FAI HW4

b08902045 資工四 袁紹奇

Hand-written Part

Problem 1

First we find the derivative of $\theta(s) = \frac{1}{1+e^{-s}}$.

$$\theta'(s) = \frac{0 \cdot (1+e^{-s}) - 1 \cdot (-e^{-s})}{(1+e^{-s})^2} = \theta'(s) = \frac{e^{-s}}{(1+e^{-s})^2}$$

Then we can find the derivative of Swish function.

$$\varphi(s) = s \cdot \theta(s) = \frac{s}{1+e^{-s}}$$

$$\varphi'(s) = s \cdot \theta'(s) + \theta(s) \cdot s' = \frac{s \cdot e^{-s}}{(1+e^{-s})^2} + \frac{s}{1+e^{-s}}$$

Problem 2

$$P = \begin{bmatrix} 0 & 1 & 0.5 \\ 0 & 0 & 0.5 \\ 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{v}_0 = \left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]^T$$

(A)

After operating the iteration method, we have

$$\mathbf{v}_1 = \left[\frac{1}{2}, \frac{1}{6}, \frac{1}{3}\right]^T$$

$$\mathbf{v}_2 = \left[\frac{1}{3}, \frac{1}{6}, \frac{1}{2}\right]^T$$

$$\mathbf{v}_3 = \left[\frac{5}{12}, \frac{1}{4}, \frac{1}{3}\right]^T$$

$$\mathbf{v}_4 = \left[\frac{5}{12}, \frac{1}{6}, \frac{5}{12}\right]^T$$

$$\mathbf{v}_5 = \left[\frac{9}{24}, \frac{5}{24}, \frac{5}{12}\right]^T$$

(B)

The equation of $P\mathbf{v}^* = \mathbf{v}^*$ is equivalent to finding the eigenvector with eigen value 1.

$$(P - I)\mathbf{v}^* = \mathbf{0}$$
$$\begin{bmatrix} -1 & 1 & 0.5 \\ 0 & -1 & 0.5 \\ 1 & 0 & -1 \end{bmatrix} \mathbf{v}^* = \mathbf{0}$$

Use Gaussian elimination, we have

$$\begin{bmatrix} -1 & 1 & 0.5 \\ 0 & -1 & 0.5 \\ 0 & 0 & 0 \end{bmatrix}$$

Then we found the corresponding normalized vector $\mathbf{v}^* = \left[\frac{2}{5}, \frac{1}{5}, \frac{2}{5}\right]^T$.

Problem 3

If $L = 1$, then $d^{(1)} = 100$, and the total number of weights is $10 \cdot 100 = 1000$.

If $L = 2$, let $d^{(1)} = x_1$ and $d^{(2)} = 100 - x_1$. The total number of weights is $10x_1 + x_1 \cdot (100 - x_1) = -x_1^2 + 110x_1$, where $x_1 > 0$ and $100 - x_1 > 0$. The maximum of 3025 occurs when $x_1 = 55$, and the minimum of 109 occurs when $x_1 = 1$.

If $L = 3$, let $d^{(1)} = x_1$, $d^{(2)} = x_2$, and $d^{(3)} = 100 - x_1 - x_2$. The total number of weights is $10x_1 + x_1x_2 + x_2(100 - x_1 - x_2) = 10x_1 - x_2^2 + 100x_2$, where $x_1, x_2 > 0$, and $100 - x_1 - x_2 > 0$. Since $x_1 + x_2 < 100$, and we can determine x_1 once we select x_2 when finding extremes, we can rewrite the terms respectively when finding the maximum and minimum.

When finding the maximum, we can substitute x_1 with $100 - 1 - x_2$, and the total number of weights becomes $990 + 90x_2 - x_2^2$. Therefore, the maximum, $990 + 45 \cdot 90 - 45^2 = 3015$, occurs when $x_1 = 55$ and $x_2 = 45$.

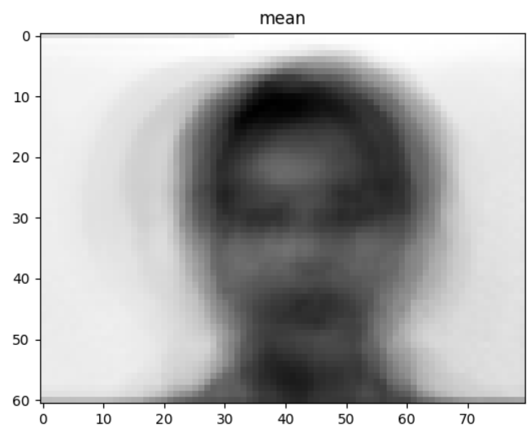
Similarly, when finding the minimum, we can substitute x_1 with 1. The minimum, $10 + 100 - 1 = 109$, occurs when $x_1 = 1$ and $x_2 = 1$.

In conclusion, the maximum number of weights is 3025 when $L = 2$, $d^{(1)} = 55$, and $d^{(2)} = 45$. The minimum number of weights is 109 when $L = 2$, $d^{(1)} = 1$, $d^{(2)} = 99$, or when $L = 3$, $d^{(1)} = 1$, $d^{(2)} = 1$, $d^{(3)} = 98$.

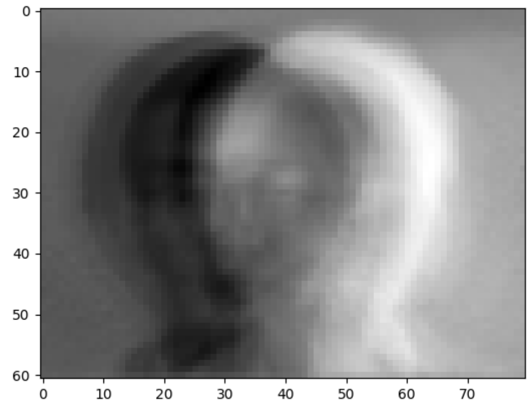
Programming Part

(a)

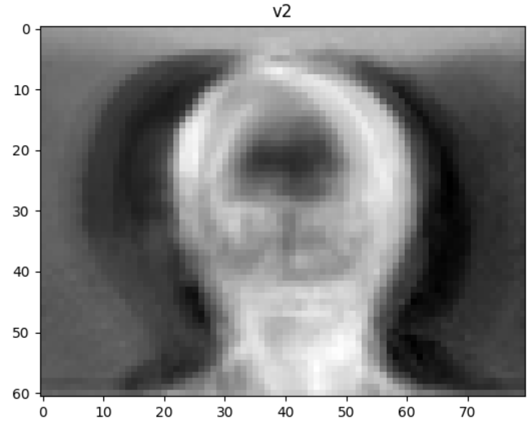
The following are the mean and top 4 eigenvectors, where v1 represents the vector with the largest eigenvalue.



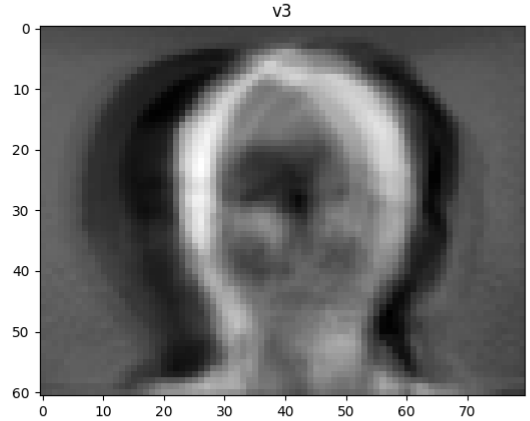
v1



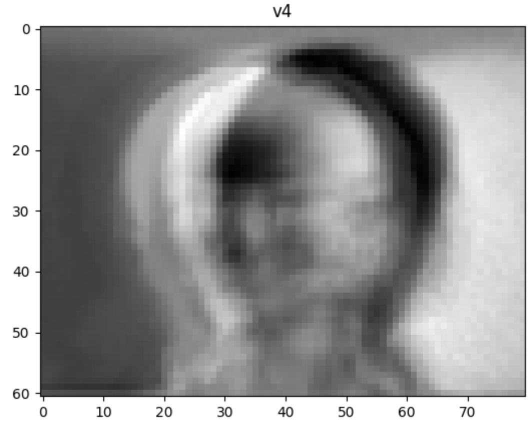
v2



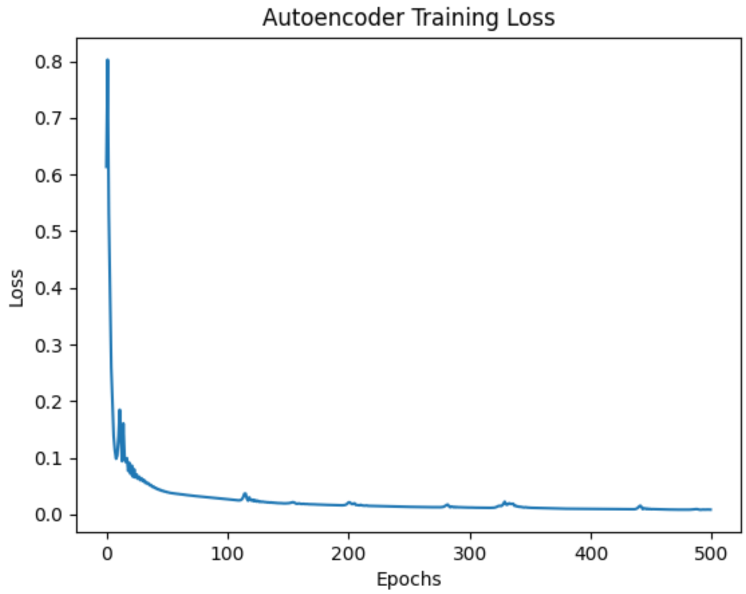
v3

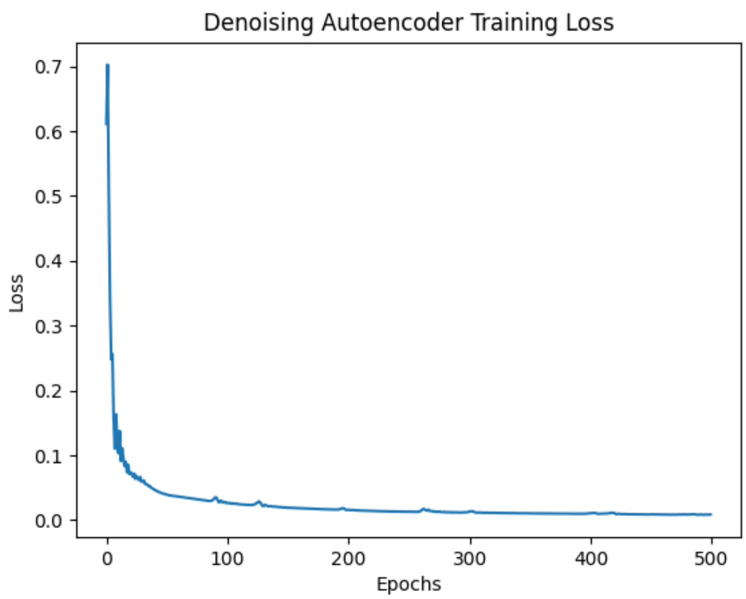


v4

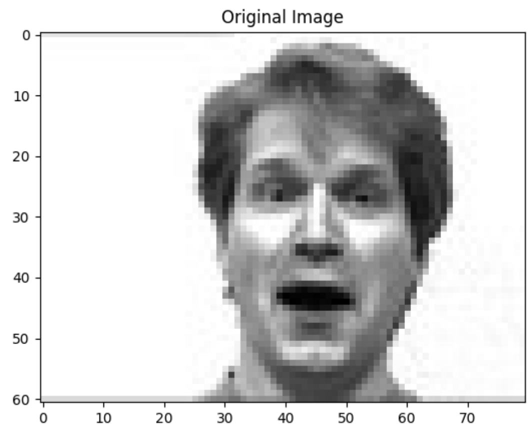


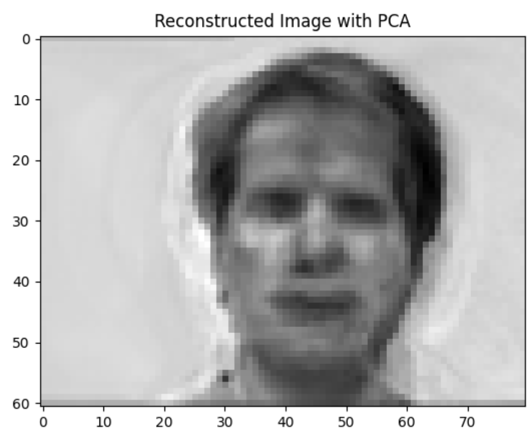
(b)

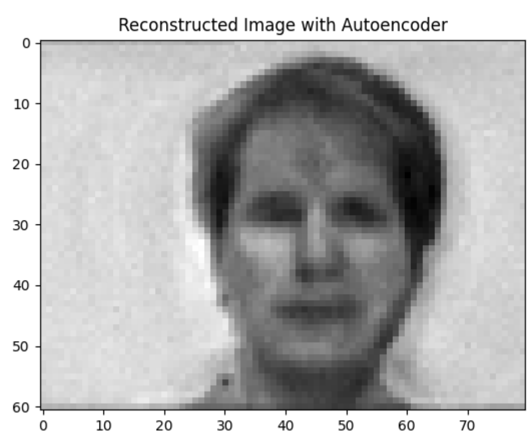

Autoencoder Training Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Loss
0
100
200
300
400
500
Epochs

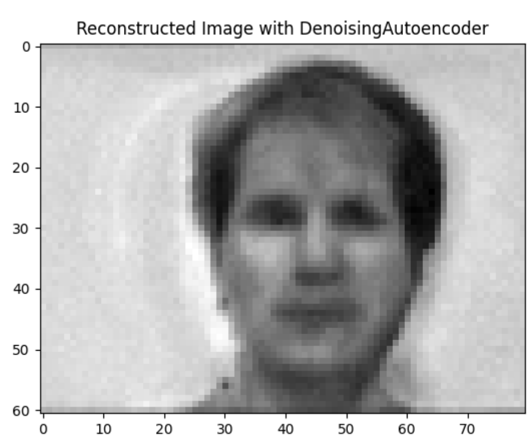

Denoising Autoencoder Training Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Loss
0
100
200
300
400
500
Epochs

(c)


Original Image
0
10
20
30
40
50
60
0
10
20
30
40
50
60
70


Reconstructed Image with PCA
0
10
20
30
40
50
60
0
10
20
30
40
50
60
70


Reconstructed Image with Autoencoder
0
10
20
30
40
50
60
0
10
20
30
40
50
60
70


Reconstructed Image with DenoisingAutoencoder
0
10
20
30
40
50
60
0
10
20
30
40
50
60
70

Reconstruction Loss with PCA: 0.010710469688056319

Reconstruction Loss with Autoencoder: 0.012685404983394908

Reconstruction Loss with DenoisingAutoencoder: 0.013614476498283883

(d)

Original Network

```
self.encoder = nn.Sequential(
    nn.Linear(input_dim, encoding_dim),
    nn.Linear(encoding_dim, encoding_dim//2),
    nn.ReLU()
)
self.decoder = nn.Sequential(
    nn.Linear(encoding_dim//2, encoding_dim),
    nn.Linear(encoding_dim, input_dim),
)
```

```
Acc from Autoencoder: 0.9333333333333333
Acc from DenoisingAutoencoder: 0.9333333333333333
Reconstruction Loss with Autoencoder: 0.012685404983394908
Reconstruction Loss with DenoisingAutoencoder: 0.013614476498283883
```

Deeper Network

```
self.encoder = nn.Sequential(
    nn.Linear(input_dim, encoding_dim),
    nn.Linear(encoding_dim, encoding_dim//2),
    nn.Linear(encoding_dim//2, encoding_dim//2),
    nn.Linear(encoding_dim//2, encoding_dim//2),
    nn.ReLU()
)
self.decoder = nn.Sequential(
    nn.Linear(encoding_dim//2, encoding_dim//2),
    nn.Linear(encoding_dim//2, encoding_dim//2),
    nn.Linear(encoding_dim//2, encoding_dim),
    nn.Linear(encoding_dim, input_dim),
)
```

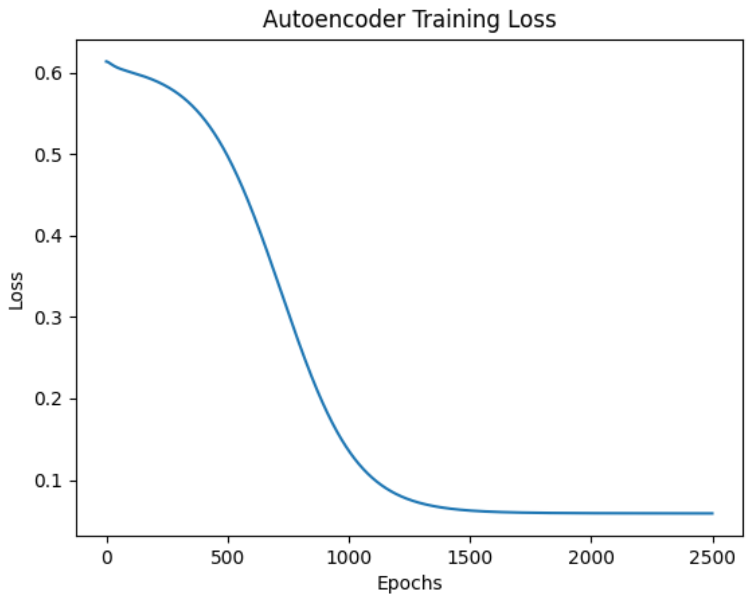
```
Acc from Autoencoder: 0.8666666666666667
Acc from DenoisingAutoencoder: 0.9
Reconstruction Loss with Autoencoder: 0.01474462375899693
Reconstruction Loss with DenoisingAutoencoder: 0.014040365749072092
```

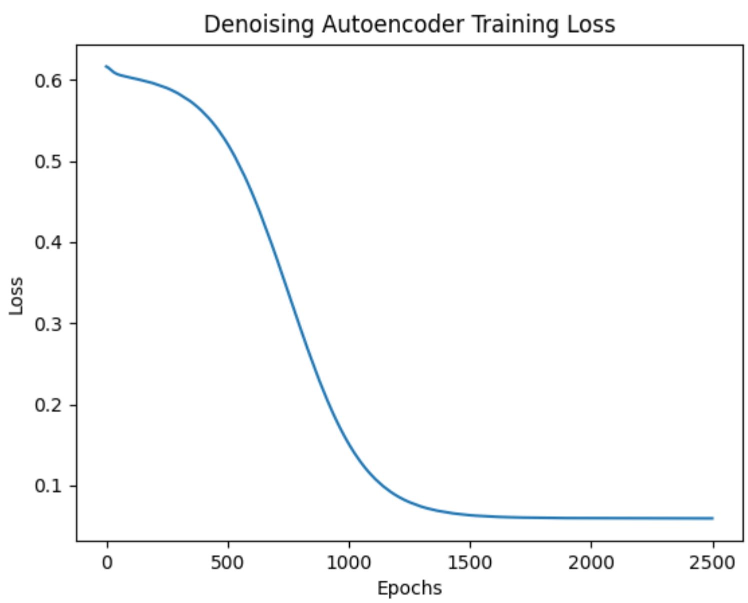
I tried a deeper network in both encoder and decoder. Although I can see that the loss had already converged, the performance is still worse than the original model. The reconstruction error is also slightly larger than the original. I think that it is because the tasks is rather simple, and it does not need a model with a deeper network.

(e)

SGD with momentum

```
optim.SGD(self.parameters(), lr=0.001, momentum=0.9)
```

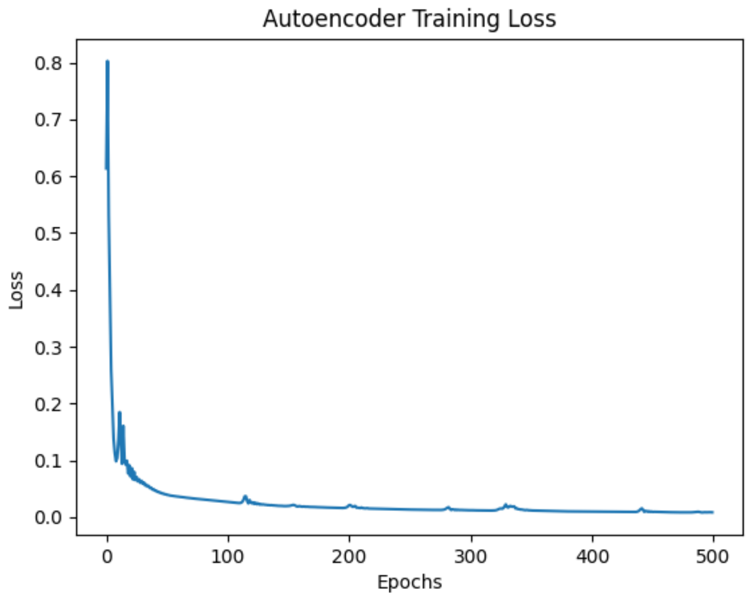


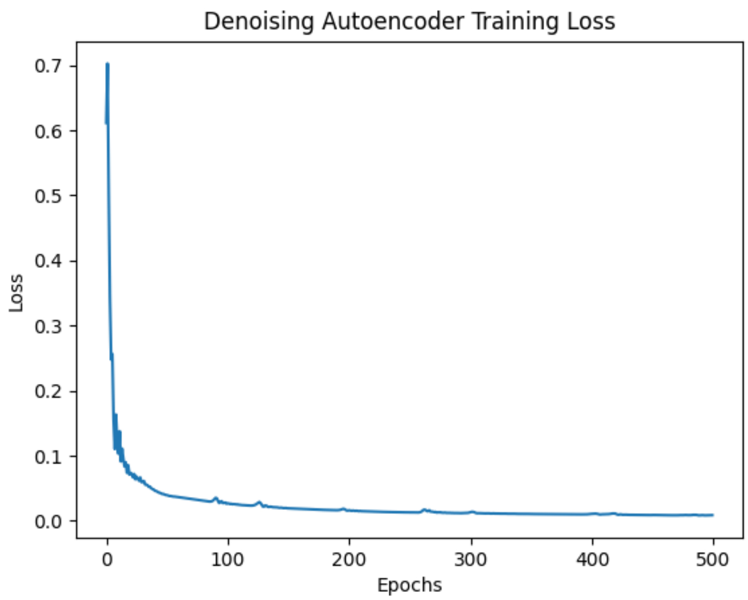



```
Acc from Autoencoder: 0.9
Acc from DenoisingAutoencoder: 0.9
Reconstruction Loss with Autoencoder: 0.035350335186838236
Reconstruction Loss with DenoisingAutoencoder: 0.03539134926334942
```

Adam

```
optim.Adam(self.parameters(), lr=0.001)
```






```
Acc from Autoencoder: 0.9333333333333333
Acc from DenoisingAutoencoder: 0.9333333333333333
Reconstruction Loss with Autoencoder: 0.012685404983394908
Reconstruction Loss with DenoisingAutoencoder: 0.013614476498283883
```

I used adam and SGD with momentum as the optimizers with the above configuration. We can see that SGD converges significantly slower than Adam. SGD needs roughly 1500 epochs to converge, while Adam only needs about 100 epochs. The performance of SGD is also slightly worse than Adam, and the reconstruction error is much higher than Adam.

Ref:
https://ithelp.ithome.com.tw/articles/10270394
https://chat.openai.com/