Practical Report

Assignment 3
Generative models

Daniel Lofeodo
Student ID: 20226991

Submitted on: **May 2, 2025**

Due Date: May 2, 23:00

<u>Instructions</u>

- For all questions that are not graded only on the answer, show your work! Any problem without work shown will get no marks regardless of the correctness of the final answer.

- Please try to use a document preparation system such as LaTeX. <u>If you write your answers by hand, note that you risk losing marks if your writing is illegible without any possibility of regrade, at the discretion of the grader.</u>

- Submit your answers electronically via the course GradeScope. <u>Incorrectly assigned answers can be given 0 automatically at the discretion of the grader.</u> To assign answers properly, please:

  - Make sure that the top of the first assigned page is the question being graded.
  - Do not include any part of answer to any other questions within the assigned pages.
  - Assigned pages need to be placed in order.
  - For questions with multiple parts, the answers should be written in order of the parts within the question.

- In the code, each part to fill is referenced by a TODO and 'Not Implemented Error'

- Questions requiring written responses should be short and concise when necessary.Unnecessary wordiness will be penalized at the grader's discretion.

- Please sign the agreement below.

- It is your responsibility to follow updates to the assignment after release. All changes will be visible on Overleaf and Piazza.

- Any questions should be directed towards the TAs for this assignment: *Vitória Barin Pacela, Philippe Martin.*

For this assignment, the GitHub link is the following: `https://github.com/philmar1/Teaching_IFT6135---Assignment-3---H25`

I acknowledge I have read the above instructions and will abide by them throughout this assignment. I further acknowledge that any assignment submitted without the following form completed will result in no marks being given for this portion of the assignment..

Signature: _____

Name: Daniel Lofeodo

UdeM Student ID: 20226991

1   VAE (68 points)

The code used in this section is available in the appendix.

6. (`report`, **8 pts**) Train a VAE using the provided network architecture and hyperparameters from '`q1_train_vae.py`'.

   Fill in the function '`loss_function.py`' from '`q1_train_vae.py`' by reusing your code from '`q1_vae.py`'.

   Optimize it with Adam, with a learning rate of 10^{-3}, and train for 20 epochs.

   Then, evaluate the model on the validation set.

   The model should achieve an average loss ≤ 104 on the validation set.

   Report the final validation loss of your model and plot the training and validation losses.
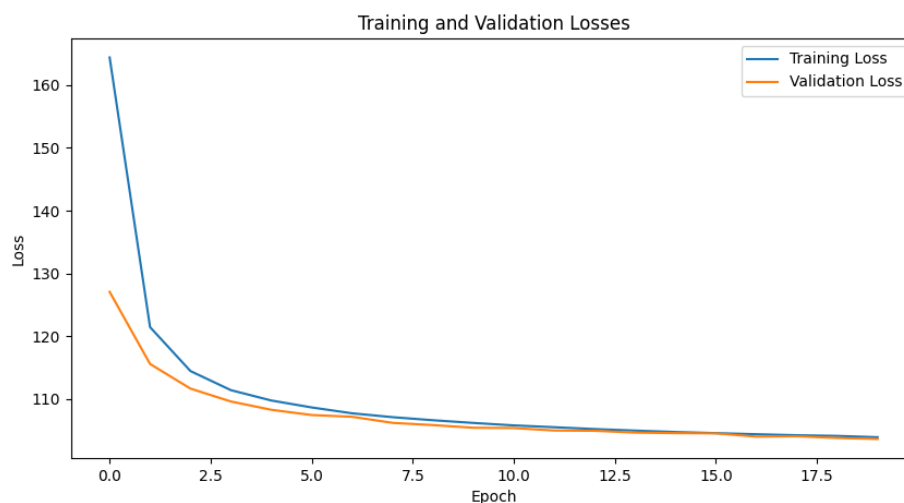

Figure 1: Training and validation losses for VAE.

Final validation loss: 103.6229

7. (`report`, **6 pts**) **Provide visual samples generated by the model.** Comment on the quality of the samples (e.g. blurriness, diversity, "realisticness").

Figure 2: Generated samples from VAE trained on MNIST.

The images generated by the VAE trained on MNIST are fairly diverse, with almost all digit classes being represented. However, nearly all the digits appear fuzzy or poorly defined, especially in the middle and bottom regions, which is a common deficiency of standard VAEs since they produce smooth, averaged outputs.

While some numbers are clearly identifiable, others are distorted, indicating that the model is struggling with sharpness and high-resolution details.

Overall, the results suggest that the VAE has been trained to learn a large latent space but without the generative detail needed for high-fidelity samples.

8. (`report`, **12 pts**) We want to see if the model has learned a disentangled representation in the
latent space. The VAE provided yields 20 latent factors. Plot the images from the latent traversals
with 5 samples per latent factor. What can you comment about the disentanglement of the latent
factors?

Figure 3: Latent traversals with 5 samples per latent factor.

The latent traversal plot illustrates how alterations to the individual latent dimensions make
minimal, often inconsequential changes in the output digits generated, with them being mostly
localized within the digit "1."

Homogeneity occurs when the VAE fails to acquire a disentangled representation within the latent.
Most of the latent factors apparently contribute little to nothing toward the output and might
represent redundancy or the decoder stifling them.

The lack of interpretable and independent variations across traversals is a sign of weak disentanglement,
likely due to issues like posterior collapse. More hierarchical goals, e.g. as in β-VAE, can be utilized
to facilitate more substantial latent factor separation.

9. (`report`, **10 pts**) **Compare between interpolating in the data space and in the latent space.** Pick two random points z_0 and z_1 in the latent space sampled from the prior.

   (a) For $\alpha = 0,\ 0.1,\ 0.2 \dots 1$ compute $z'_\alpha = \alpha z_0 + (1 - \alpha)z_1$ and plot the resulting samples $x'_\alpha = g(z'_\alpha)$.

   (b) Using the data samples $x_0 = g(z_0)$ and $x_1 = g(z_1)$ and for $\alpha = 0,\ 0.1,\ 0.2 \dots 1$ plot the samples $\hat{x}_\alpha = \alpha x_0 + (1 - \alpha)x_1$.

   Explain the difference between the two schemes to interpolate between images.

Figure 4: Interpolation in the latent space.

Figure 5: Interpolation in the pixel space.

The two interpolation schemes provide fundamentally different paradigms for generating intermediate samples between two points.

In the first image, interpolation is performed in the **latent space**, in which two latent vectors z_0 and z_1 are linearly mixed as $z_\alpha = \alpha z_0 + (1 - \alpha)z_1$, and the resulting vectors are decoded via the generator $x_\alpha = g(z_\alpha)$. This provides a smooth semantic transition between digit types, transitioning smoothly from a "6" to a "1" through intermediate digits such as "5" and "4". This is due to the character of the learned latent space, which represents high-level semantic features in a continuous and disentangled manner.

For comparison, the second picture shows interpolation **in pixel space**, with the images x_0 and x_1 being mixed together directly as $\hat{x}_\alpha = \alpha x_0 + (1 - \alpha)x_1$. This creates a linear fade between the two digits but lacks semantic consistency. The samples in between appear like blurry mixtures instead of actual digits, as pixel space is high-dimensional and does not correspond to meaningful structure for semantic content.

Generally, interpolation over latent space produces coherent and realistic intermediate digits, while pixel-space interpolation produces uninterpretable interpolations due to the lack of semantic structure.

2   Diffusion models

Implementing Denoising Diffusion Probabilistic Models (DDPM) and Classifier Free Guidance (CFG) models to generate MNIST style images (100 pts) .
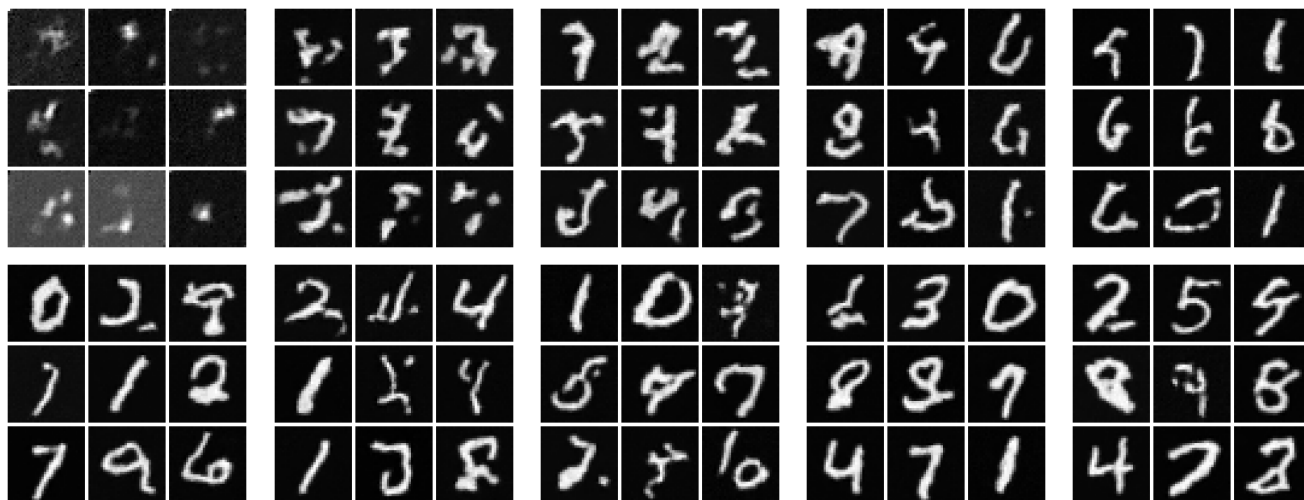
1. Unconditional generation (45 pts.)

Figure 6: DDPM-generated samples across epochs (from 0 to 18).

Sample Image Generation   The samples from the DDPM model over training epochs improve in quality and sharpness over time. The images at epoch 0 are mostly noise, as one might expect of a model that has not yet meaningfully learned. At epoch 2, blurry digit-shaped forms begin to appear, but the numbers remain noisy and vague. From epoch 4 to 10, the samples progressively improve, becoming cleaner and more coherent, with more distinguishable digits, but with some distortions and artifacts.

Beginning at epoch 12, the generated digits become more realistic and sharper, with digits such as "0", "1", and "7" becoming easily identifiable. By epoch 18, all the samples are clear and distinct from one another, indicating that the model has discovered a good generative prior. Nonetheless, there are certain images that still appear abnormal, such as questionable or deformed digits that may fall between class boundaries.

For even higher sample quality, several improvements can be made. Additional training steps or longer training for more epochs can lead to incremental improvements, especially if using learning rate scheduling to prevent overfitting. Augmenting the U-Net architecture with attention, increased batch sizes, or a change to a more powerful noise scheduler (e.g. cosine) can also be helpful. Finally, sampling tricks or classifier-free guidance like temperature scaling can be used to increase diversity and realism of the generated digits.
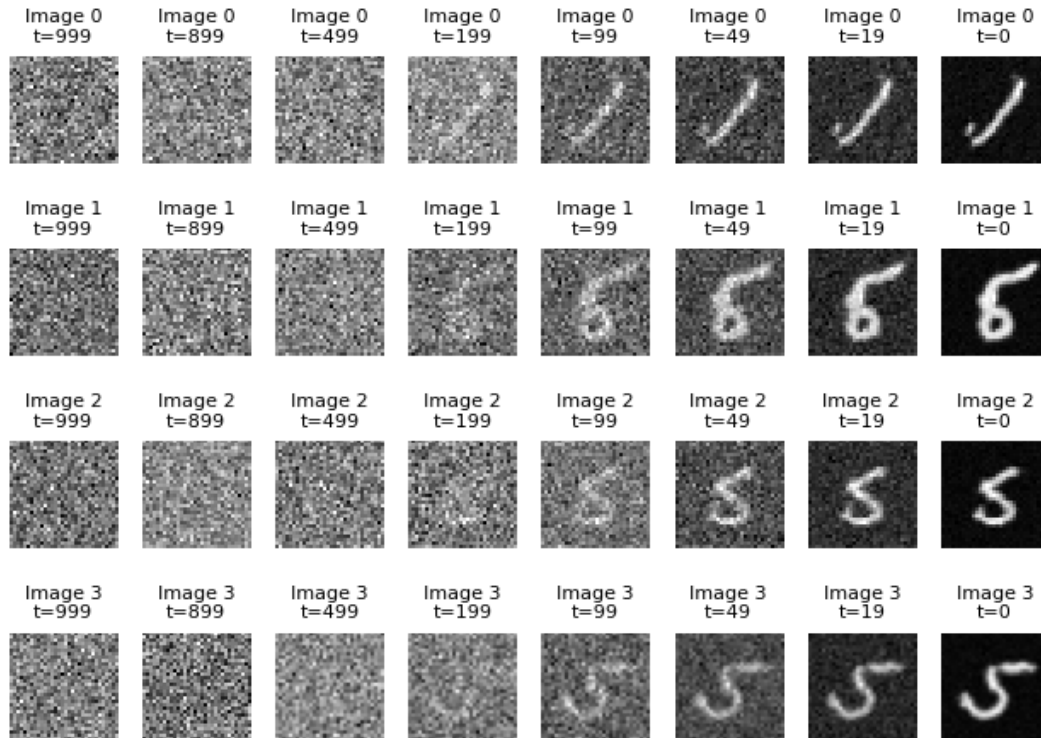
Figure 7: Intermediate samples generated at different steps of the diffusion process.

Reverse Diffusion Process   The above figure demonstrates the reverse diffusion process of the trained DDPM model, where samples transform from noise to structured digits as the timestep t drops from 999 to 0. The images at $t = 999$ are completely unrecognizable from random Gaussian noise. As the steps progress through $t = 899$, $t = 499$, and $t = 199$, the noise level gradually decreases and rough structures begin to develop.

At $t = 99$ and $t = 49$, digit-like patterns can be observed clearly, although still somewhat noisy. At $t = 19$ and $t = 0$, the samples become clean and realistic, very close to real MNIST digits. This gradual improvement confirms that the model has learned a good denoising trajectory, where early steps reconstruct global semantic structure and later steps refine fine details.

Smooth interpolation among timesteps allows the model to be properly trained. Additional generation quality improvement can be achieved with the choice of using DDIM-based sampling for improved convergence or classifier-free guidance for sharper and more class-consistent samples.

2. Conditional generation (55 pts.)

Figure 8: CFG-guided DDPM samples at various training epochs (0 to 14).

Sample Quality Across Epochs (CFG-DDPM)   The sampled images generated by the CFG-DDPM model during training epochs show a clear enhancement in image quality, structure, and recognition of digits.

By epoch 0, the outputs are just noise, with no visible digit structure, so the model has not yet learned a useful representation. By epoch 2, the digit-like shapes are beginning to emerge, but the images are extremely degraded, with indistinct edges and overlapping strokes.

By epoch 4 and epoch 6, the digits are identifiable, but many samples are still blurry, deformed, or half-created. Some digits such as "2" or "7" begin to show up in crisp form, while others appear as ambiguous combinations. The variety of shapes increases, but the model is still weak in semantic coherence.

Starting from epoch 8, the sample quality dramatically improves. Digits are crisper and more class representative at epoch 10 and epoch 12, although erratically distorted digits still remain. Visual accuracy and style diversity improve much more, with clear strokes and fewer ambiguous ones.

By epoch 14, the model generates sharp and diverse digits regularly. Most of the generated digits are well-defined, classifiable, and consist of smooth realistic contours. It shows that the model has learned to have very high generative ability and effectively use classifier-free guidance during sampling.

Overall, the training path confirms the effectiveness of CFG to improve generation quality. For further optimization, fine-tuning with lower learning rates, more profound model architectures, or employing DDIM sampling could attain sharper and faster convergence.

A   Appendix - VAE Code

A.1   1.6

The `main` function in `q1_train_vae.py` was modified to report and plot the losses.

```
if __name__ == "__main__":
# Lists to store losses for plotting
train_losses = []
val_losses = []

for epoch in range(1, args.epochs + 1):
    train_loss = train(epoch)
    val_loss = test()

    # Store losses for plotting
    train_losses.append(train_loss)
    val_losses.append(val_loss)

# Save the model
torch.save(model, 'model.pt')

# Print final validation loss
print('Final validation loss: {:.4f}'.format(val_losses[-1]))

# Plot training and validation losses
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Losses')
plt.legend()
plt.savefig('loss_plot.png')
plt.close()
```

A.2 1.7

A script was created to generate samples.

```python
import torch
from torchvision.utils import save_image
from q1_train_vae import VAE

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = torch.load('results/q1/model.pt', map_location=device)
model.eval()

def generate_samples(num_samples=64):
    with torch.no_grad():
        z = torch.randn(num_samples, 20).to(device)
        samples = model.decode(z)

        samples = samples.view(num_samples, 1, 28, 28)
        save_image(samples, 'results/q1/q7_generated_samples.png', nrow=8, normalize=True)

if __name__ == "__main__":
    print("Generating samples...")
    generate_samples()
    print("Samples saved to 'results/q1/q7_generated_samples.png'")
```

A.3 1.8

A script was created to generate the latent traversals.

```python
import torch
from torchvision.utils import save_image
from q1_train_vae import VAE

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = torch.load('results/q1/model.pt', map_location=device)
model.eval()

def generate_latent_traversals(num_samples=5, epsilon=2.0):
    """
    Generate latent traversals for each dimension of the latent space.

    Args:
        num_samples: Number of samples per dimension
        epsilon: Range of perturbation for each dimension
    """
    with torch.no_grad():
        z_base = torch.randn(1, 20).to(device)

        grid = []

        for dim in range(20):
            row = []
            for i in range(num_samples):
                z = z_base.clone()
                z[0, dim] = z_base[0, dim] + \
                    epsilon * (i - num_samples//2) / (num_samples//2)
                sample = model.decode(z)
                row.append(sample.view(1, 28, 28))

            row = torch.cat(row, dim=0)
            grid.append(row)

        grid = torch.cat(grid, dim=0)

        grid = grid.view(-1, 1, 28, 28)

        save_image(grid, 'results/q1/q8_latent_traversals.png', \
            nrow=num_samples, normalize=True)
```

```
if __name__ == "__main__":
    print("Generating latent traversals...")
    generate_latent_traversals()
    print("Latent traversals saved to 'results/q1/q8_latent_traversals.png'")
```

A.4 1.9

A script was created to generate the latent and data space interpolations.

```python
import torch
from torchvision.utils import save_image
from q1_train_vae import VAE

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = torch.load('results/q1/model.pt', map_location=device)
model.eval()

def generate_interpolations():
    """
    Generate and visualize interpolations between two random points in latent space
    and their corresponding pixel space interpolations.
    """
    with torch.no_grad():
        z0 = torch.randn(1, 20).to(device)
        z1 = torch.randn(1, 20).to(device)

        x0 = model.decode(z0)
        x1 = model.decode(z1)

        alphas = torch.linspace(0, 1, 11)

        latent_interpolations = []
        pixel_interpolations = []

        for alpha in alphas:
            z_alpha = alpha * z0 + (1 - alpha) * z1
            x_alpha = model.decode(z_alpha)
            latent_interpolations.append(x_alpha.view(1, 28, 28))

            x_hat_alpha = alpha * x0 + (1 - alpha) * x1
            pixel_interpolations.append(x_hat_alpha.view(1, 28, 28))

        latent_interpolations = torch.cat(latent_interpolations, dim=0)
        pixel_interpolations = torch.cat(pixel_interpolations, dim=0)

        latent_interpolations = latent_interpolations.view(-1, 1, 28, 28)
        pixel_interpolations = pixel_interpolations.view(-1, 1, 28, 28)

        save_image(latent_interpolations, 'results/q1/q9_latent_interpolation.png', \
            nrow=11, normalize=True)
```

```
        save_image(pixel_interpolations, 'results/q1/q9_pixel_interpolation.png', \
            nrow=11, normalize=True)


if __name__ == "__main__":
    print("Generating interpolations...")
    generate_interpolations()
    print("Interpolations saved to 'results/q1/q9_latent_interpolation.png' and \
        'results/q1/q9_pixel_interpolation.png'")
```