FAI HW4

1

Swish: $\varphi(s) = s * \theta(s)$
$\varphi'(s) = \theta(s) + s * \theta'(s)$
$= \frac{1}{1+exp(-s)} + s * (\frac{1}{1+exp(-s)} * (1 - \frac{1}{1+exp(-s)}))$
$= \theta(s) + s\theta(s) - s\theta^2(s)$
$= -s\theta^2(s) + (1+s)\theta(s)$

2

cost matrix:
$\begin{bmatrix} 0 & 1 & 10 & 100 \\ 200 & 0 & 2 & 20 \\ 30 & 300 & 0 & 3 \\ 4 & 40 & 400 & 0 \end{bmatrix}$
$E_{in}(g): 5$
$\begin{bmatrix} 0 & 1 & 10 & 100 \\ 200 & 0 & 2 & 20 \\ 30 & 300 & 0 & 3 \\ 4 & 40 & 400 & 0 \end{bmatrix}$
Let result be:
$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix}$, $a_1 + b_2 + c_3 + d_4 = 0.95$
maximum: $0.05 * 400 = 20$
minimum: $0.05 * 1 = 0.05$

3

$\sum_{l=1}^{L} d^l = 100$
maximum weight: $10 * 55 + 55 * 45 = 3025 \rightarrow$ one layer NN
minimum weight: $10 + 1 + 1 + ... + 1 = 109 \rightarrow$ n layer NN

4

$u(x) = 1 - 2|\theta_w(x) - \frac{1}{2}|$

$u(x) = \begin{cases} 2 - 2\theta_w(x), \text{ if } w^T x \leq 0 \\ 2\theta_w(x), \text{ if } w^T x > 0 \end{cases}$

$\theta_w(x)$ is monotonic increasing $\rightarrow u(x)$ is monotonic decreasing if $w^T x \leq 0$
, else increasing

if

$w^T x \leq 0, \text{argmax } u(x_n) = \text{argmin } \theta_w(x_n)$

$= \text{argmax } exp(-w^T x_n)$

$= \text{argmax } - w^T x_n$

$= \text{argmax } - |w^T x_n| = argmin|w^T x_n|$

else,

$\text{argmax } u|x_n| = \text{argmax } \theta_w(x_n)$

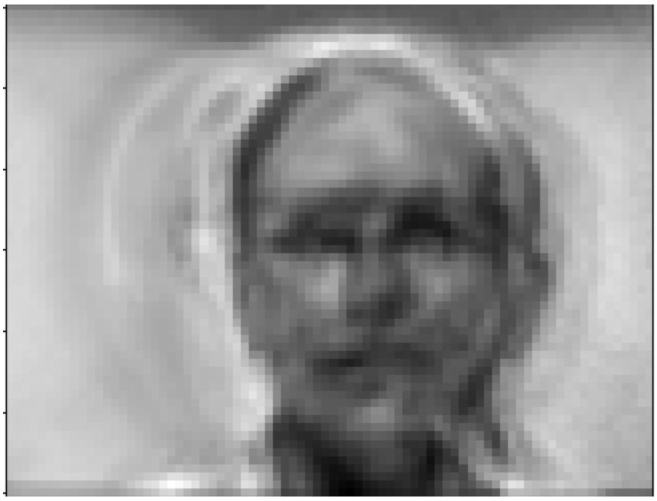
$= \text{argmin } exp(-w^T x_n) = \text{argmin } - w^T x_n$

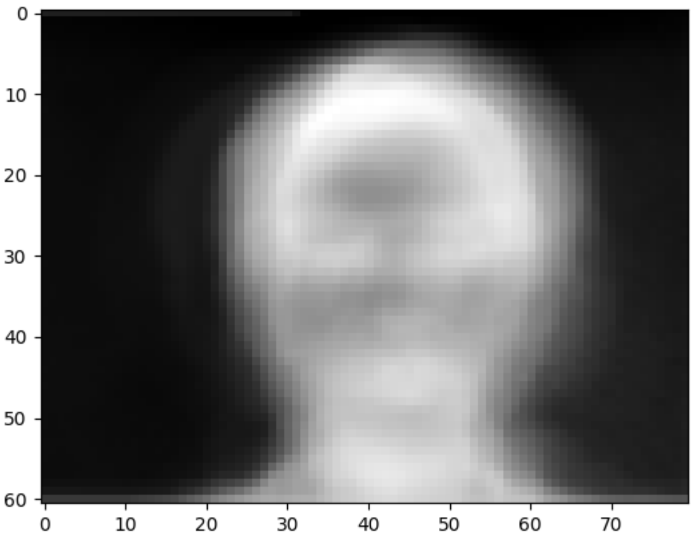
$= \text{argmax } - w^T x_n$

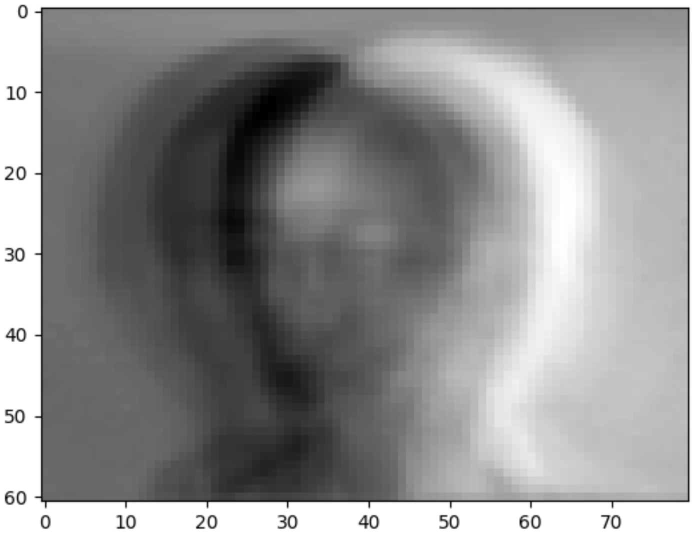
$= \text{argmin } w^T x_n$

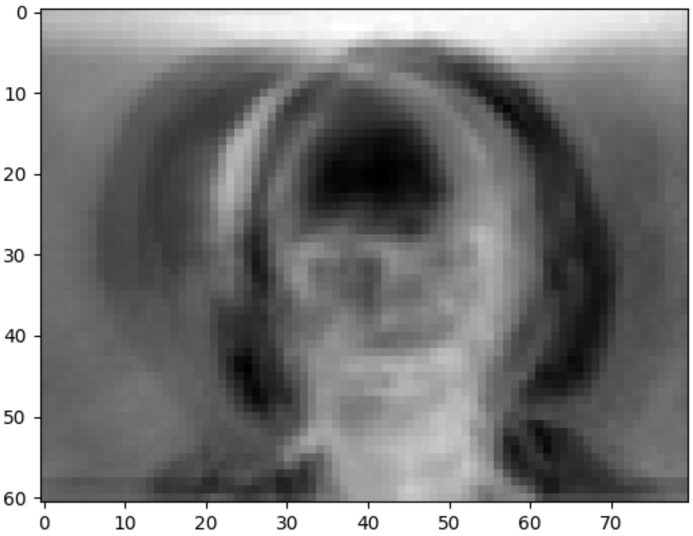
$= argmin|w^T x_n|$

5

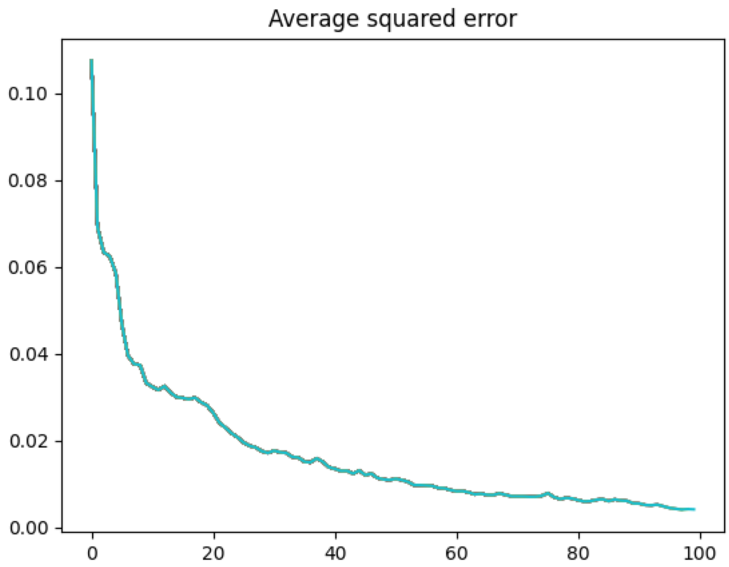



0
10
20
30
40
50
60
0
10
20
30
40
50
60
70

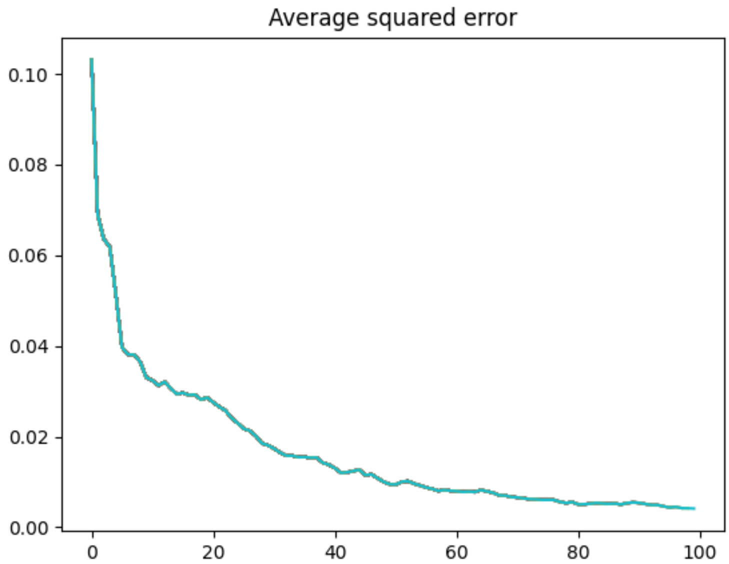

0
10
20
30
40
50
60
0
10
20
30
40
50
60
70


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

6


Average squared error
0.10
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80
100

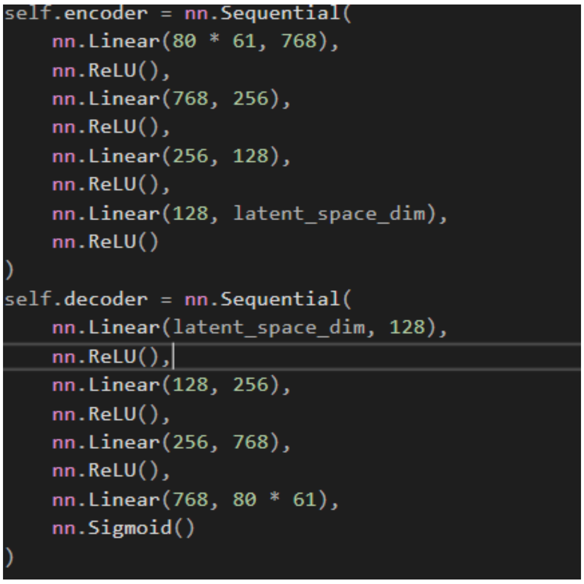
7


Average squared error
0.10
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80
100

8

Try 1:

model structure: add one more layer in between



```
self.encoder = nn.Sequential(
    nn.Linear(80 * 61, 768),
    nn.ReLU(),
    nn.Linear(768, 256),
    nn.ReLU(),
    nn.Linear(256, 128),
    nn.ReLU(),
    nn.Linear(128, latent_space_dim),
    nn.ReLU()
)
self.decoder = nn.Sequential(
    nn.Linear(latent_space_dim, 128),
    nn.ReLU(),
    nn.Linear(128, 256),
    nn.ReLU(),
    nn.Linear(256, 768),
    nn.ReLU(),
    nn.Linear(768, 80 * 61),
    nn.Sigmoid()
)
```

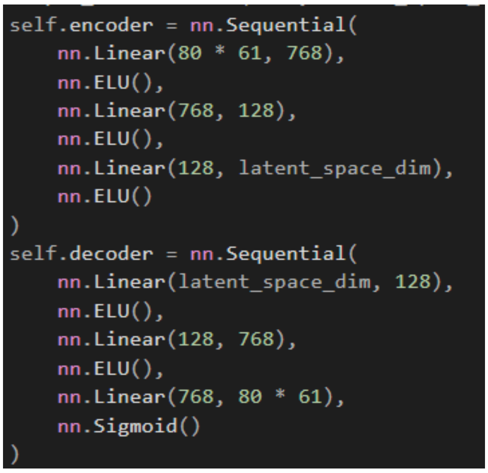
autoencoder mse: 0.0016, acc: 0.8

denoising autoencoder mse: 0.0071, acc: 0.7333

Possible reason: overfitting (cause I do not use dropout here)

Try 2:

model structure: use `ELU()` instead of `Relu()` . ELu() is said to have better performance



```
self.encoder = nn.Sequential(
    nn.Linear(80 * 61, 768),
    nn.ELU(),
    nn.Linear(768, 128),
    nn.ELU(),
    nn.Linear(128, latent_space_dim),
    nn.ELU()
)
self.decoder = nn.Sequential(
    nn.Linear(latent_space_dim, 128),
    nn.ELU(),
    nn.Linear(128, 768),
    nn.ELU(),
    nn.Linear(768, 80 * 61),
    nn.Sigmoid()
)
```

autoencoder mse: 0.0042, acc: 0.83334

denoising autoencoder mse: 0.0128, acc: 0.83334

Unfortunately, the performance is about the same as using Relu, Elu only performs better in theory

Try 3:

model structure: batchsize to 32

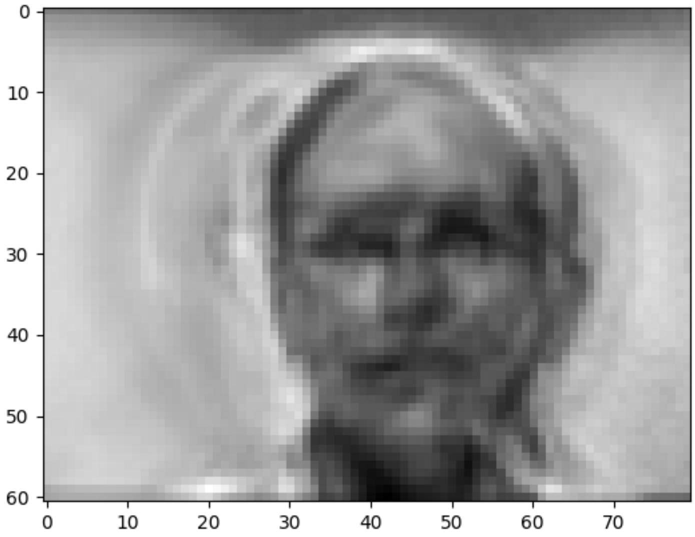
autoencoder mse: 0.0018, acc: 0.86667

denoising autoencoder mse: 0.0119, acc: 0.83334

slightly better, changing batch size is a way to improve performance. (However, using bigger or smaller batch size to have better performance requires testing, there is no certain evidence that shows one is better that the other.)

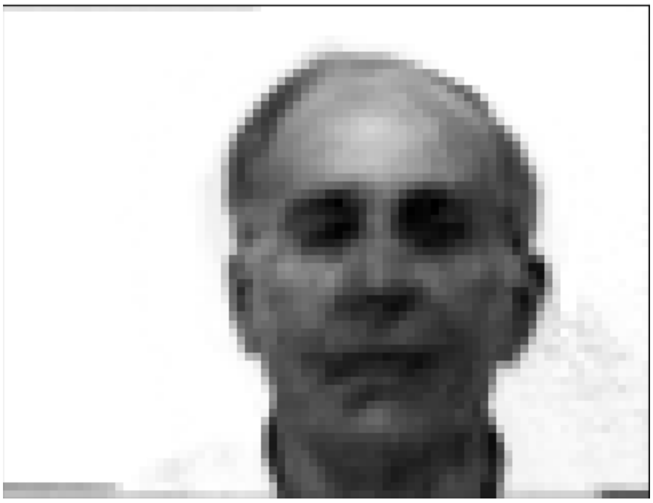
9

PCA:



mse: 0.0426

Autoencoder:



mse: 0.0008

Denoising autoencoder:

mse: 0.0025

10

PCA: 0.9

Autoencoder: 0.83334

Denoising autoencoder: 0.86667