CPSC 453 Problem Set 3: Feed-Forward Neural Networks, Autoencoders, and Generative Models

Author: Wenxin Xu

The `xu_wenxin_ps3.zip` file includes following files:

- `xu_wenxin_ps3_report.pdf` : A detailed report.
- `/code`
    - `ps3_functions.py`: contains 2 classes (`FeedForwardNet, Autoencoder`) and 4 functions (`train, evaluate, mmd, kernel`)
    - `vae.py`: contains 1 class (`VAE`) and 3 functions (`VAE_loss_function, train, test`)
    - `GAN.py`: contains 2 classes (`Generator` and `Discriminator`) and 2 functions (`train_discriminator` and `train_generator`)
    - `xu_wenxin_ps3.ipynb`: A Jupyter Notebook contains all the code.

2 Classification of handwritten digits

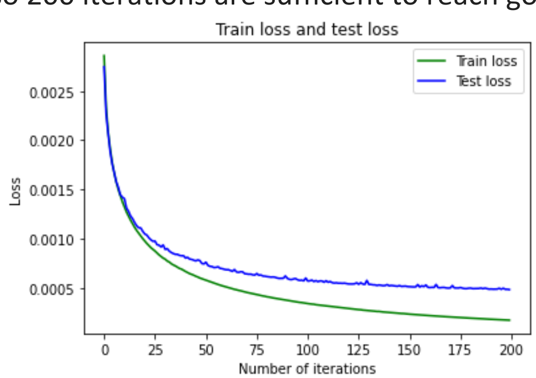
2.2 Feed-forward Neural Network

Question 2.2.1. *What percentage classification accuracy does this network achieve?*

My best performing net achieves 98% of classification accuracy on test set.

Question 2.2.2. *Create a plot of the training and test error vs the number of iterations. How many iterations are sufficient to reach good performance?*

I train the Feedforward Network using 200 iterations, the best validation accuracy of 0.981 occurred after epoch 183, so 200 iterations are sufficient to reach good performance.



Question 2.2.3. *Print the confusion matrix showing which digits were misclassified, and what they weremisclassified as. What numbers are frequently confused with one another by your model?*

Digit 5 is misclassified as digit 3 by 39 times, which is the most frequent misclassification; digit 9 is misclassified as digit 4 by 34 times; digit 2 is misclassified as digit 8 by 26 times; digit 7 is misclassified as digit 2 by 23 times and misclassified as digit 9 by 20 times; digit 5 is misclassified as digit 8 by 20 times. It makes sense because these digits are similar in some features.

Confusion Matrix

Predicted labels

True labels

	0	1	2	3	4	5	6	7	8	9
0	961	0	0	2	0	5	9	1	2	0
1	0	1113	3	3	1	1	4	2	8	0
2	12	1	939	12	15	0	11	15	26	1
3	1	0	19	934	1	22	2	12	14	5
4	2	2	3	0	929	0	8	2	5	31
5	7	2	2	39	7	792	13	4	20	6
6	11	3	3	2	12	12	911	1	3	0
7	4	9	23	3	9	1	0	956	3	20
8	5	4	3	14	8	11	9	8	909	3
9	13	7	0	11	34	7	1	16	8	912

Question 2.2.4. *Experiment with the learning rate, optimizer and activation function of your network.Report the best accuracy and briefly describe the training scheme that reached this accuracy.*

The best test accuracy is 0.981 with the following hyperparameters setting: learning rate = 0.01, optimizer = SGD, activation function before the hidden layer ReLU, number of epochs = 200, batch size = 128, hidden size = 512, number of hidden layers = 1.

The yellow highlighted hyperparameters are the one with high test accuracy and nice loss plot.

Table 1: Hyperparameter tuning of activation function

Fixed: Learning rate = 0.05, hidden_size = 128		
Activation function	Test accuracy	epoch
Linear	0.927	124
ReLU	0.980	73
SoftPlus	0.979	131
ELU	0.979	130
Tanh	0.980	147
sigmoid	0.936	147

Table 2: Hyperparameter tuning of learning rate

Fixed: Activation function = ReLU, hidden_size = 128		
Learning rate	Test accuracy	epoch
0.01	0.974	148
0.05	0.981	118
0.1	0.981	88
0.25	0.980	13
0.5	0.982	48

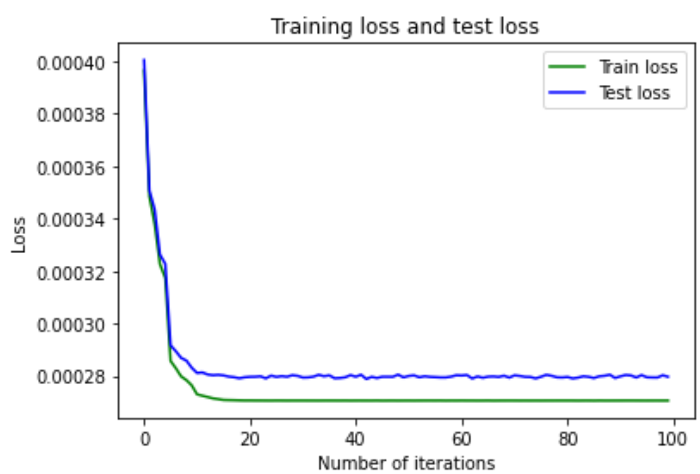
Table 3: Hyperparameter tuning of hidden size

Fixed: Activation function = ReLU, Learning rate = 0.01		
hidden_size	Test accuracy	epoch
32	0.973	15
64	0.981	39
128	0.980	22
256	0.982	23
512	0.981	183

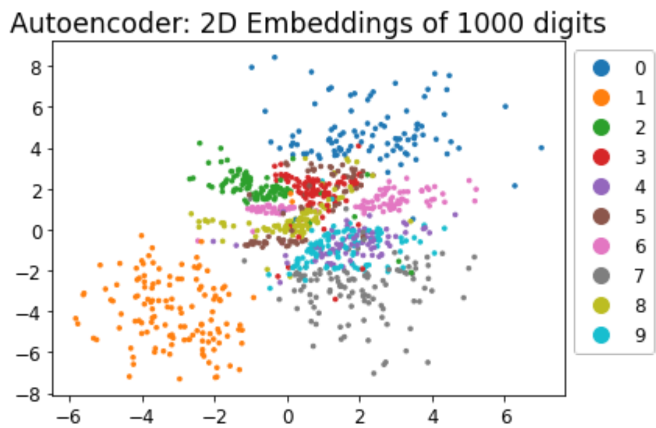
3 Autoencoder

3.1 MNIST

- *Train your Autoencoder on MNIST dataset.*



- *After training your model, plot the 2 dimensional embeddings of 1000 digits, colored by the image labels.*

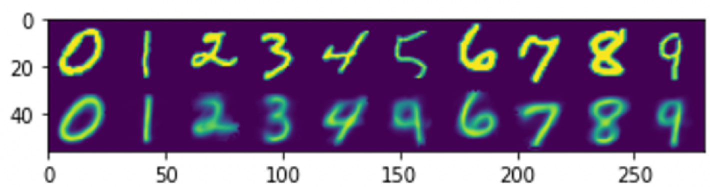


- *Produce side-by-side plots of one original and reconstructed sample of each digit (0 - 9). You can use the save_image function from torchvision.utils.*

From the comparison figure below, we can conclude that the trained Autoencoder performs a good job at reconstructing original digits, except for the original digit 5, it mistakenly reconstructed as digit 9.

Original digits

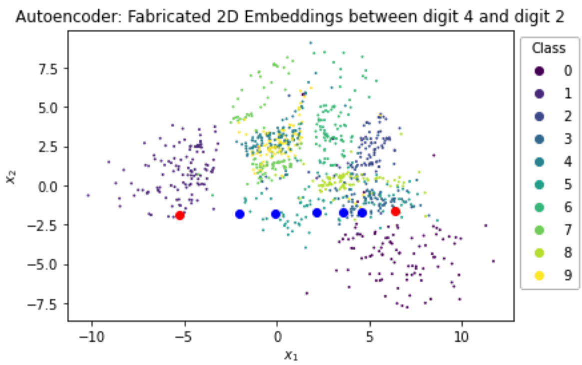
Reconstructed digits



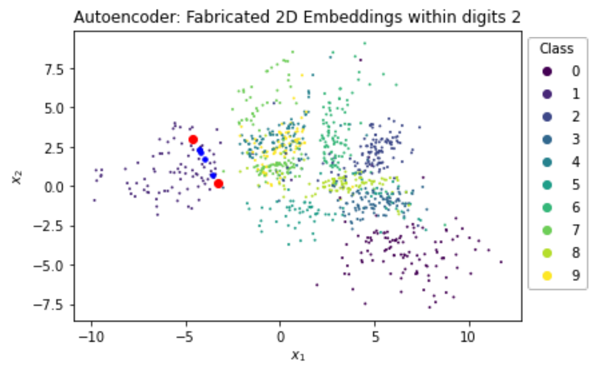
- *Now for something fun: locate the embeddings of two distinct images, and interpolate between them to produce some intermediate point in the latent space. Visualize this point in the 2D embedding. Then, run your decoder on this fabricated "embedding" to see if it the output looks anything like a handwritten digit. You might try interpolating between and within several different classes.*

The left image below are 5 digits (large blue points) generated by interpolating between 2 embeddings (large red points) from the first image in digit 1 class and that 50th image in digit 3 class using random weight. They can be easily identified as digits 5, 5, 5, 3, 3. The right image below are 5 digits generated by interpolating between 2 embeddings within digit 1 class, respectively using random weight. They can be easily identified as all digit 1s.

Between classes



Within classes



Question 3.1.1. *Do the colors easily separate, or are they all clumped together? Which numbers arefrequently embedded close together, and what does this mean?*

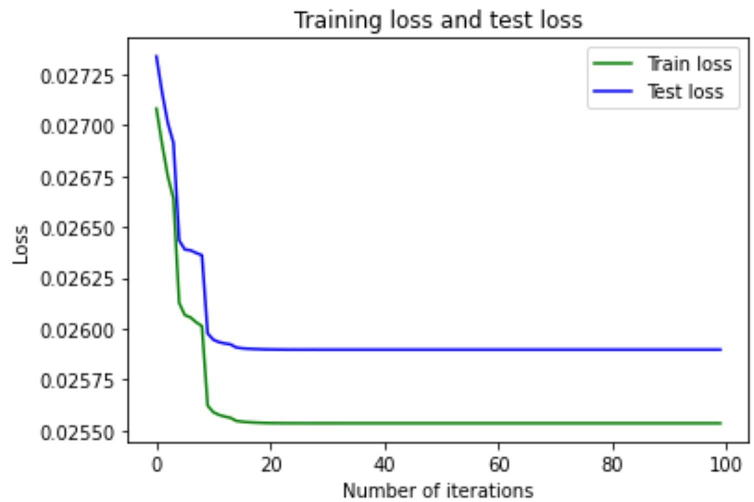
Clusters of digit 1 (orange), digit 6 (pink), digit 2 (dark green), and digit 0 (dark blue) are easily separate, but the clusters of digit 7 (gray), digit 4 (purple), digit 9 (light blue) have overlaps, also, clusters of digit 8 (light green), digit 3 (red), digit 5 (brown) have overlaps. Digits 4, 7 and 9 are embedded close together, digits 3, 5 and 8 are embedded close together. This means these handwritten digits are similar in some features that the network learned (e.g., perhaps the shape).

Question 3.1.2. *How realistic were the images you generated by interpolating between points in the latentspace? Can you think of a better way to generate images with an autoencoder?*

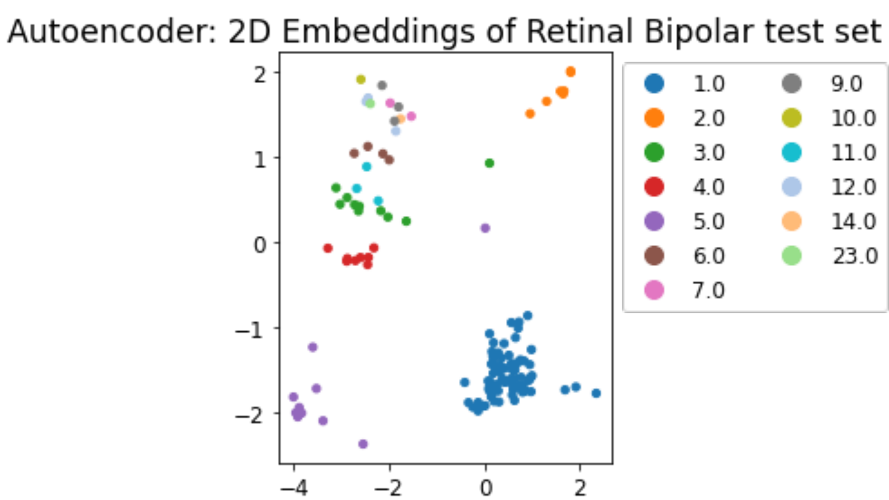
The generated images by interpolating between points in latent space aren't realistic, I can't recognize any digits from 0 to 9 from them. More hidden layers between encoder and decoder of the Autoencoder may give better result.

3.2 Biological Data: Retinal Bipolar Dataset

- *Train your Autoencoder on Retinal Bipolar dataset.*



- *After training your model, plot the 2-dimensional embedding of the test set. Color this with the ground truth cluster labels.*



Question 3.2.1. *How many clusters are visible in the embedding? Do they correspond to the cluster labels?*

I can see 6 clusters in the embeddings. They correspond to cluster label 1.0 (blue), label 5.0 (purple), label 2.0 (orange), label 4.0 (red), label 3.0 (dark green), label 6.0 (brown).

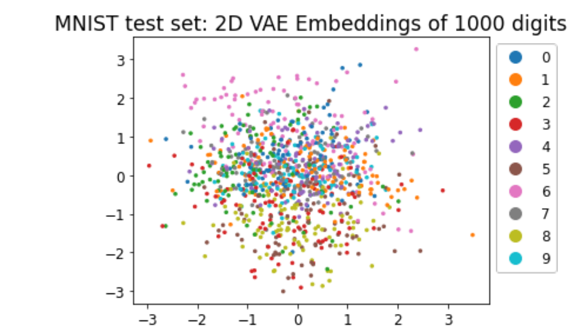
4 Generative Models

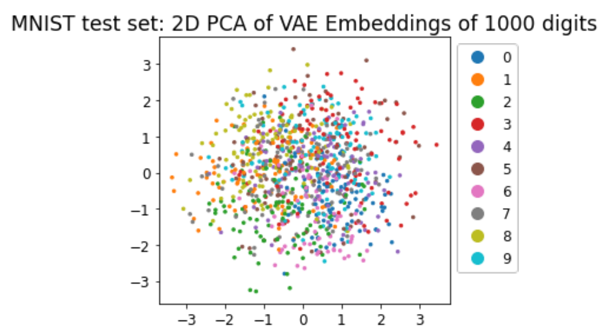
4.1 The Variational Autoencoder

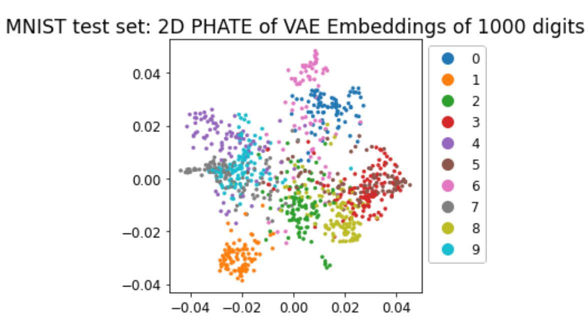
- *Train your VAE on MNIST. How well does it perform on the test set relative to your vanilla autoencoder?*

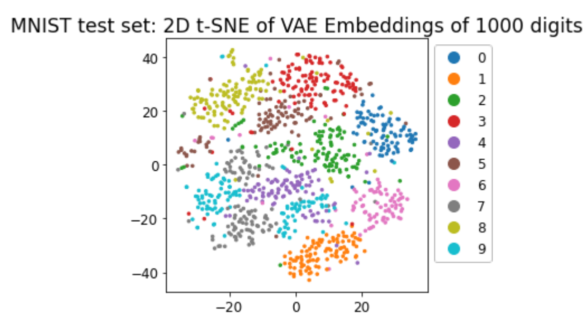
  For VAE, the minimum test loss of 100.764 occurred after epoch 93. For vanilla autoencoder, the minimum test loss of 0.000279 occurred after epoch 42. Although the loss function is different, we can see that VAE takes much longer to converge than vanilla autoencoder.

- *Visualize the latent space as a 2D plot, coloring each point by its label. Since our VAE is using a 20 dimensional latent space, you can try some of our dimensionality reduction tricks from the previous pset (PCA, PHATE, tSNE) to get a coherent 2 dimensional representation.*







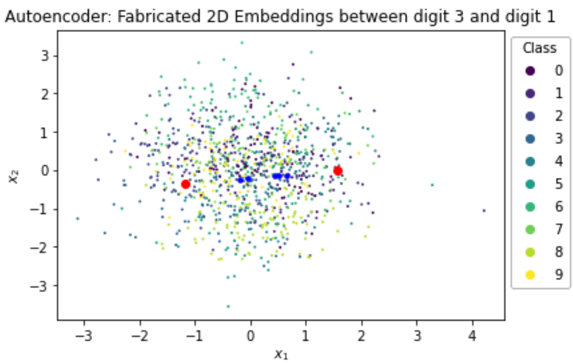


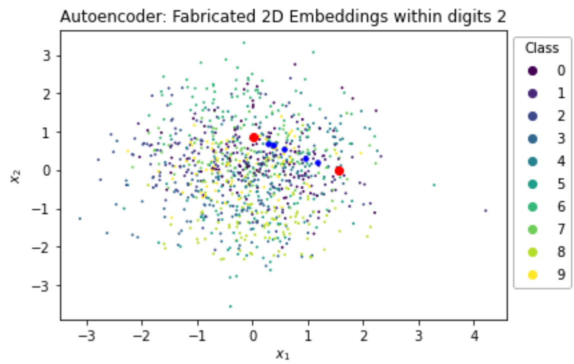
- *As before, try interpolating between two different images in the latent space. Run the fabricated embedding through the decoder to generate a never-before seen digit. You may wish to try interpolating between digits of the same class in addition to digits of different classes.*

  The left image below are 5 digits (large blue points) generated by interpolating between 2 embeddings (large red points) from the first image in digit 1 class and that 50th image in digit 3 class using random weight. They are like fusions of digits 5 and 3. The right image below are 5 digits generated by interpolating between 2 embeddings within digit 1 class, respectively using random weight. They can be easily identified as all digit 1s.

Between classes

Within classes





Question 4.1.1. *How does the VAE's latent space compare to the latent space of your previous autoencoder? Do the generated images have more clarity? Is this most noticeable between or within classes?*

Different classes in VAE's latent space have more overlap than that in autoencoder's latent space. But when applied PHATE and t-SNE, the embeddings of VAE of different classes are separate. This is most noticeable between classes.

Question 4.1.2. *In what situations would a VAE be more useful than a vanilla autoencoder, and whenwould you prefer a vanilla autoencoder to a VAE?*

If we want to generate new images, VAE is more useful than a vanilla autoencoder because for an image input, vanilla autoencoder can only generate images from embeddings what it had learned and will always produce the same reconstructed image in the training set. But VAE can generate images which aren't in the training set. And VAE is better in learning a more meaningful latent space than vanilla autoencoder.

If I want to do clustering analysis, i.e., classify between classes, I would choose vanilla autoencoder, because the KL Divergence regularization term of objective function of VAE imposes similarity between classes, which isn't good for separating different classes.

Question 4.1.3. *The distance between embeddings in your first autoencoder provided some measure of the similarity between digits. To what extent is this preserved, or improved, by the VAE?*

The distance between embeddings in VAE encoded more meaningful measure of similarity between digits, so it's improved. VAE encourages similar latent representations within clusters by sampling from a Gaussian distribution, and prevent different clusters from drifting too far away from the others by KL Divergence regularization term.

4.2 GANs

- *Train your GAN for 100 epochs, or more if necessary. After each epoch, visualize the generated images. Include some of the images from different epochs in your report.*

  The 6 images below are the generated images at different epochs (1, 20, 40, 60, 80, and 100) respectively. We can see, as training epochs increasing, the generated digits become more clear and realistic, which means the training is effective and GAN has learned some features of handwritten digits.

1 epoch

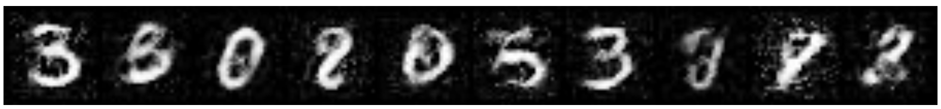
20 epoch

40 epoch

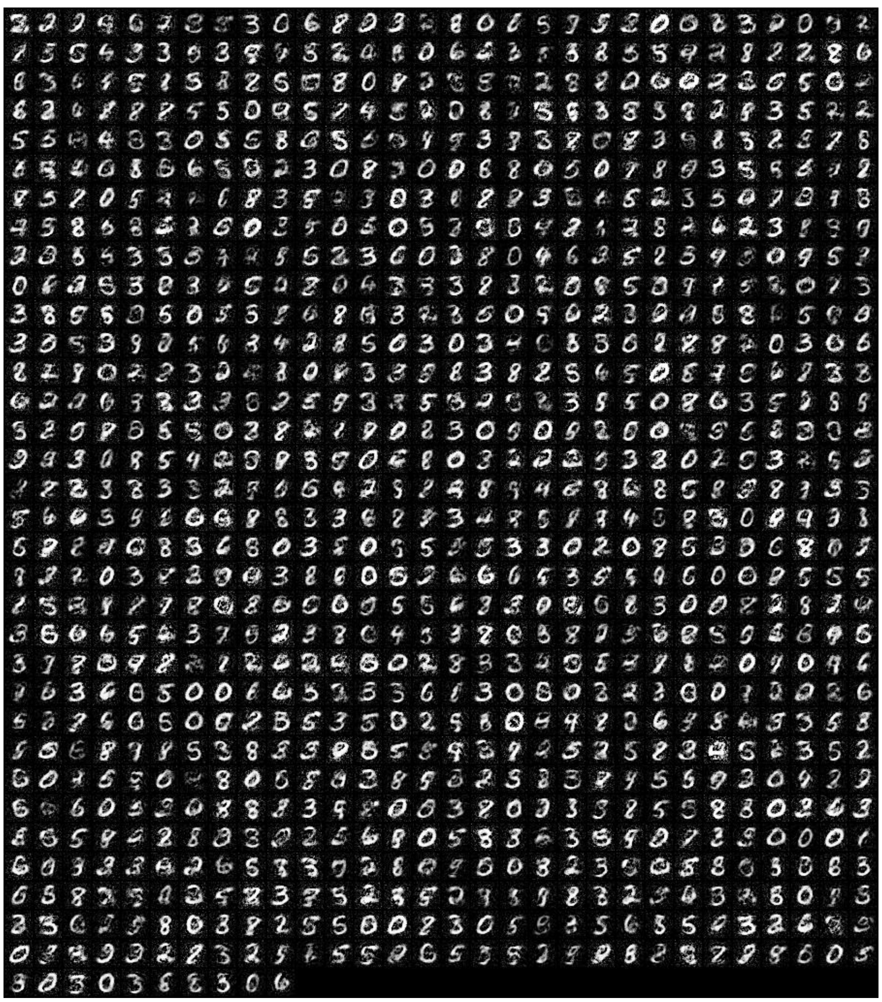
60 epoch

80 epoch

100 epoch



- *After your GAN has trained, generate 1000 sample digits, and include a few in the report.*



- *Using your best performing classifier from Part 2, classify these samples.*

  When compare the image of first 100 sample digits (left figure) of 1000 sample digits generated by generator of GAN to the predicted labels (right figure) generated by best performing Feedforward Net classifier trained in part 1 by feeding these fake images, we can see that the generator does a good job of fooling the classifier.

```
[[2 6 8 4 8 9 5 0 4 8]
 [2 0 5 8 0 5 5 0 3 5]
 [2 0 4 8 5 2 5 3 5 3]
 [5 5 8 5 2 2 3 8 5 8]
 [2 8 8 2 2 9 7 8 2 8]
 [3 0 8 8 6 3 3 3 2 0]
 [2 3 6 7 4 5 4 3 3 0]
 [2 3 3 7 2 0 2 3 3 0]
 [8 5 0 5 8 5 3 0 5 8]
 [9 2 3 3 5 3 8 8 5 6]]
```

Question 4.2.1. *Which generates more realistic images: your GAN, or your VAE? Why do you think this is?*

VAE generate more realistic images than GAN. Because the VAE is trained on real images, while GAN is trained on fake images generated by noise.

Question 4.2.2. *Does your GAN appear to generate all digits in equal number, or has it specialized in a smaller number of digits? If so, why might this be?*

Among the generated 1000 digits, GAN tend to generate digit 8 (193 times), digit 3 (181 times), digit 5 (155 times), digit 2 (139 times), digit 0 (132 times). I notice that digit 1 is only generated 3 times. The reason might be that the **curly shape** of these frequently generated digits are common among 0-9 digits, which is the nature of human handwriting.

5 Information Theory

5.1 Simple Distribution

- *Compute the Kullback-Leibler (KL) Divergence of the two distributions*
- *Compute the Earth Mover's Distance (EMD) between the two distributions.*
- *Compute the Maximum Mean Discrepancy (MMD) between the two distributions.*
- *Repeat all of the above as you increase the variance of your normal distribution to make it increasingly flat. As you do this, increase the range of the uniform distribution to match the range of the normal distribution, to ensure that the densities are comparable.*

```
sigma = 1
KL Divergence(a, b): 0.13190   time: 0.16147
KL Divergence(b, a): 0.16369   time: 0.16042
EMD: 0.30119       time: 0.00941
MMD: 0.08205       time: 33.96614

sigma = 2
KL Divergence(a, b): 0.27257   time: 0.15953
KL Divergence(b, a): 0.34455   time: 0.16436
EMD: 0.63254       time: 0.00291
MMD: 0.04548       time: 33.95288

sigma = 3
KL Divergence(a, b): 0.27812   time: 0.15806
KL Divergence(b, a): 0.34478   time: 0.15821
EMD: 0.91766       time: 0.00231
MMD: 0.02065       time: 33.81540
```

Question 5.1.1. *Based on the above measures alone, which divergence seems most accurate?*

I think EMD is most accurate. Because as the variance of normal distribution increases, the difference between the distribution of normal distribution and uniform distribution should also increase. EMD is the only measure that follows this trend.

5.2 MNIST Sample Distributions

- *Compute the Kullback-Leibler (KL) Divergence of the two distributions*
- *Compute the Earth Mover's Distance (EMD) between the two distributions.*
- *Compute the Maximum Mean Discrepancy (MMD) between the two distributions.*
- *Repeat the above a few times with different sets of MNIST samples, to get an idea of the expected range for each distance.*

I run the calculation for 5 times, from the results below. The KL Divergence is always 0 in both directions. The range of EMD is 10^1. The range of MMD is 10^-3. I would recommend MMD, because the difference between two different sets of MNIST shouldn't be too much. Given that, KL Divergence is too slow while EMD is too large and changes a lot.

```
Trial 1:

KL Divergence(a, b): -0.00000       time: 0.02319
KL Divergence(b, a): 0.00000  time: 0.02040
EMD: 9.63460       time: 0.00435
MMD: 0.00200       time: 34.96452

Trial 2:

KL Divergence(a, b): -0.00000       time: 0.01831
KL Divergence(b, a): 0.00000  time: 0.02318
EMD: 8.26964       time: 0.00387
MMD: 0.00200       time: 36.14374

Trial 3:

KL Divergence(a, b): -0.00000       time: 0.02207
KL Divergence(b, a): 0.00000  time: 0.01945
EMD: 13.70314      time: 0.00420
MMD: 0.00200       time: 35.67262

Trial 4:

KL Divergence(a, b): 0.00000  time: 0.03075
KL Divergence(b, a): -0.00000       time: 0.02219
EMD: 5.26309       time: 0.00398
MMD: 0.00200       time: 35.24130

Trial 5:

KL Divergence(a, b): 0.00000  time: 0.01920
KL Divergence(b, a): -0.00000       time: 0.02438
EMD: 9.35590       time: 0.00416
MMD: 0.00200       time: 35.48500
```

5.3 The GAN Distribution

- *Compute the Kullback-Leibler (KL) Divergence of the two distributions*
- *Compute the Earth Mover's Distance (EMD) between the two distributions.*
- *Compute the Maximum Mean Discrepancy (MMD) between the two distributions.*

```
KL Divergence(a, b): 0.04538     time: 0.13392
KL Divergence(b, a): 0.04993     time: 0.10319
EMD: 0.00048   time: 0.19552
MMD: 1.99581   time: 0.00080
```

Question 5.3.1. *Which divergence or distance showed the greatest discrepancy between the comparisonbetween real MNIST data and the comparison with the GAN?*

MMD showed greatest discrepancy between real MNIST data and GAN.

Question 5.3.2. *Which of these information measures would you recommend for judging a GAN's output? Why?*

I would recommend MMD, because the GAN's output is not very similar to real digits, so the divergence should be large.

Question 5.3.3. *How do the runtimes of these measures compare?*

The length of runtime: EMD > KL Divergence > MMD.