FAI HW4

B10902024 林宸宇

May 21, 2024

1 Hand-written Part

1.1 Problem 1

$$\begin{aligned}\varphi(s) &= \frac{s}{1+e^{-s}} \\ \varphi'(s) &= \frac{1+e^{-s}-s(-e^{-s})}{(1+e^{-s})^2} \\ &= \frac{1+e^{-s}+se^{-s}}{(1+e^{-s})^2}\end{aligned} \tag{1}$$

1.2 Problem 2

1.2.1 (A)

$v_0 = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]^T$, we can calculate v_1 to v_5 using the equation $v_t = Pv_{t-1}$.

$$\begin{aligned}v_1 &= [\frac{1}{2}, \frac{1}{6}, \frac{1}{3}]^T \\ v_2 &= [\frac{1}{3}, \frac{1}{6}, \frac{1}{2}]^T \\ v_3 &= [\frac{5}{12}, \frac{1}{4}, \frac{1}{3}]^T \\ v_4 &= [\frac{5}{12}, \frac{1}{6}, \frac{5}{12}]^T \\ v_5 &= [\frac{3}{8}, \frac{5}{24}, \frac{5}{12}]^T\end{aligned} \tag{2}$$

1.2.2 (B)

We can set $v^* = [x, y, z]^T$. Using the equation $v^* = Pv^*$ and $\sum v^* = 1$, we can derive the following equations,

$$\begin{cases} y + 0.5z = x \\ 0.5z = y \\ x = z \\ x + y + z = 1 \end{cases} \tag{3}$$

By solving it, we get $v^* = [0.4, 0.2, 0.4]^T$.

1.3 Problem 3

1.3.1 (A)

The result is demonstrated in Tab.1. The **partition** operation will reassign points based on current centroids, and the **update centroid** operation will update the centroids based on current partition. The return data of **partition** will be a set of points while the **update centroid** will be a point.

Operation	1st cluster	2nd cluster
partition	$\{(1,2)\}$	$\{(3,4),(7,0),(10,2)\}$
update centroid	$(1,2)$	$(\frac{20}{3},2)$
partition	$\{(1,2),(3,4)\}$	$\{(7,0),(10,2)\}$
update centroid	$(2,3)$	$(\frac{17}{2},1)$

Table 1: Result after performing K-means with initial centroids $\{\mu_1,\mu_2\} = \{(1,2),(3,4)\}$

1.3.2 (B)

The result is demonstrated in Tab.2. The process is different but the result converges to the same as (A).

Operation	1st cluster	2nd cluster
partition	$\{(1,2),(3,4)\}$	$\{(7,0),(10,2)\}$
update centroid	$(2,3)$	$(\frac{17}{2},1)$

Table 2: Result after performing K-means with initial centroids $\{\mu_1,\mu_2\} = \{(1,2),(7,0)\}$

1.3.3 (C)

If we set the initial centroids as $\{\mu_1,\mu_2\} = \{(\frac{11}{3},2),(\frac{15}{2},4)\}$, we will converges to a local minimum. The iterated process is shown in Tab.3. To achieve global minimum, we should converges to $\{(3,4),(\frac{17}{2},1)\}$, where the $E_{in} = \frac{1}{5}(4+0+4+\frac{13}{4}+\frac{13}{4})$ is smaller than the $E_{in} = \frac{1}{5}(\frac{64}{9}+\frac{40}{9}+\frac{136}{9}+\frac{41}{4}+\frac{41}{4})$ in our case.

Operation	1st cluster	2nd cluster
partition	$\{(1,2),(3,4),(7,0)\}$	$\{(5,6),(10,2)\}$
update centroid	$(\frac{11}{3},2)$	$(\frac{15}{2},4)$

Table 3: Result after performing K-means with initial centroids $\{\mu_1,\mu_2\} = \{(\frac{11}{3},2),(\frac{15}{2},4)\}$

2 Programming Part

2.1 (a)

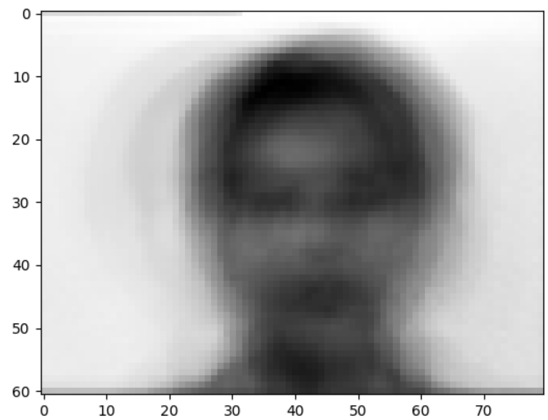


Figure 1: Mean vector

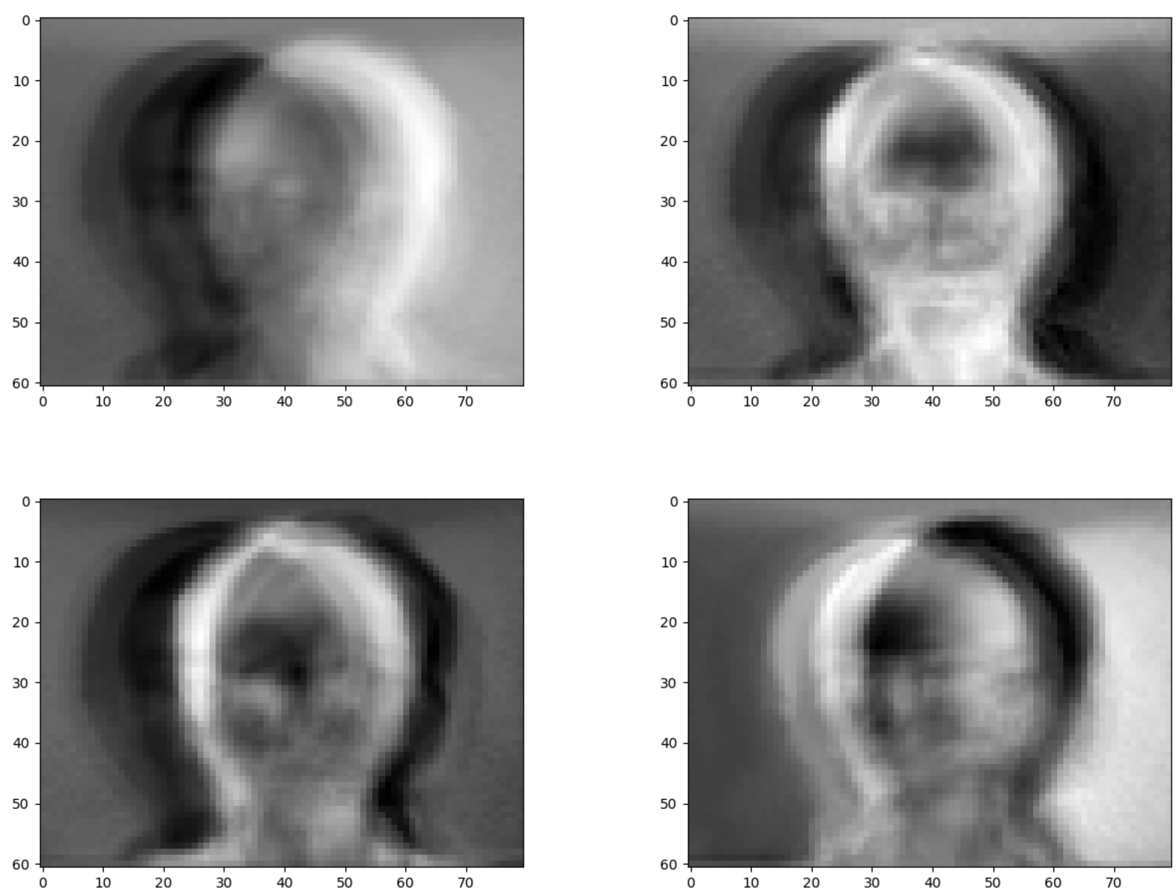
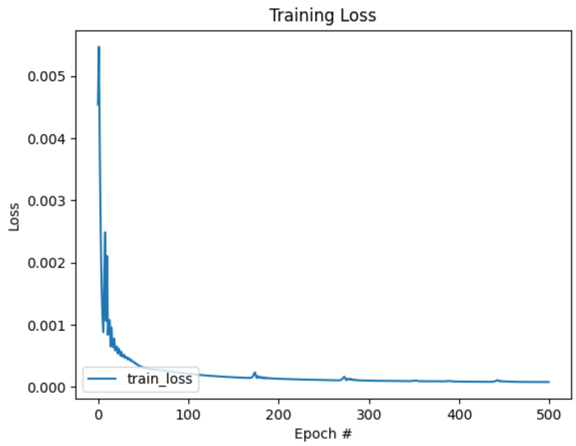
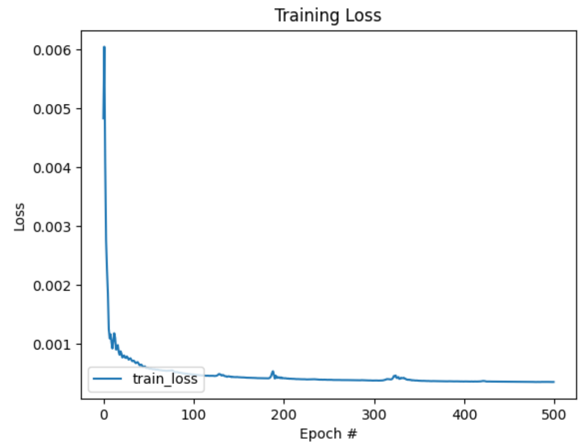


Figure 2: Top 4 eigenvectors

2.2 (b)



(a) Autoencoder



(b) DenoisingAutoencoder

Figure 3: Training curves of Autoencoder and DenoisingAutoencoder

2.3 (c)

The figures and the MSE between the original image and each reconstructed image are shown below.

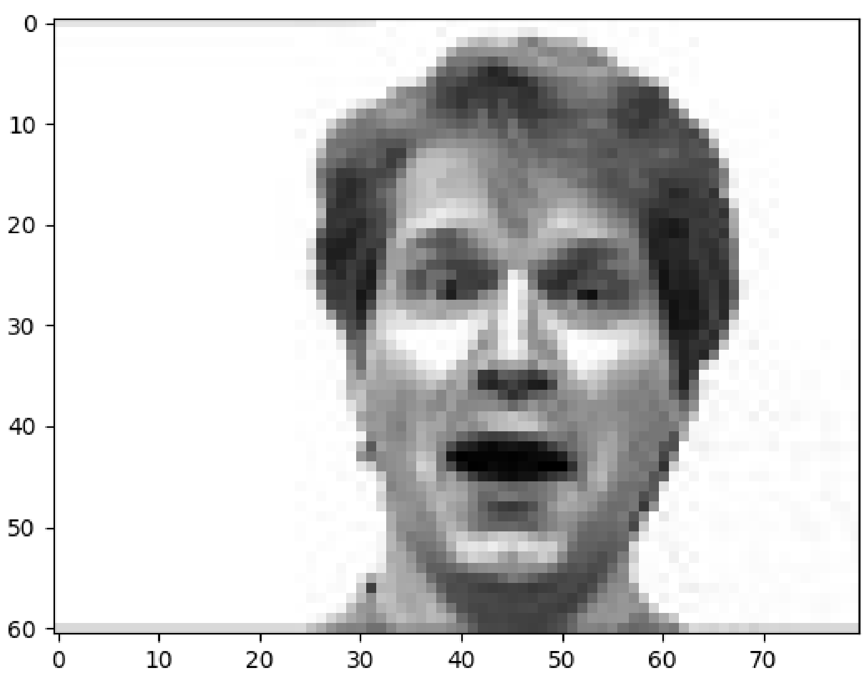


Figure 4: Original image

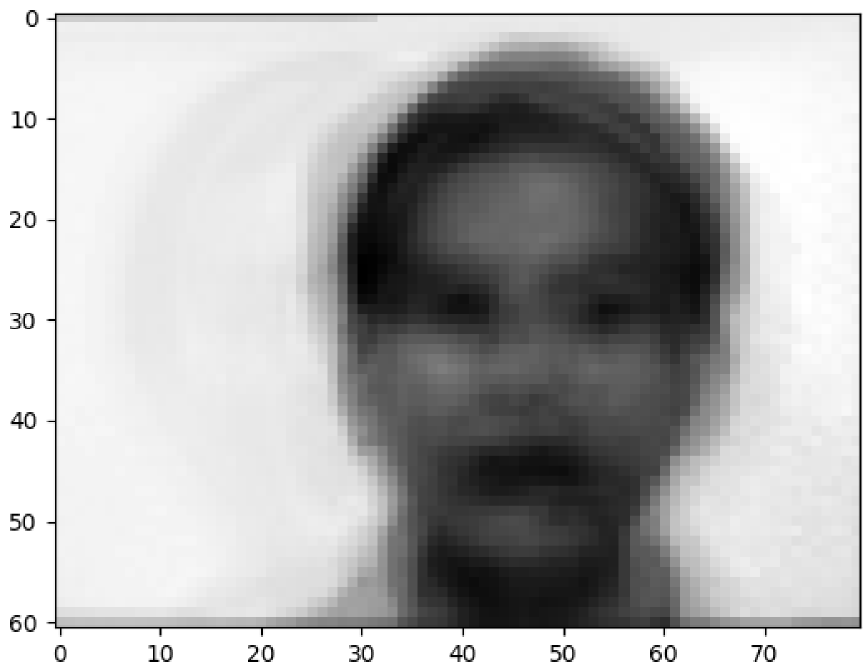


Figure 5: Reconstructed with PCA, MSE=0.01071

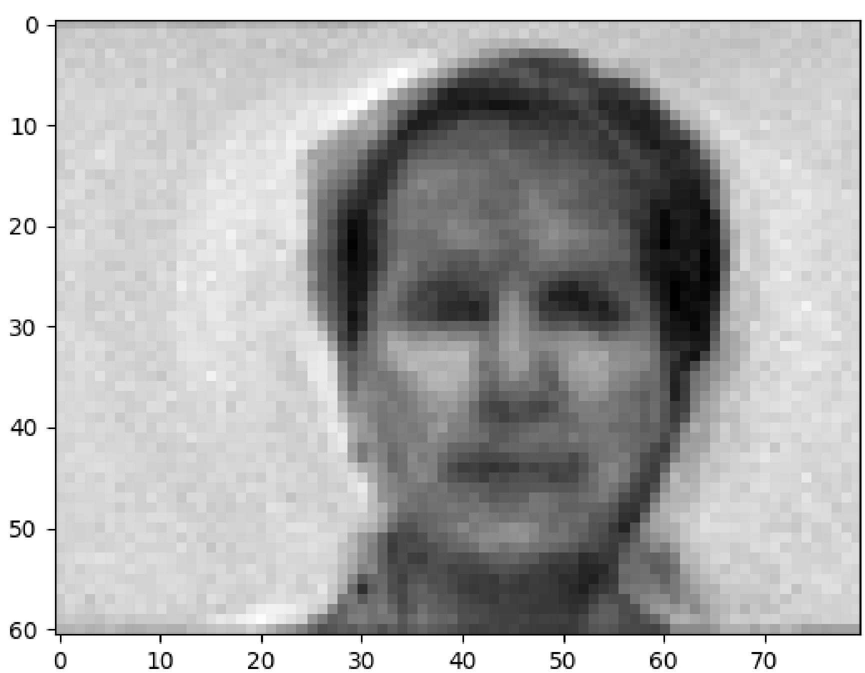


Figure 6: Reconstructed with Autoencoder, MSE=0.01477

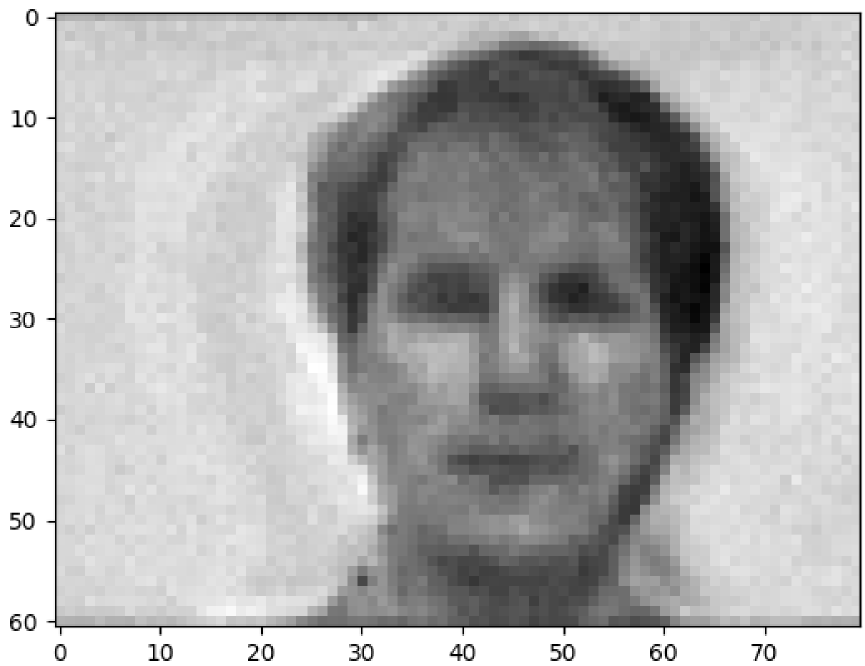


Figure 7: Reconstructed with DenoisingAutoencoder, MSE=0.01375

2.4 (d)

I've tried two different architectures, including a deeper one and a shallower one. The units in the diagram are not correct; they are just for illustration purposes.

2.4.1

For the deeper network, the architecture is shown below.

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

The accuracy and reconstruction loss is illustrated in Tab.4.

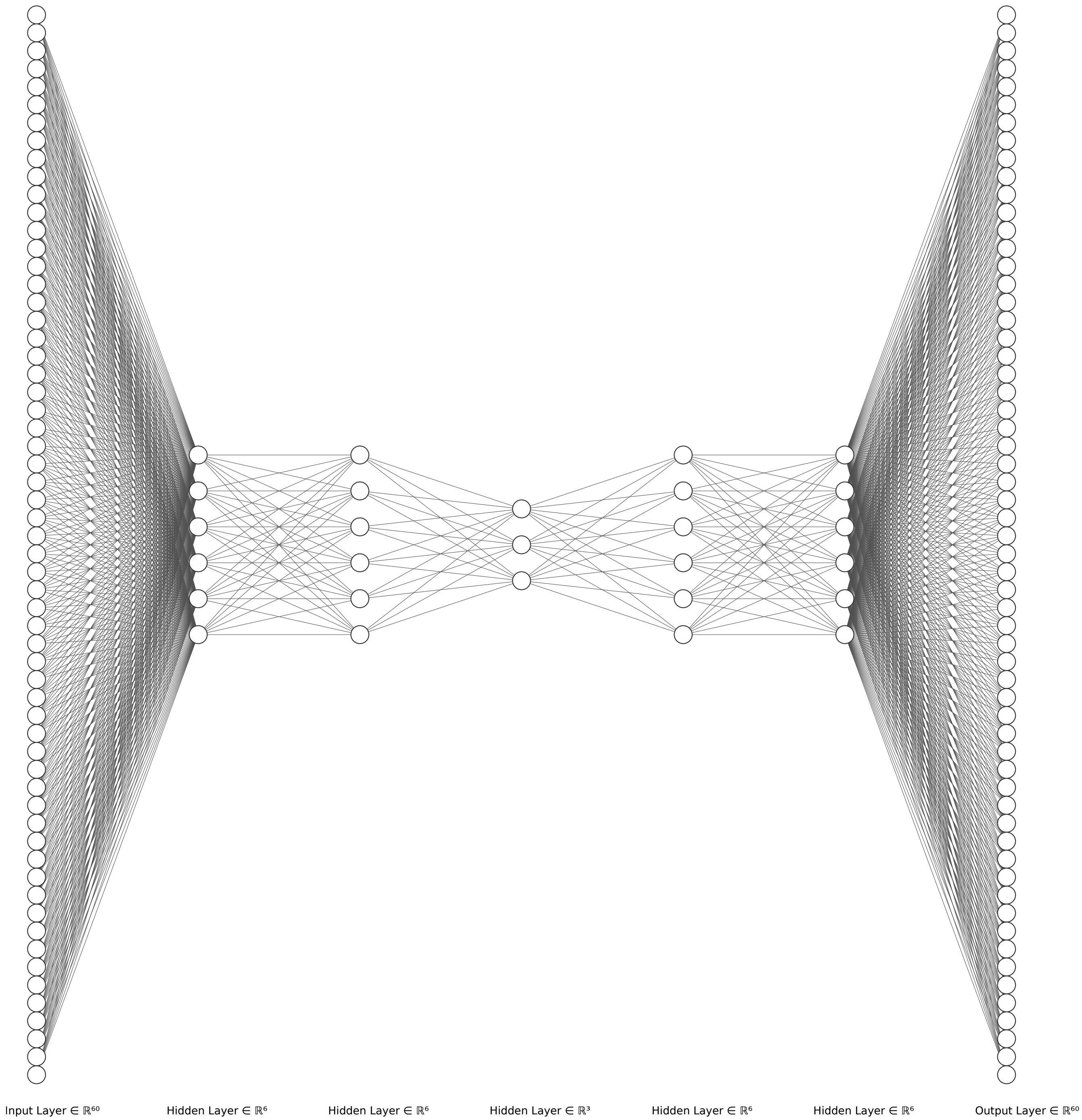


Figure 8: Deeper network

Method	Accuracy	Reconstruction Loss
Autoencoder	0.833	0.02538
DenoisingAutoencoder	0.833	0.02158

Table 4: Overall performance in deeper network

2.4.2

For the shallower network, the architecture is shown below.

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

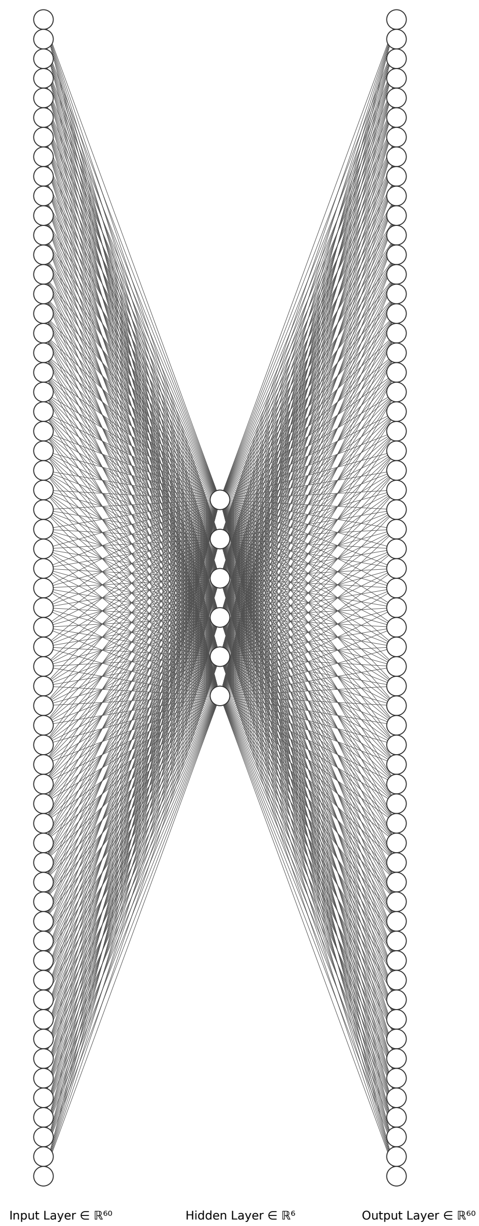


Figure 9: Shallower network

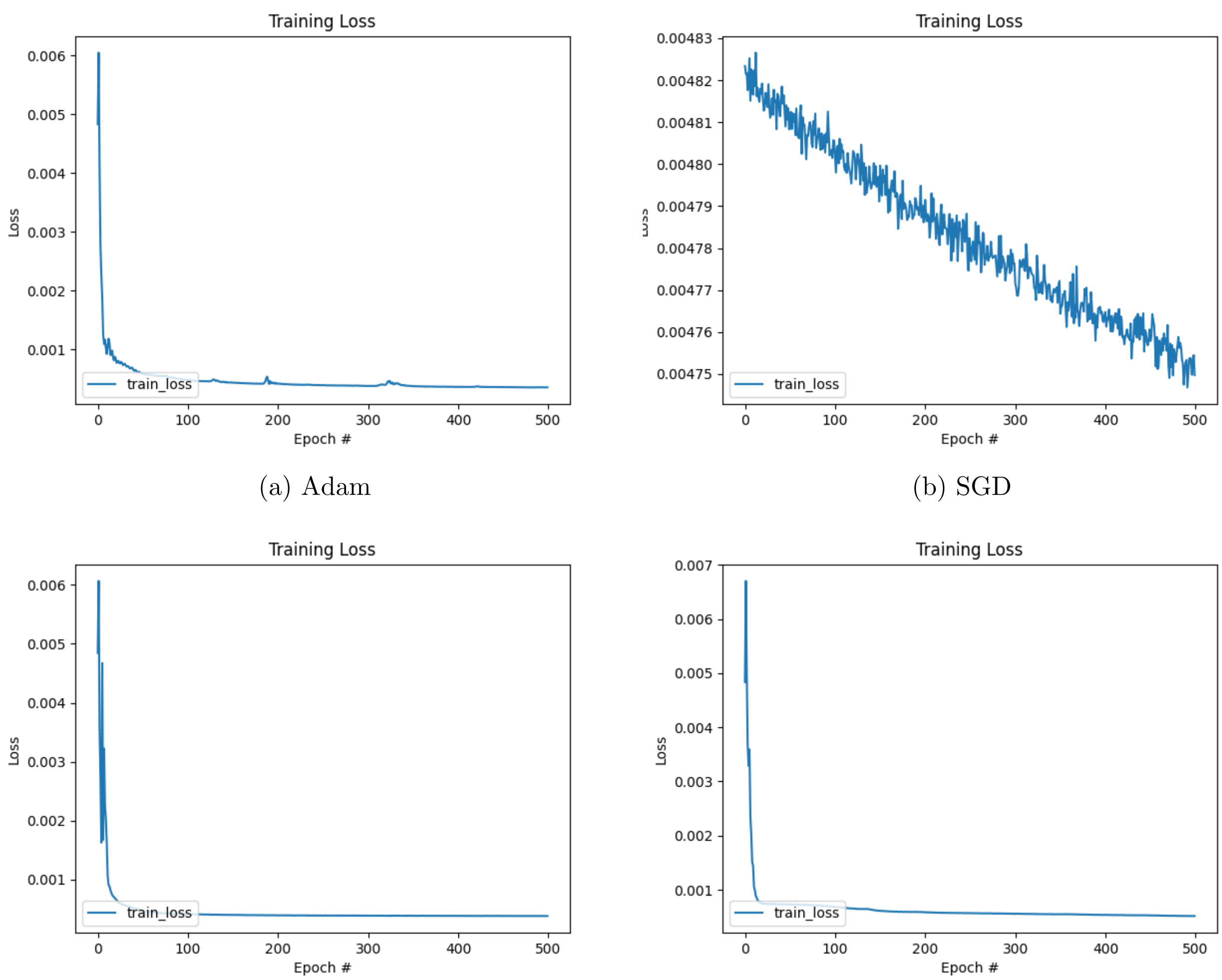
The accuracy and reconstruction loss is illustrated in Tab.5.

Method	Accuracy	Reconstruction Loss
Autoencoder	0.867	0.02177
DenoisingAutoencoder	0.867	0.01944

Table 5: Overall performance in shallower network

For both deeper and shallower networks, their performance are worse than the original network. For a shallow network, it might be not powerful enough to model all the cases and leads to underfitting. On the other hand, the size of the given dataset is too small for a large model and will lead to overfitting as well. Therefore, choosing a modest size and complexity of network is of paramount importance.

2.5 (e)



(a) Adam

(b) SGD

(c) Rprop

(d) Adagrad

Figure 10: Overall performance of different optimizers

In terms of convergence speed and training loss, Adam, Rprop and Adagrad have similar performance, while SGD performs much worse. It has higher training loss and didn't converge well. What surprises me is even though SGD seems to have poor performance, and it gets a high reconstruction loss as well, but it achieves 0.933 accuracy, which ranks the top among all optimizers. Although Rprop and Adagrad have low reconstruction loss and fast convergence speed, their accuracy is lower than Adam. Therefore,

Optimizer	Accuracy	Reconstruction Loss
Adam	0.933	0.01776
SGD	0.933	0.69116
Rprop	0.867	0.01612
Adagrad	0.800	0.02685

Table 6: Overall performance using different optimizer

taking all factors into consideration, I'll choose Adam as my optimizer in this case.