

# Machine Learning Homework 10

Hairui Yin

1.

(a) We construct a simple autoencoder with a linear encoder and decoder as an inherit from Autoencoder from torch.nn.module. The data is standized using StandardScaler in sklearn. The model is trained with learning rate of 0.01, optimizer as Adam, loss function as MSE. And seed is fixed. As the code shows below:

```
1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 from sklearn.preprocessing import StandardScaler
7
8 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
9 def seed_torch(seed):
10     torch.manual_seed(seed)
11     torch.cuda.manual_seed(seed)
12     torch.cuda.manual_seed_all(seed)
13     torch.backends.cudnn.deterministic = True
14     torch.backends.cudnn.benchmark = False
15 seed_torch(1)
16
17 class Autoencoder(nn.Module):
18     def __init__(self, input_dim, code_dim, activation=None):
19         super(Autoencoder, self).__init__()
20         self.encoder = nn.Linear(input_dim, code_dim, dtype=torch.float64)
21         self.decoder = nn.Linear(code_dim, input_dim, dtype=torch.float64)
22         self.activation = activation
23     def forward(self, x):
24         if self.activation:
25             encoded = self.activation(self.encoder(x))
26             decoded = self.decoder(encoded)
27         else:
28             encoded = self.encoder(x)
29             decoded = self.decoder(encoded)
30         return decoded
31
32 def train_autoencoder(model, data, epochs=1000, learning_rate=0.01):
```

```

33     criterion = nn.MSELoss()
34     optimizer = optim.Adam(model.parameters(), lr=learning_rate)
35     for epoch in range(epochs):
36         optimizer.zero_grad()
37         reconstructed = model(data)
38         loss = criterion(reconstructed, data)
39         loss.backward()
40         optimizer.step()
41     return model, loss.item()
42
43 # Main
44 data = pd.read_csv('Pizza.csv')
45 features = data.iloc[:, 2:9].values
46 scalar = StandardScaler()
47 features = scalar.fit_transform(features)
48 X = torch.tensor(features, dtype=torch.float64).to(device)
49 input_dims, hid_dims = X.shape[1], range(1, 7)
50 # Linear
51 linear_mse = []
52 for d in hid_dims:
53     model = Autoencoder(input_dims, d).to(device)
54     trained_model, mse = train_autoencoder(model, X)
55     pred_y = trained_model(X)
56     mse = torch.mean((X - pred_y) ** 2).item()
57     linear_mse.append(mse)
58
59 # Result
60 print(linear_acc)

```

MSE given by each hidden dimensions is

Table 1: MSE and hidden size

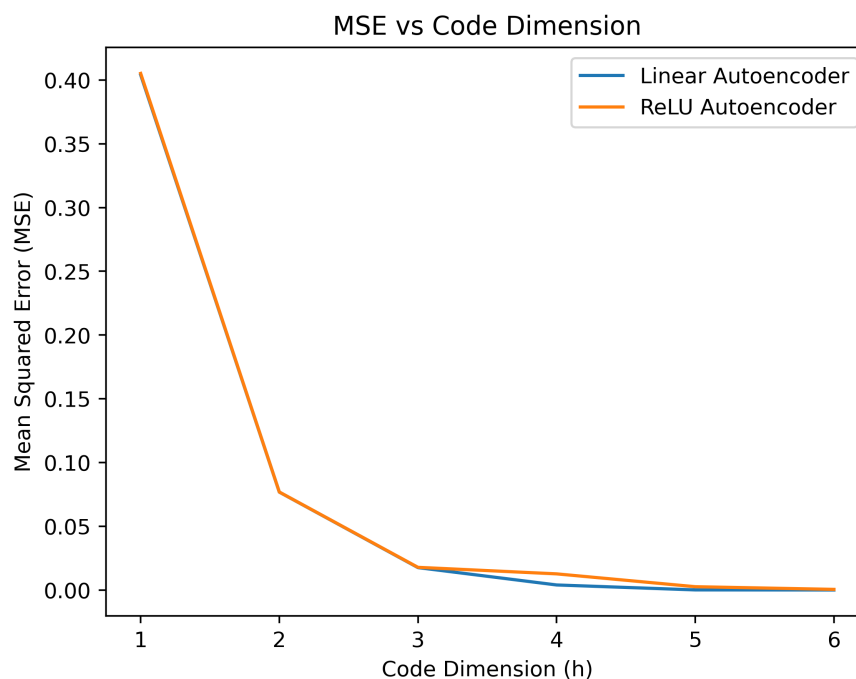
h	1	2	3	4	5	6
MSE	0.4040	0.0768	0.0176	0.0040	0.0001	0.00006

(b) We add ReLU activation function before the hidden layer, and retrain the model. The code follows is:

```
1 relu_mse = []
2 for d in hid_dims:
3     relu_model = Autoencoder(input_dims, d, activation=nn.ReLU()).to(device)
4     relu_model, mse = train_autoencoder(relu_model, X)
5     reconstructed = relu_model(X)
6     mse = torch.mean((X - reconstructed) ** 2).item()
7     relu_mse.append(mse)
8
9 # Result
10 print(relu_mse)
```

Then we plot the MSE as a function of h (both linear and non-linear):

```
1 plt.plot(hid_dims, linear_mse, label='Linear Autoencoder')
2 plt.plot(hid_dims, relu_mse, label='ReLU Autoencoder')
3 plt.xlabel('Code Dimension (h)')
4 plt.ylabel('Mean Squared Error (MSE)')
5 plt.title('MSE vs Code Dimension')
6 plt.legend()
7 plt.show()
```



## 2.

The optimal autoencoder in **q1** has hidden layer dimension  $h = 6$ . Let the encoder weight matrix be  $W$ , decoder weight matrix be  $V$ , encoder bias be  $W_b$  and decoder bias be  $V_b$ , we can get these item using code below:

```
1 W, V = trained_model.encoder.weight.data, trained_model.decoder.weight.data
2 Wb, Vb = trained_model.encoder.bias, trained_model.decoder.bias
3 W, V, Wb, Vb
```

And output is:

```
(tensor([[ -0.6267, -0.0089, -0.2987,  0.3788,  0.2406,  0.6258,  0.1638],
         [ -0.1310, -0.4929,  0.3337, -0.6449, -0.0094, -0.1735,  0.0929],
         [ -0.1689, -0.6133, -0.0054,  0.0263, -0.2782,  0.2428, -0.1298],
         [ -0.0031, -0.2417,  0.3165,  0.5515,  0.2711, -0.0676,  0.6554],
         [ -0.3615,  0.3481,  0.3281, -0.0364, -0.6882,  0.2393,  0.4586],
         [ -0.1191,  0.1888, -0.0485, -0.4245,  0.5237,  0.3295,  0.6537]]),
 device='cuda:0', dtype=torch.float64),
 tensor([[ -3.6613e-01,  1.9819e-01, -1.8601e-01, -1.8679e-04, -3.7844e-01,
          -4.2736e-01],
         [ -1.3161e-01, -5.6754e-01, -7.8360e-01, -2.7811e-01,  3.6852e-01,
           2.6310e-01],
         [ -3.5372e-01,  2.9869e-01, -1.9352e-01,  6.4410e-01,  1.9663e-01,
          -5.6802e-03],
         [  2.6191e-01, -4.7008e-01, -7.7669e-02,  6.3678e-01, -6.1590e-02,
          -5.8015e-01],
         [  1.3693e-01, -2.3506e-02, -3.4514e-01,  3.0875e-01, -7.3005e-01,
           5.4527e-01],
         [  5.6653e-01,  2.0216e-01,  2.6839e-01, -2.8181e-01, -3.8635e-02,
          -1.9250e-02],
         [ -7.8088e-02,  9.9342e-02,  1.4463e-02,  4.1786e-01,  3.8731e-01,
           3.3512e-01]]), device='cuda:0', dtype=torch.float64),
 Parameter containing:
 tensor([ 0.0041, -0.1504,  0.2661, -0.2501,  0.0954,  0.0828], device='cuda:0',
        dtype=torch.float64, requires_grad=True),
 Parameter containing:
 tensor([ 0.1522, -0.0028,  0.2407,  0.1621,  0.1895, -0.1086,  0.0513],
        device='cuda:0', dtype=torch.float64, requires_grad=True))
```

Consider shape of  $W$  is  $[6, 7]$ , shape of  $W_b$  is  $[6]$ , shape of  $V$  is  $[7, 6]$ , shape of  $V_b$  is  $[7]$ , and singular values of  $X$  has shape  $[7]$ . The MSE value given input  $x$  is calculated through:

$$MSE = ||V(Wx + W_b) + V_b - x||_F^2$$