HW4

1.

$\varphi(s) = s * f(s)$，$f(s) = \frac{1}{1+e^{-s}} \rightarrow \varphi'(s) = f(s) + s * f'(s) = \frac{1}{1+e^{-s}} + s * \frac{e^{-s}}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}} + s * \frac{e^{-s}}{(1+e^{-s})^2} = \frac{(s+1)*e^{-s}+1}{1+e^{-s}}$

2.

(a)

初始v_0是$[\frac{1}{3},\frac{1}{3},\frac{1}{3}]$，所以$v_1$是$[\frac{1}{2},\frac{1}{6},\frac{1}{3}]$，$v_2$是$[\frac{1}{3},\frac{1}{6},\frac{1}{2}]$，$v_3$是$[\frac{5}{12},\frac{1}{4},\frac{1}{3}]$，$v_4$是$[\frac{5}{12},\frac{1}{6},\frac{5}{12}]$，$v_5$是$[\frac{3}{8},\frac{5}{24},\frac{5}{12}]$

(b)

令答案是$[a, b, c]$，所以$a = b + \frac{c}{2}$，$b = \frac{c}{2}$，$c = a$，所以a:b:c=1:1/2:1，所以答案是$[\frac{2}{5},\frac{1}{5},\frac{2}{5}]$。

3.

首先考慮 L=1，這時$d^0 = 10$，$d^1 = 100 - 10 = 90$就是全部，所以 weight 的數量是 90+10=900。

考慮 L=2，這時$d^0 = 10$，$d^1 =$ 某個變數 k，$d^2 = 90 - k$，所以 weight 的數量是 10k+k(90-k)，最大值為 k=50，10k+k(90-k)=2500，最小值為 k=1，畢竟一層至少 1 個 neuron。這時 10k+k(90-k)=99。

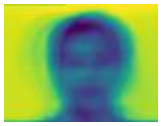
考慮 L=3，這時$d^0 = 10$，$d^1 =$ 某個變數 a，$d^2 =$ 某個變數 b，$d^3 = 90 - a - b$，所以 weight 的數量是 10a+ab+b(90 − a − b)= 10a+90b-b^2，最大值為 b=45，a=44，10a+90b-b^2=2465，最小值為 b=1，a=1 畢竟一層至少 1 個 neuron。這時 10a+90b-b^2=99。

所以最多 weight 的結構是$d^0 = 10$，$d^1 = 50$，$d^2 = 40$，weight 數量是 2500。最少 weight 的結構是$d^0 = 10$，$d^1 = 1$，$d^2 = 1$，$d^3 = 88$，weight 數量是 99。

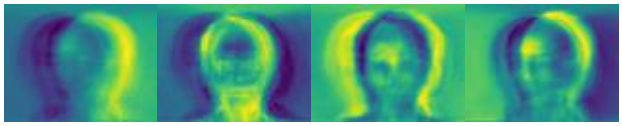
Report:

(a)

Mean 的圖如下:

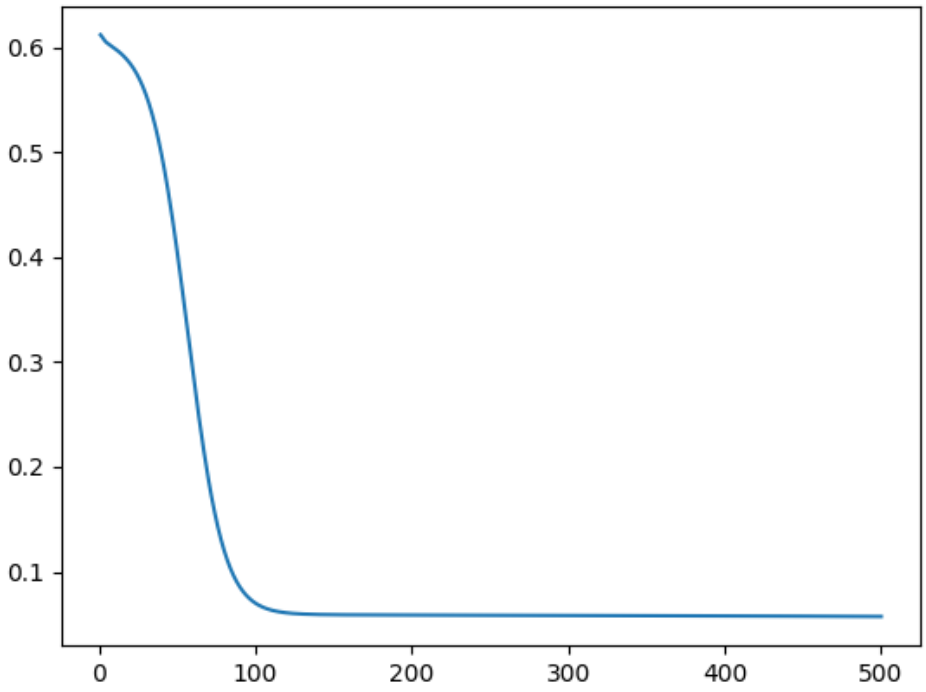


4 個特徵向量如下:



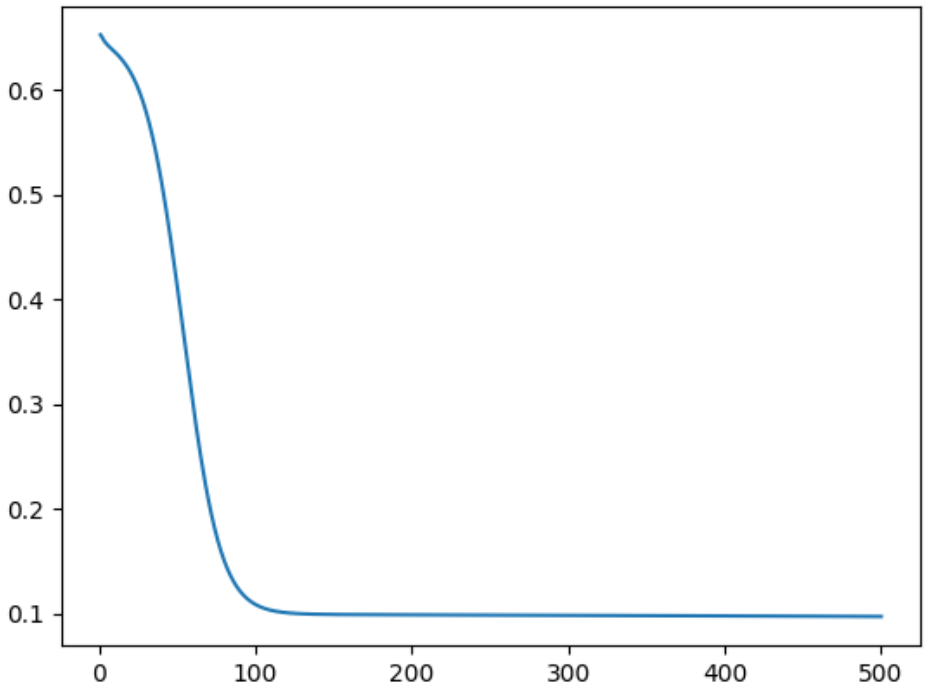
(b)

Autoencoder 的 loss 圖如下:



X 軸是 epoch 數，y 軸是 loss

Denoise autoencoder 的 loss 圖如下:

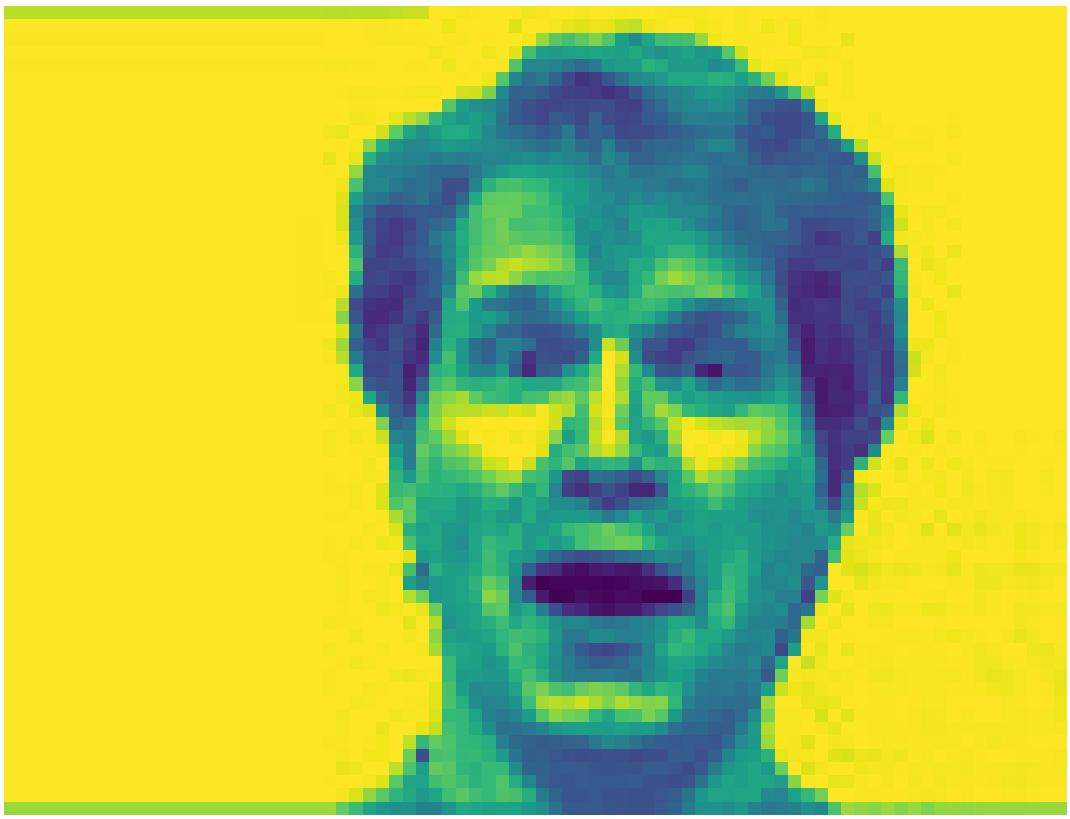


可以看到初始的 loss 比較高，後面的 loss 也比較高，整體曲線跟上面差不多。

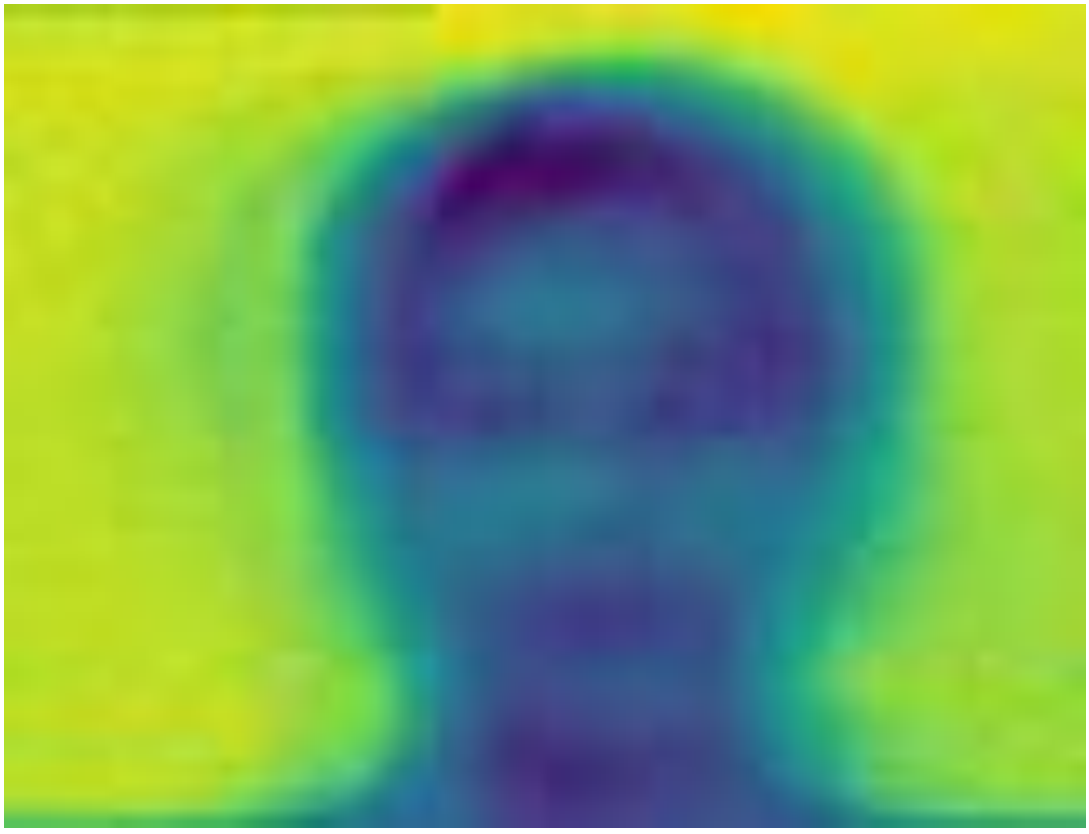
(c)

原圖如下:

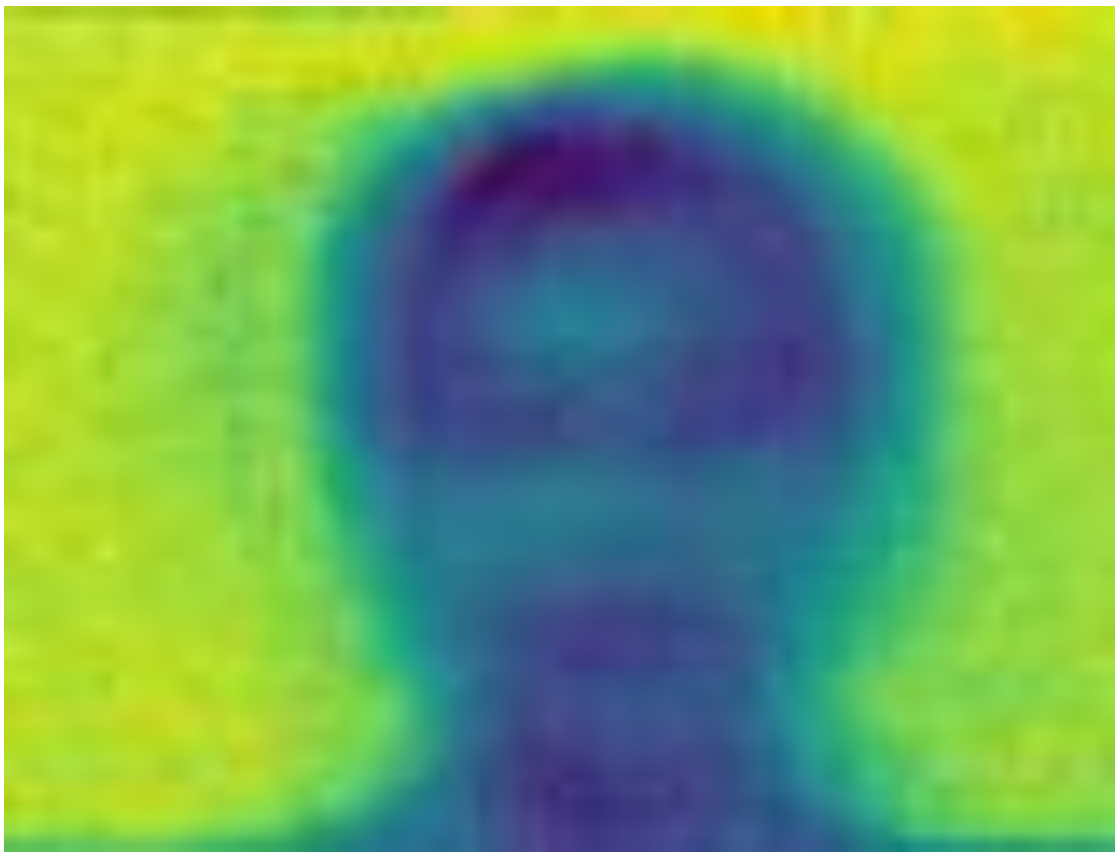
PCA 結果如下:



Autoencoder 結果如下:



Denoise autoencoder 結果如下;



mean squared error 如下:

Reconstruction Loss with PCA: 0.0

Reconstruction Loss with Autoencoder: 0.03538994262679841

Reconstruction Loss with DenoisingAutoencoder:

0.03582127402240118

(d)

首先我嘗試了增加一層的 layer，原本是

input_dim→encoder_dim→ encoder_dim//2→ RELU→ encoder_dim//2→ encoder_dim→ input_dim，我加了一層 encoder_dim//16，變成了 input_dim→encoder_dim→ encoder_dim//2→ encoder_dim//4→ RELU→ encoder_dim//16→encoder_dim//2→ encoder_dim→ input_dim。結果是 Acc from DenoisingAutoencoder: 0.5333333333333333，Reconstruction Loss with DenoisingAutoencoder: 0.036971449908690966
可以看到因為壓縮的維度變少了，所以資訊量損失比較大，所以 loss 會比較大，分數比較低。其實 reconstruction loss 分數變化不大，epoch 數夠就 ok，我把 encoder_dim//16 改成 encoder_dim//48 結果也差不多，但 epochs 數從 500 改成 10，reconstruction loss 就從 0.03 變成 0.68。可以看到雖然會損失資訊量，但壓縮能力還是很不錯的

第二個部分我改了 activation function，從 RELU 改成 sigmoid。結果如下:
Acc from DenoisingAutoencoder: 0.5333333333333333

Reconstruction Loss with DenoisingAutoencoder:

0.047502436050608624

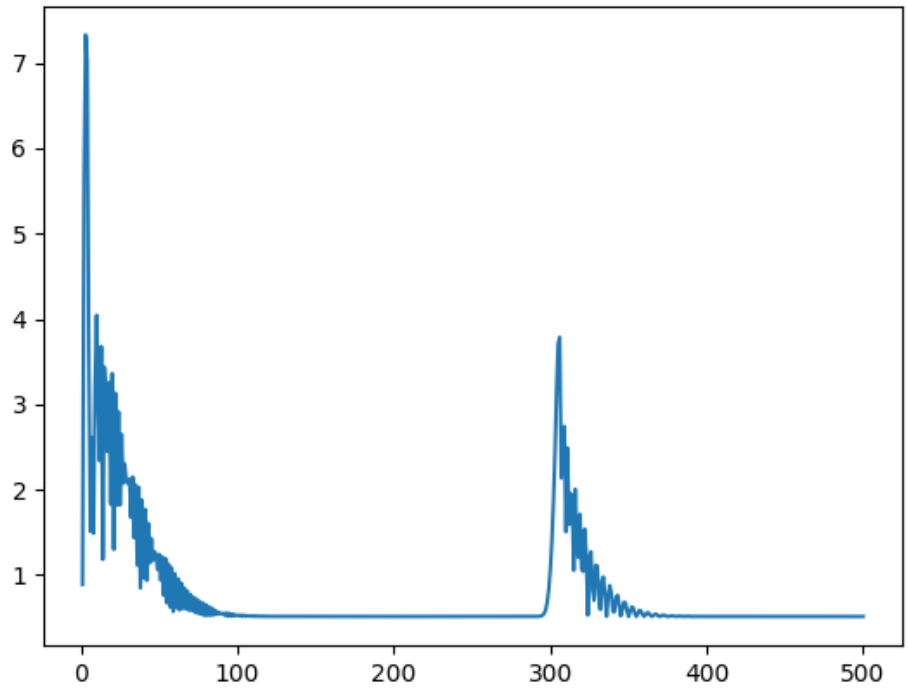
可以看到 loss 很大，表示訓練效果並不好，所以 RELU 在這題是比較好的。

(e)

我用的 optimizer 是 SGD，結果就是(b)的圖。我改成 Adam 和 RMSprop 看看。結果如下:

Adam:

因為前期 loss 太大了，為了比較好比較所以我設定 loss 都變成 0.3 次方來做圖。



Acc from DenoisingAutoencoder: 0.06666666666666667

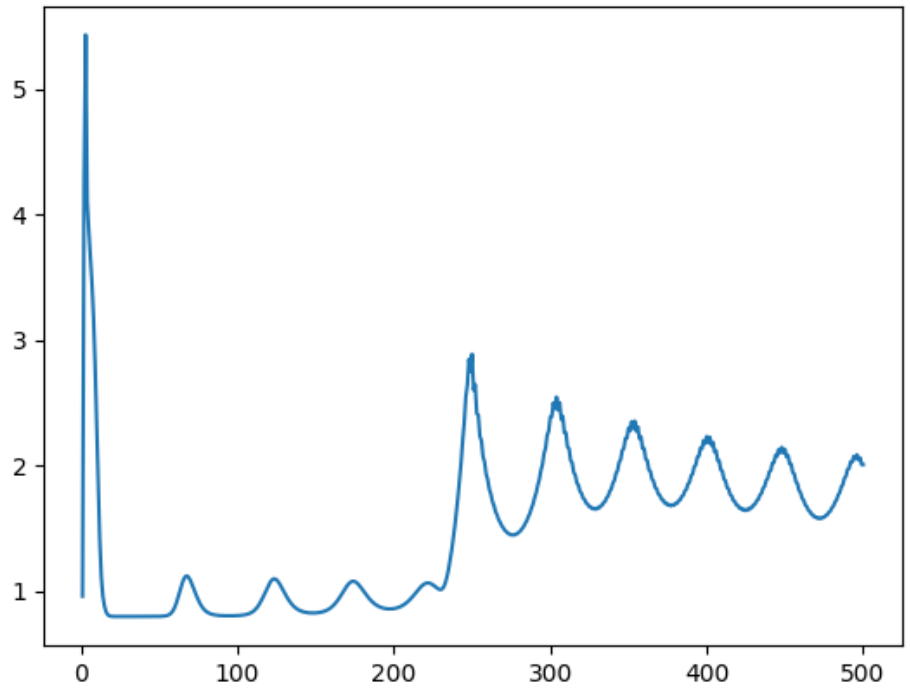
Reconstruction Loss with DenoisingAutoencoder:

0.042296824361242207

可以看到這個 optimizer 的行為非常神奇，前面下降速度很快，在 epoch=300 時 loss 突然變大，然後再下降。可以看到這個 optimizer 的 acc 非常非常低，我把中間值因出來，發現 transform 的結果是零向量。可以知道訓練出來的結果其實跟 input 沒關係，反正中間 encode vector 都是 0。雖然說 model 的訓練結果很爛，但 reconstruct 的結果還不錯，就很神奇

RMSprop:

因為前期 loss 太大了，為了比較好比較所以我設定 loss 都變成 0.1 次方來做圖。



Acc from DenoisingAutoencoder: 0.06666666666666667

Reconstruction Loss with DenoisingAutoencoder:

789.9607236266676

可以看到這個 optimizer 非常不契合這個 model。前面下降速度很快，Epoch=100 多時還好，epoch=250 之後就開始發瘋。結果超爛。