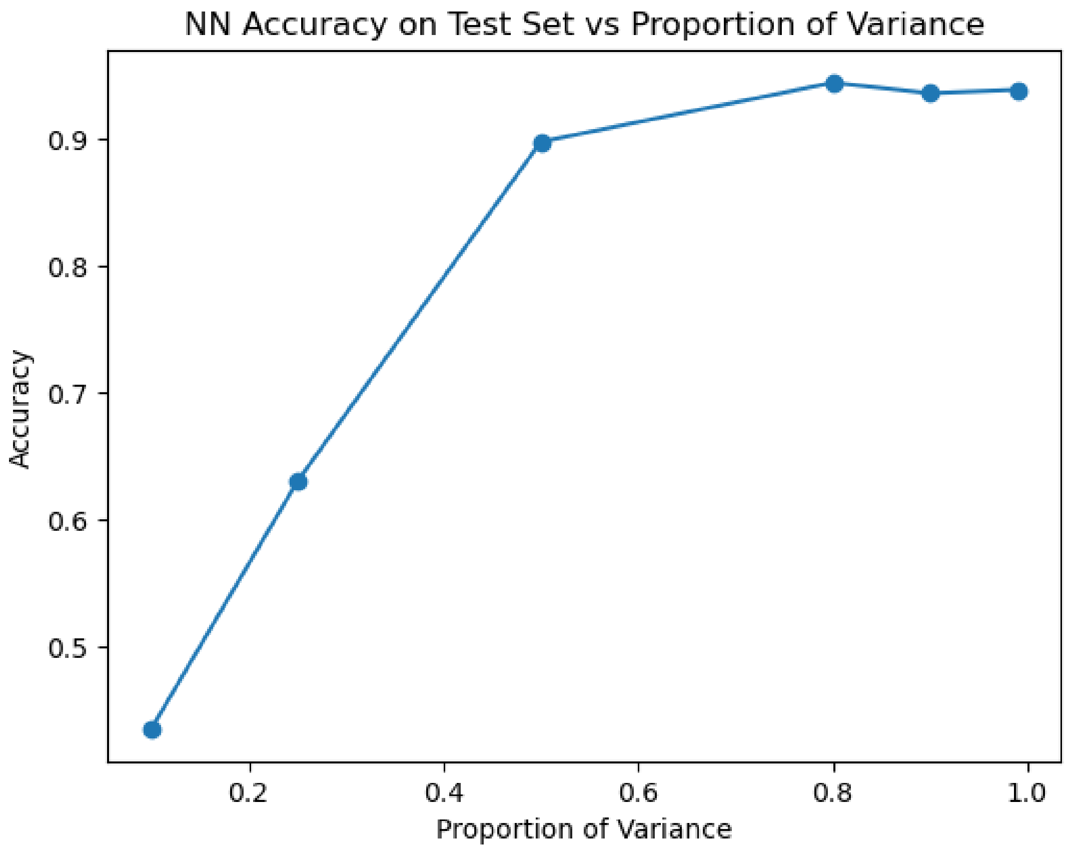


```
activation:Relu
loss function:SquaredLoss
Layer 1 w:(256, 2)      b:(256, 1)
Layer 2 w:(256, 256)    b:(256, 1)
Layer 3 w:(10, 256)     b:(10, 1)
Epoch   1/20    loss:0.36894    dev_acc:0.40800
Epoch   2/20    loss:0.31951    dev_acc:0.41640
Epoch   3/20    loss:0.35647    dev_acc:0.42810
Epoch   4/20    loss:0.36225    dev_acc:0.43190
Epoch   5/20    loss:0.35008    dev_acc:0.43290
Epoch   6/20    loss:0.34070    dev_acc:0.43690
Epoch   7/20    loss:0.29937    dev_acc:0.43570
Epoch   8/20    loss:0.33532    dev_acc:0.43440
Epoch   9/20    loss:0.32132    dev_acc:0.43450
Epoch  10/20    loss:0.32183    dev_acc:0.43870
Epoch  11/20    loss:0.33723    dev_acc:0.43730
Epoch  12/20    loss:0.31511    dev_acc:0.43400
Epoch  13/20    loss:0.36144    dev_acc:0.43850
Epoch  14/20    loss:0.32491    dev_acc:0.43490
Epoch  15/20    loss:0.30594    dev_acc:0.43120
Epoch  16/20    loss:0.35773    dev_acc:0.43540
Epoch  17/20    loss:0.31421    dev_acc:0.43490
Epoch  18/20    loss:0.32615    dev_acc:0.43530
Epoch  19/20    loss:0.32410    dev_acc:0.43500
Epoch  20/20    loss:0.32848    dev_acc:0.43560
activation:Relu
loss function:SquaredLoss
Layer 1 w:(256, 4)      b:(256, 1)
Layer 2 w:(256, 256)    b:(256, 1)
Layer 3 w:(10, 256)     b:(10, 1)
Epoch   1/20    loss:0.33211    dev_acc:0.56230
Epoch   2/20    loss:0.26983    dev_acc:0.58580
Epoch   3/20    loss:0.29336    dev_acc:0.60050
Epoch   4/20    loss:0.27551    dev_acc:0.60600
Epoch   5/20    loss:0.25036    dev_acc:0.60860
Epoch   6/20    loss:0.24512    dev_acc:0.61140
Epoch   7/20    loss:0.22633    dev_acc:0.61210
Epoch   8/20    loss:0.25574    dev_acc:0.61400
Epoch   9/20    loss:0.25584    dev_acc:0.61360
Epoch  10/20    loss:0.24913    dev_acc:0.61730
Epoch  11/20    loss:0.25191    dev_acc:0.62020
Epoch  12/20    loss:0.26015    dev_acc:0.61810
Epoch  13/20    loss:0.27953    dev_acc:0.62270
Epoch  14/20    loss:0.26127    dev_acc:0.62270
Epoch  15/20    loss:0.23365    dev_acc:0.62570
Epoch  16/20    loss:0.23718    dev_acc:0.62480
Epoch  17/20    loss:0.20384    dev_acc:0.62640
Epoch  18/20    loss:0.23292    dev_acc:0.62990
Epoch  19/20    loss:0.25073    dev_acc:0.63280
Epoch  20/20    loss:0.25028    dev_acc:0.63040
activation:Relu
loss function:SquaredLoss
Layer 1 w:(256, 11)     b:(256, 1)
Layer 2 w:(256, 256)    b:(256, 1)
Layer 3 w:(10, 256)     b:(10, 1)
Epoch   1/20    loss:0.21811    dev_acc:0.75630
Epoch   2/20    loss:0.19837    dev_acc:0.81490
Epoch   3/20    loss:0.15654    dev_acc:0.83790
Epoch   4/20    loss:0.17383    dev_acc:0.84930
Epoch   5/20    loss:0.10895    dev_acc:0.85700
Epoch   6/20    loss:0.11517    dev_acc:0.86380
Epoch   7/20    loss:0.13261    dev_acc:0.86950
Epoch   8/20    loss:0.10374    dev_acc:0.87520
Epoch   9/20    loss:0.14593    dev_acc:0.87860
Epoch  10/20    loss:0.10694    dev_acc:0.88150
Epoch  11/20    loss:0.12225    dev_acc:0.88500
Epoch  12/20    loss:0.10560    dev_acc:0.88670
Epoch  13/20    loss:0.10249    dev_acc:0.88920
Epoch  14/20    loss:0.08595    dev_acc:0.89100
Epoch  15/20    loss:0.12218    dev_acc:0.89160
Epoch  16/20    loss:0.12055    dev_acc:0.89310
Epoch  17/20    loss:0.06458    dev_acc:0.89480
Epoch  18/20    loss:0.13032    dev_acc:0.89680
Epoch  19/20    loss:0.10664    dev_acc:0.89790
Epoch  20/20    loss:0.10433    dev_acc:0.89820
activation:Relu
loss function:SquaredLoss
Layer 1 w:(256, 44)     b:(256, 1)
Layer 2 w:(256, 256)    b:(256, 1)
Layer 3 w:(10, 256)     b:(10, 1)
Epoch   1/20    loss:0.32145    dev_acc:0.81350
Epoch   2/20    loss:0.21285    dev_acc:0.87480
Epoch   3/20    loss:0.17348    dev_acc:0.89660
Epoch   4/20    loss:0.16596    dev_acc:0.90950
Epoch   5/20    loss:0.13465    dev_acc:0.91640
Epoch   6/20    loss:0.12378    dev_acc:0.92290
Epoch   7/20    loss:0.13350    dev_acc:0.92530
Epoch   8/20    loss:0.09641    dev_acc:0.92960
Epoch   9/20    loss:0.11166    dev_acc:0.93210
Epoch  10/20    loss:0.11379    dev_acc:0.93330
Epoch  11/20    loss:0.09142    dev_acc:0.93640
Epoch  12/20    loss:0.11195    dev_acc:0.93690
Epoch  13/20    loss:0.08888    dev_acc:0.93760
Epoch  14/20    loss:0.09207    dev_acc:0.94000
Epoch  15/20    loss:0.09940    dev_acc:0.93940
Epoch  16/20    loss:0.08760    dev_acc:0.94200
Epoch  17/20    loss:0.10389    dev_acc:0.94160
Epoch  18/20    loss:0.10943    dev_acc:0.94270
Epoch  19/20    loss:0.08941    dev_acc:0.94380
Epoch  20/20    loss:0.09978    dev_acc:0.94480
activation:Relu
loss function:SquaredLoss
Layer 1 w:(256, 87)     b:(256, 1)
Layer 2 w:(256, 256)    b:(256, 1)
Layer 3 w:(10, 256)     b:(10, 1)
Epoch   1/20    loss:0.32317    dev_acc:0.78410
Epoch   2/20    loss:0.25379    dev_acc:0.85300
Epoch   3/20    loss:0.19423    dev_acc:0.87890
Epoch   4/20    loss:0.14025    dev_acc:0.89390
Epoch   5/20    loss:0.17181    dev_acc:0.90150
Epoch   6/20    loss:0.15374    dev_acc:0.90650
Epoch   7/20    loss:0.13893    dev_acc:0.91110
Epoch   8/20    loss:0.11867    dev_acc:0.91460
Epoch   9/20    loss:0.13896    dev_acc:0.91950
Epoch  10/20    loss:0.10716    dev_acc:0.92220
Epoch  11/20    loss:0.15091    dev_acc:0.92370
Epoch  12/20    loss:0.12702    dev_acc:0.92620
Epoch  13/20    loss:0.10267    dev_acc:0.92710
Epoch  14/20    loss:0.11721    dev_acc:0.92960
Epoch  15/20    loss:0.08857    dev_acc:0.93040
Epoch  16/20    loss:0.09396    dev_acc:0.93190
Epoch  17/20    loss:0.09741    dev_acc:0.93360
Epoch  18/20    loss:0.09721    dev_acc:0.93530
Epoch  19/20    loss:0.09332    dev_acc:0.93530
Epoch  20/20    loss:0.10726    dev_acc:0.93650
activation:Relu
loss function:SquaredLoss
Layer 1 w:(256, 331)    b:(256, 1)
Layer 2 w:(256, 256)    b:(256, 1)
Layer 3 w:(10, 256)     b:(10, 1)
Epoch   1/20    loss:0.29699    dev_acc:0.78120
Epoch   2/20    loss:0.23514    dev_acc:0.84640
Epoch   3/20    loss:0.20420    dev_acc:0.87660
Epoch   4/20    loss:0.16098    dev_acc:0.89210
Epoch   5/20    loss:0.14342    dev_acc:0.90270
Epoch   6/20    loss:0.13605    dev_acc:0.91130
Epoch   7/20    loss:0.17145    dev_acc:0.91540
Epoch   8/20    loss:0.11264    dev_acc:0.92070
Epoch   9/20    loss:0.14375    dev_acc:0.92280
Epoch  10/20    loss:0.11768    dev_acc:0.92550
Epoch  11/20    loss:0.14149    dev_acc:0.92810
Epoch  12/20    loss:0.11686    dev_acc:0.92940
Epoch  13/20    loss:0.10794    dev_acc:0.93140
Epoch  14/20    loss:0.10906    dev_acc:0.93240
Epoch  15/20    loss:0.09770    dev_acc:0.93410
Epoch  16/20    loss:0.10696    dev_acc:0.93580
Epoch  17/20    loss:0.09318    dev_acc:0.93630
Epoch  18/20    loss:0.08980    dev_acc:0.93710
Epoch  19/20    loss:0.12739    dev_acc:0.93790
Epoch  20/20    loss:0.09715    dev_acc:0.93930
```


NN Train Time vs Dimension
Time in Seconds
Dimension
14
16
18
20
22
24
0
50
100
150
200
250
300


NN Test Time vs Dimension
Time in Seconds
0.0525
0.0550
0.0575
0.0600
0.0625
0.0650
0.0675
Dimension
0
50
100
150
200
250
300


NN Accuracy on Test Set vs Dimension
Accuracy
0.5
0.6
0.7
0.8
0.9
Dimension
0
50
100
150
200
250
300


NN Accuracy on Test Set vs Proportion of Variance
Accuracy
0.5
0.6
0.7
0.8
0.9
Proportion of Variance
0.2
0.4
0.6
0.8
1.0