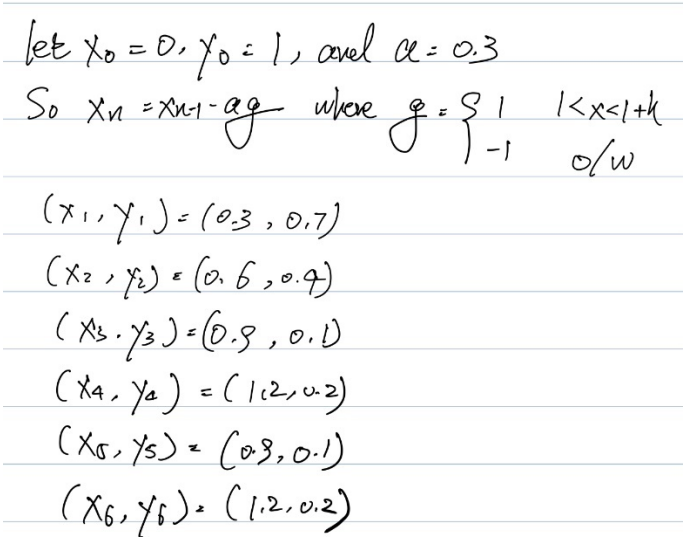
BS6207

Q1A:



The above result shows that it stuck at point 'x', making sense since gradient descent only looks for a local minimum, not a global minimum. As we can see, the gradients at both ends of 'x' increase, the cost function at the point 'o' would converge to 'x' after several epochs. So it results in stuck at 'x'.

Q1B:

```
import math

m = [0]
v = [0]
nm = [0]
nv = [0]
theta = [0]

b1 = 0.9
b2 = 0.999
a = 0.3
t = 0

for i in range(20):
  t += 1
  if theta[-1] < 1:
    g = -1
  else:
    g = 1
  m.append(b1 * m[-1] + (1 - b1) * g)
  v.append(b2 * v[-1] + (1 - b2) * (g ** 2))
  nm.append(m[-1] / (1 - (b1 ** t)))
  nv.append(v[-1] / (1 - (b2 ** t)))
  theta.append(theta[-1] - (a * nm[-1] * (math.sqrt(nv[-1]))))

print(max(theta))
```

```
1.4101842951299737
```

The above code simulates Adam Optimization with given parameters and 20 steps. From the result, we can see that the maximum of theta is 1.41. Thus, the bump's max-height 'h' in which the Adam Optimiser will escape the local minimum at 'x' equals 1.41 – 1 = 0.41.

Q2A:

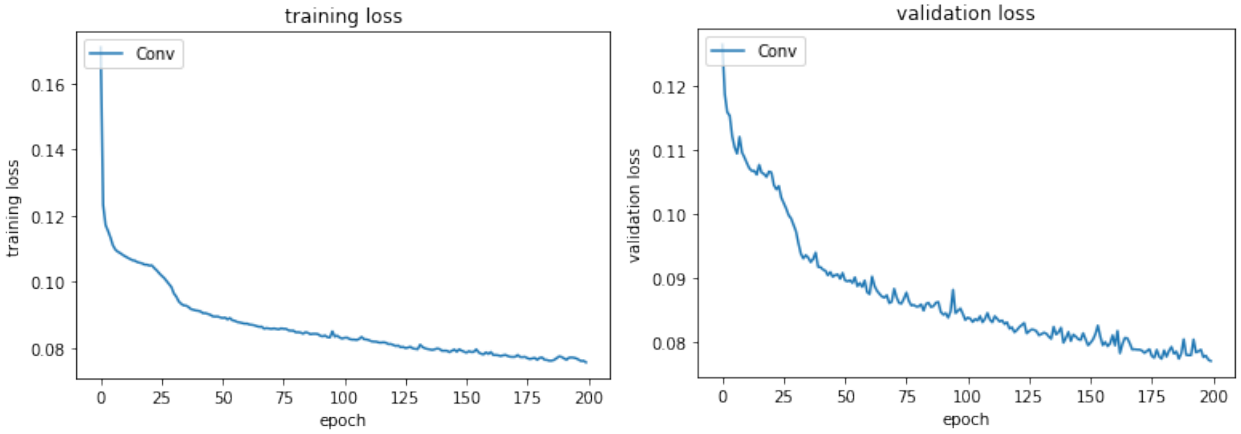
First, I loaded the MNIST data; then, I did some normalization to the MNIST dataset, including dividing by 255 and flattening to 1d to fit with the L1-loss function. After that, I split the dataset into training, validation and testing sets.

The network I choose for the autoencoder is a sequential network since CNN is not fit with the L1-loss function in this situation. The networks include several dense layers in both encoder and decoder for all three networks, each with a Relu activation function. The number of layers and each layer's size differs in each network because they have different latent space. For latent space = 2, I have 8 layers, size of each layer reduced from 256 to 2 for encoder; for latent space = 16, I have 4 layers, size of each layer reduced from 256 to 16 for encoder; and for latent space = 256, I have 1 layer.

Each network is compiled with Adam Optimiser and L1-loss Function, trained with 200 epochs and 256 batch size. For each network, I plot the loss graph and validation loss graph, as well the cluster map of 2d latent space and 16/256d latent space with t-SNE dimension reduction, and the confusion matrix of k-means clustering (where k = 10 since we know there are 10 clusters).

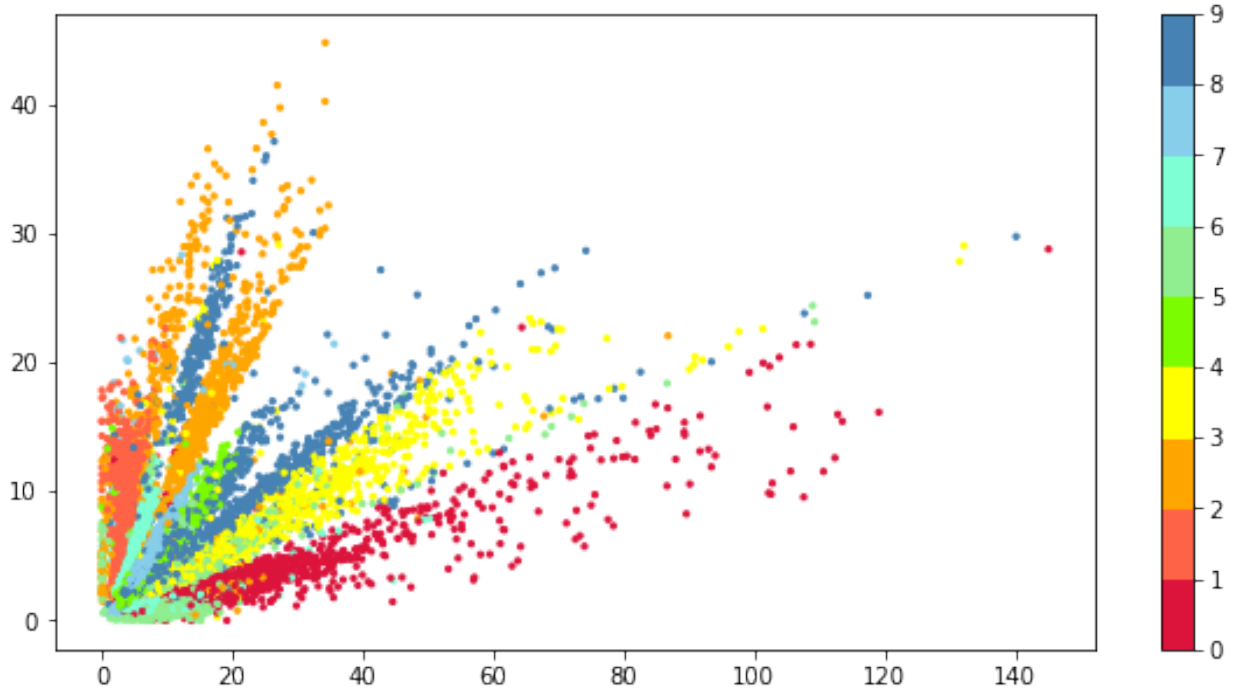
Latent Space = 2

Loss Graph:



Sample Re-constructions:

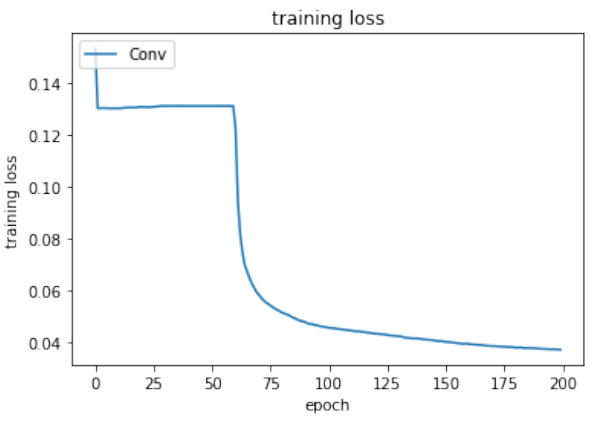
2D Plot:

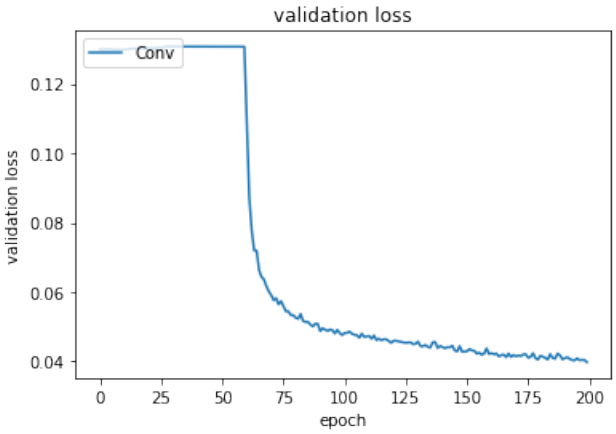


From the above results, we can see that the loss graph shows a significant decreasing trend along with epochs; the reconstructed graphs seem reasonable, but some flaws can be seen, such 2 is reconstructed as 1, 5 is reconstructed as 6, etc, which indicates the information loss along the way is significant. The 2D cluster plot is not very clear as well, where we can see a lot of digits are tangled together. Some digits are easy to recognize, such as 0 and 3; others are not clear.

Latent Space = 16

Loss Graph:



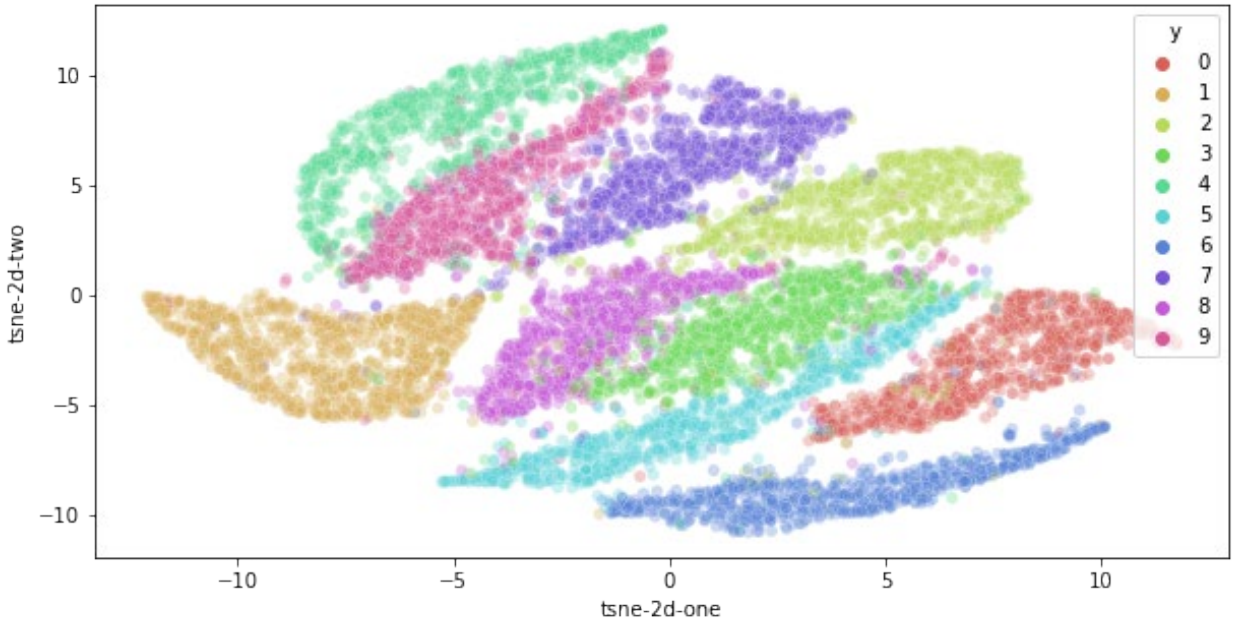


Sample Re-constructions:

K-Means Confusion Matrix:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	980
1	0.00	0.00	0.00	1135
2	0.02	0.01	0.01	1032
3	0.01	0.02	0.02	1010
4	0.00	0.00	0.00	982
5	0.11	0.13	0.12	892
6	0.00	0.00	0.00	958
7	0.00	0.00	0.00	1028
8	0.37	0.54	0.44	974
9	0.01	0.02	0.01	1009
accuracy			0.07	10000
macro avg	0.05	0.07	0.06	10000
weighted avg	0.05	0.07	0.06	10000

t-SNE Plot:

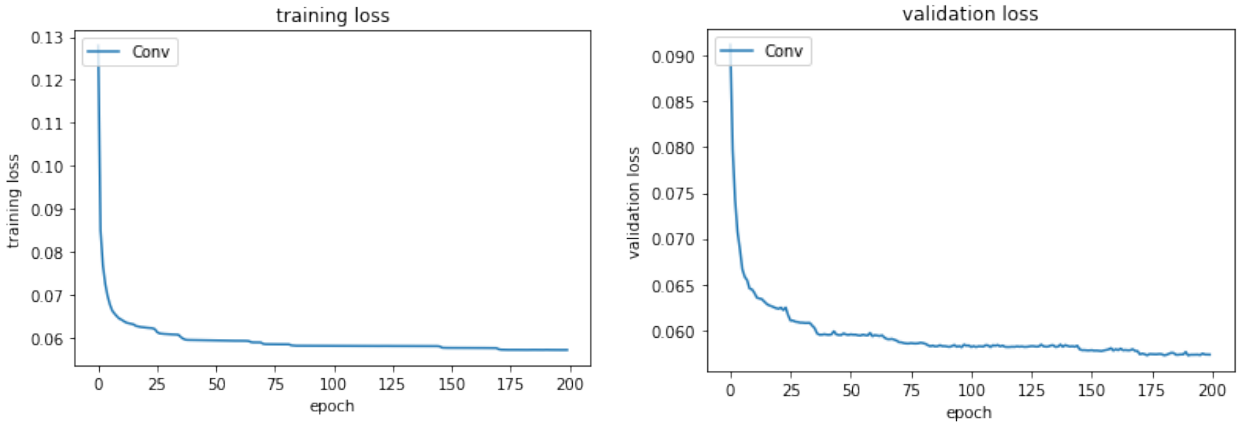


With latent space equal to 16, the loss graphs show some exciting trend but a decreasing elbow shape overall. The reconstructed graphs are obviously clearer and more accurate than the previous network.

However, the result of K-means does not look good, where the overall accuracy is less than 10%, but there are a lot of factors that may affect this result, such as the parameter of K-means, even the k-means itself is a good choice of model or not. The t-SNE graph shows a decent clustering map, where each cluster is well separated.

Latent Space = 256

Loss Graph:

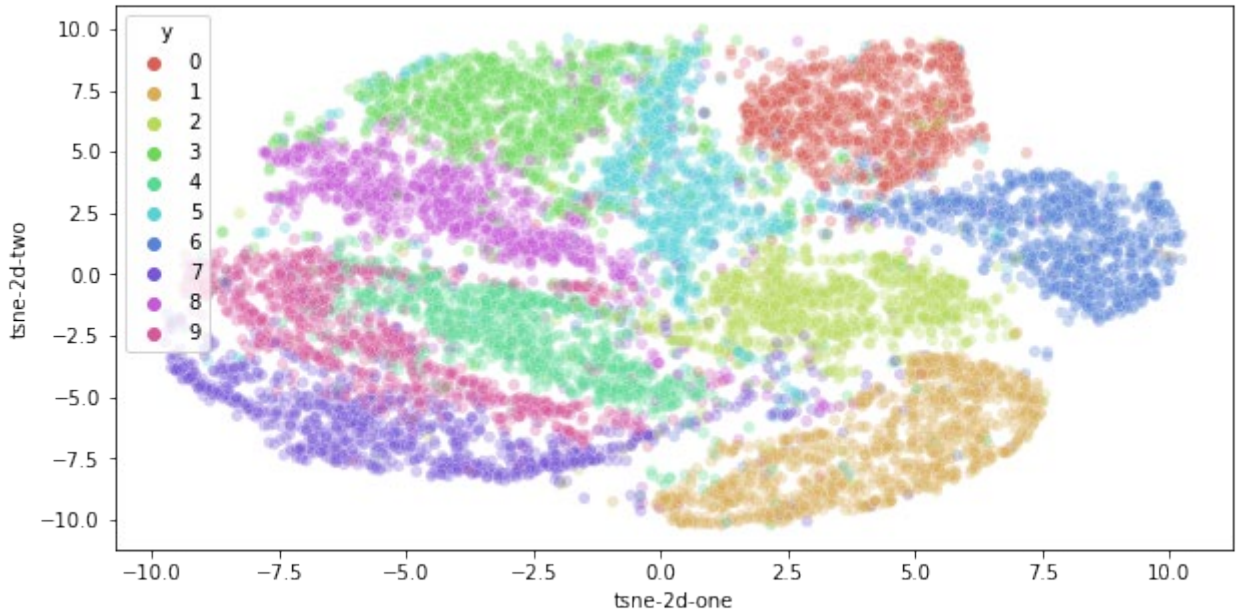


Sample Re-constructions:

K-Means Confusion Matrix:

	precision	recall	f1-score	support
0	0.01	0.01	0.01	980
1	0.00	0.00	0.00	1135
2	0.04	0.04	0.04	1032
3	0.01	0.01	0.01	1010
4	0.21	0.22	0.21	982
5	0.08	0.10	0.09	892
6	0.10	0.17	0.12	958
7	0.02	0.01	0.01	1028
8	0.01	0.01	0.01	974
9	0.00	0.00	0.00	1009
accuracy			0.06	10000
macro avg	0.05	0.06	0.05	10000
weighted avg	0.05	0.06	0.05	10000

t-SNE Plot:



With latent space equal to 256, the loss graphs show a perfect decreasing elbow shape, which means the learning process is perfect. However, the reconstructed graphs are obviously less clear than the previous network, making it hard to see whether they are accurate. It seems a huge latent space overkills the feature of each graph. Similarly, the result of K-means does not look good, where the overall accuracy is less than 10%. The t-SNE graph still shows a decent clustering map, where each cluster is well separated, but we can see a swirl pattern this time, which may also result from the huge latent space.

Q2B:

First, I used OpenCV to apply Gaussian Blur to the images. Here are some samples of the blurred images.

I used a CNN with 4 layers to build the dis_net, where the previous 3 layers are Conv2D layers with a Relu Activation function, a 2*2 maxplooling filter, and 0.25 dropout. The last layer combines falttern and dense layers to create output, using Relu and Softmax activation functions. I trained the CNN with Adam Optimiser and Binary Cross-Entropy Loss Function.

After that, I did a similar step with the previous question but changed the input images to the blurred images.

Latent Space = 2:

Latent Space = 16:

Latent Space = 256:

The reconstruction images show that the noise (blur) got filtered out after encoding. The dis_net result of each decoded image suggests that all of them are clear images. We can still see the same pattern as the previous question, such that 16 latent spaces produce the most accurate result.

Github Link:

https://github.com/Ironarrow98/Zhang_Chenxi_BS6207_A2