Variational Autoencoders

Songlin Zhao

August 1, 2023

Abstract

This report presents an extensive investigation into Variational Autoencoders (VAEs), a powerful class of generative models. I made an in-depth exploration of the mathematical formulations underpinning VAE and achieved a comprehensive understanding of latent models. I implemented VAE and tested it on MNIST and FashionMNIST dataset. A comparative analysis was performed between MLP-based and CNN-based encoder-decoder architectures, as well as different latent space dimensions, revealing insights into how specific network architectures can influence overall performance.

1 Background

1.1 Dimensionality Reduction

High-dimensional data poses significant challenges in terms of model overfitting and computational complexity. The vast number of features often leads to an overcomplicated model which tends to overfit the training data and poorly generalize to unseen data. Additionally, processing high-dimensional data requires substantial computational resources. Dimensionality reduction techniques address these issues by transforming the original high-dimensional data into a lower-dimensional space, while preserving as much information as possible. Techniques like PCA addresses this issue by projecting the original data onto a new subspace, which is spanned by the orthogonal axes (principal components) that explain the most variance in the data.

1.2 Autoencoders

Autoencoder (AE) is a neural network utilized for dimensionality reduction. It trains an encoder to transform the high-dimensional input data into a lower-dimensional latent space representation, and a decoder to reconstruct from the latent representation. However, one significant limitation of standard AEs is the lack of control over the latent space. **The autoencoder is solely trained to encode and decode with as few loss as possible, no matter how the latent space is organized.** Typically, the latent space of AEs is discontinuous. If the latent space has discontinuities, sampling from these regions can result

in the decoder producing unrealistic outputs. This occurs because, during training, the decoder never saw encoded vectors from these specific areas of the latent space.

1.3 Intuition behind VAE

What we expect is that we can generate new data by randomly sampling points from the latent space and passing them through the decoder. This requires the latent space to be **continuous** (two close points in the latent space should give similar reconstructions), **meaningful** (the decoded latent representations look alike training data), and **structured** (latent representations from different classes should not be far apart). To enable these properties, we can modify the model based on Autoencoders to form VAE:

- **Probabilistic Encoding and Decoding**: Unlike traditional AEs that directly map inputs to a point in latent space, VAEs map inputs to a distribution. When an input is processed through the encoder of a VAE, **the output isn't a fixed point but rather mean and variance of a gaussian distribution**. Rather than decoding a single point, we can sample multiple points from this gaussian distribution and decode them, yielding slightly different outputs each time. Intuitively, the mean vector controls the center of an encoded input while variance controls the radius of the "circle". In this case, the decoder can see a bunch of data from the "circle" and learns to reconstruct them, so that nearby points in the latent space will be decoded similarly. Now we make the latent space continuous (two close points in the latent space give similar reconstructions).

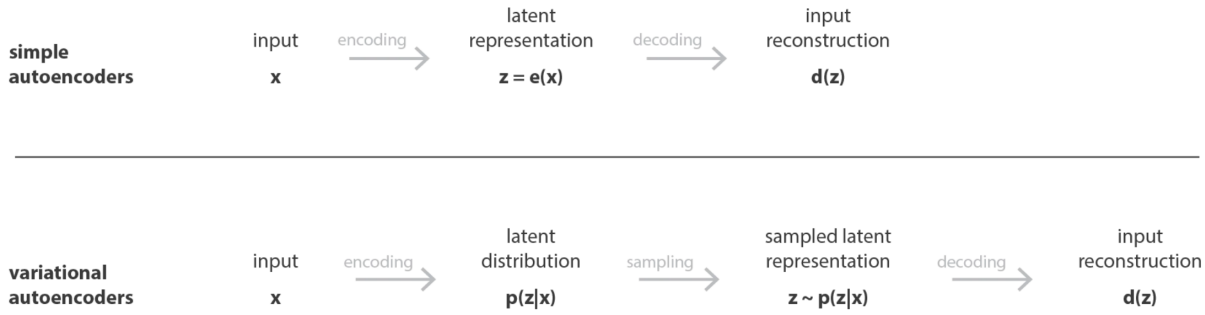


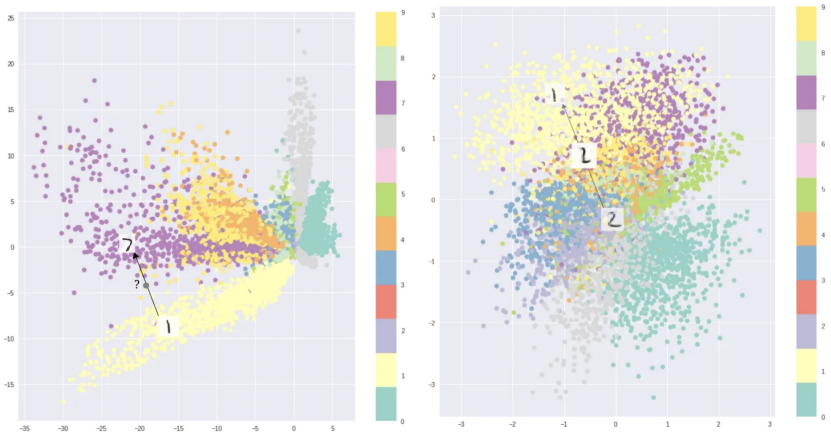
Figure 1: Difference between AE and VAE

- **Regularization of the Latent Space**: Probabilistic encoding enables the model to learn a continuous latent space on a local scale. However, the encoder can learn to generate very different μ's and very small σ's for different classes, resulting in a sparse latent space where many spaces are still meaningless, which is not what we want. So, in order to avoid these effects we have to set constraints on both the covariance matrix and the mean of the distributions returned by the encoder. **This is done by enforcing distributions to be close to a standard normal distribution**. This

way, we require the covariance matrices to be close to the identity, preventing punctual distributions, and the mean to be close to 0, preventing encoded distributions to be too far apart from each others.

By modeling in this magical way, the latent space of a VAE is suitable for generating new samples. As shown in the figure, the latent space of AE is discontinuous, and poorly-structured: the range of the latent representations can be wide, and interpolation does not generate meaningful samples. The latent space of VAE is well-regularised.

Figure 2: Comparison of latent space between AE and VAE



(a) Latent space of AE (b) Latent space of VAE

2 Mathematical Formulation

Previously we discussed the intuition behind VAE, not let's delve into the mathematics behind VAE to gain a deeper understanding of why we want to minimize both the reconstruction loss and KL divergence loss.

By assumption, all observed data have an underlying latent representation z. Let's now assume that $p(z)$ is a standard Gaussian distribution and $p(x \mid z)$ is a normal distribution whose mean is a function of latent variable z:

$$p(z) = \mathcal{N}(0, I)$$

$$p(x \mid z) = \mathcal{N}(f(z), cI)$$

VAE learns a model to maximize the likelihood $p(x)$ for all observed x. The probability $p(x)$ can be computed in two ways:

$$p(x) = \int p(x, z)dz$$

$$p(x) = \frac{p(x, z)}{p(z \mid x)}$$

The first equation is intractable because it involves integrating all possible latent variables z. The second equation needs the ground truth $p(z|x)$. However, we can derive a term called Evidence Lower Bound (ELBO) to be a proxy objective with which to optimize a latent variable model.

2.1 Evidence Lower Bound

The definition of ELBO is:

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right] = ELBO$$

where $q_\phi(z \mid x)$ is the probability that we want to optimize. We can prove the above inequality by two methods:

$$\begin{aligned}
\log p(\boldsymbol{x}) &= \log \int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} \\
&= \log \int \frac{p(\boldsymbol{x}, \boldsymbol{z}) q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})} d\boldsymbol{z} \\
&= \log \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&\geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right]
\end{aligned}$$

In this derivation, Jensen's Inequality is applied. To gain insights into why optimizing ELBO is useful, we can prove it in another way:

$$\begin{aligned}
\log p(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x})] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z}) q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p(\boldsymbol{z} \mid \boldsymbol{x}) q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right] + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}{p(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right] + \mathcal{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p(\boldsymbol{z} \mid \boldsymbol{x})\right) \\
&\geq \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right]
\end{aligned}$$

Since $\log p(x)$ is constant with respect to ϕ, maximizing ELBO with respect to ϕ is equivalent to minimizing $\mathcal{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p(\boldsymbol{z} \mid \boldsymbol{x})\right)$, i.e., to make the approximated posterior $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ match the truth posterior $p(\boldsymbol{z} \mid \boldsymbol{x})$.

We can dissect the ELBO term further:

$$\begin{aligned} ELBO &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z})p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z})\right] + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right] \\ &= \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}\mid\boldsymbol{z})\right]}_{\text{reconstruction term}} - \underbrace{\mathcal{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\|p(\boldsymbol{z})\right)}_{\text{KL divergence of priors}} \end{aligned}$$

In the above derivation, ϕ is the parameters for the encoder $q_\phi(\boldsymbol{z}\mid\boldsymbol{x})\|p(\boldsymbol{z}))$, and θ is the parameters for the decoder. During training of VAE, we aim to find the optimal θ and ϕ such that ELBO of all training data is maximized. Since by assumption, $p(x\mid z)$ is a normal distribution whose mean is $f(z)$ and variance is cI, we can rewrite the above formula as:

$$\begin{aligned} \boldsymbol{\phi}^*, \boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\phi},\boldsymbol{\theta}} E_x\left(\log p(x)\right) \\ &\approx \arg\max_{\boldsymbol{\phi},\boldsymbol{\theta}} E_x\left(\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right]\right) \\ &= \arg\max_{\boldsymbol{\phi},\boldsymbol{\theta}} E_x\left(\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}(\log p(\boldsymbol{x}\mid\boldsymbol{z})) - \mathcal{D}_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\mid\boldsymbol{x}), p(\boldsymbol{z})\right)\right) \\ &= \arg\max_{\boldsymbol{\phi},\boldsymbol{\theta}} E_x\left(\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left(-\frac{\|\boldsymbol{x}-f_\theta(\boldsymbol{z})\|^2}{2\boldsymbol{c}}\right) - \mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}(\boldsymbol{z}|\boldsymbol{x}),p(\boldsymbol{z})}\right)\right) \\ &\approx \arg\max_{\boldsymbol{\phi},\boldsymbol{\theta}} E_x\left(\sum_{l=1}^{L}\left(-\frac{\|\boldsymbol{x}-f_\theta(\boldsymbol{z}^{(l)})\|^2}{2\boldsymbol{c}}\right) - \mathcal{D}_{\mathrm{KL}}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}\mid\boldsymbol{x})\|p(\boldsymbol{z})\right)\right) \end{aligned}$$

where c is a hyperparameter that controls the balance between reconstruction term and KL divergence term. We use Monte Carlo estimate to approximate the reconstruction term, L is typically 1 in practice.

3 Implementation and Results

In my implementation, I used $c = 0.05$ in the loss. The optimizer is Adam with learning rate 0.0025 for MLP encoder, and 0.0015 for CNN encoder, the batch size is 256, and the model is trained for 15 epochs. Test data is a set of randomly sampled data of size 100, and the test loss is monitored during training.
The training data is normalized to $[0, 1]$. The encoder output is of shape $(batch\_size, 2z)$, where z is the latent dimension. The output of the encoder is split into m, v representing

mean and variance. To guarantee the variance to be positive, I used $v = softplus(h) + 10^{-8}$, where softplus function transforms the output to $[0, 1]$ and 10^{-8} is to guarantee the variance to be non-zero. The latent representation is sampled using reparameterization trick to guarantee the gradient flow. The deocoder output $\hat{x}$ is transformed to $sigmoid(\hat{x})$ to make the reconstruction in the range $[0, 1]$.
I implemented two types of encoder-decoder network.

3.1 MLP Encoder-Decoder

The input data [batch, 1, 28, 28] is first flattened to a 784-long vector and then fed into a fully connected network with hidden dimensions 300, 100, 20 and ELU activation. The mean and variance of the encoded distribution is obtained by a final FC layer, and the latent representation is obtained by the reparameterization trick. The latent dimension is 2 by default.

Figure 3: VAE Network structure of MLP

```
FCEncoder(
  (activation): ELU(alpha=1.0)
  (encode_layers): Sequential(
    (0): Linear(in_features=784, out_features=300, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=300, out_features=100, bias=True)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=100, out_features=10, bias=True)
    (5): ELU(alpha=1.0)
    (6): Linear(in_features=10, out_features=4, bias=True)
  )
)
```

(a) MLP Encoder

```
FCDecoder(
  (activation): ELU(alpha=1.0)
  (decode_layers): Sequential(
    (0): Linear(in_features=2, out_features=10, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=10, out_features=100, bias=True)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=100, out_features=300, bias=True)
    (5): ELU(alpha=1.0)
    (6): Linear(in_features=300, out_features=784, bias=True)
  )
)
```

(b) MLP Decoder

3.2 CNN Encoder-Decoder

The CNN encoder utilizes CNN and Batch Normalization layers to extract the information in the image. The extracted filters are flattened and pass through the final FC layer to get the encoder output. The decoder utilizes Transposed Convolution to upsample the extracted feature maps and reconstruct the original image.

Figure 4: VAE Network structure of CNN

```
CNNEncoder(
  (activation): ELU(alpha=1.0)
  (conv_layers): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ELU(alpha=1.0)
    (6): Conv2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ELU(alpha=1.0)
  )
  (fc): Linear(in_features=512, out_features=4, bias=True)
)
```

(a) CNN Encoder

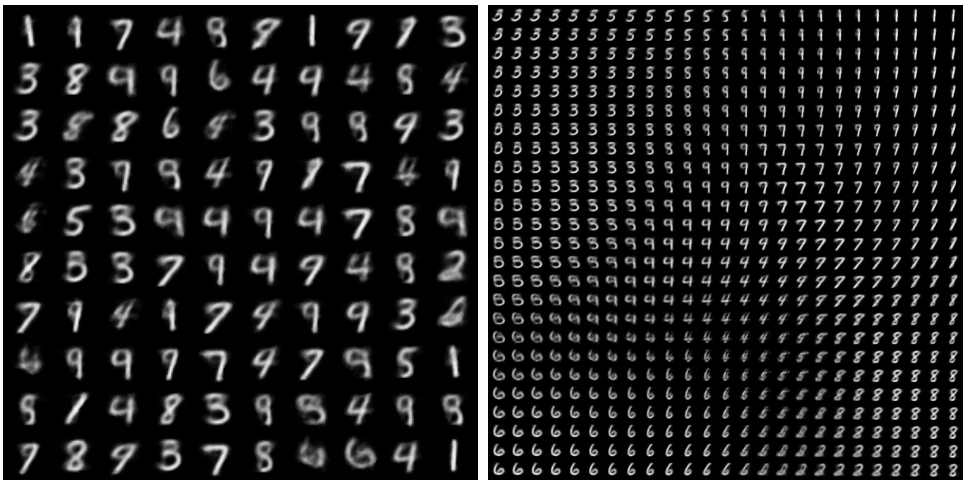
```
CNNDecoder(
  (activation): ELU(alpha=1.0)
  (fc): Linear(in_features=2, out_features=512, bias=True)
  (conv_transpose_layers): Sequential(
    (0): ConvTranspose2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ELU(alpha=1.0)
    (6): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ELU(alpha=1.0)
    (9): Conv2d(32, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
)
```

(b) CNN Decoder

3.3 Result

The training result using MLP architecture on MNIST dataset for latent dimension 2 is as follows: The final training reconstruction loss is 293, and KL divergence is 14. We can observe that there exists obvious interpolation property on latent space, and different classes cluster together, which satisfies our requirements on the latent space discussed in the beginning.

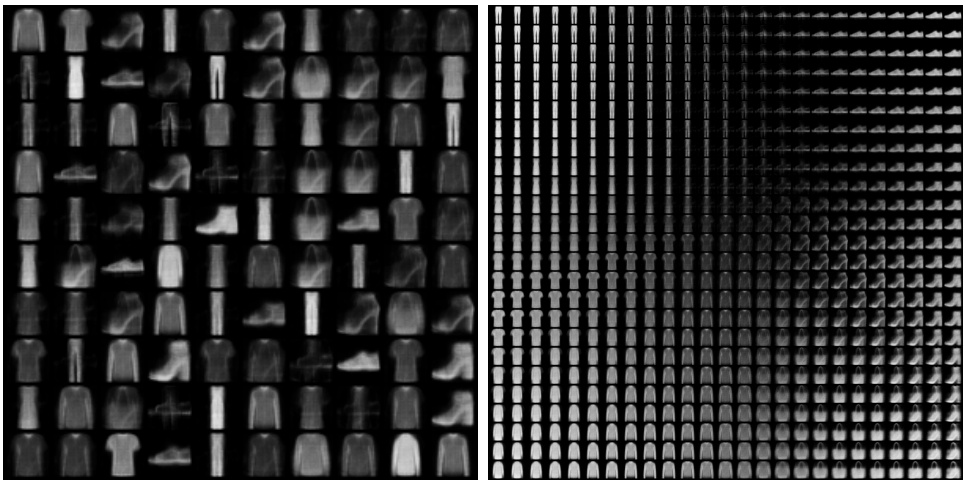
Figure 5: Generation using MLP architecture on MNIST (final loss: 307)



(a) Samples from latent space (b) The structure of 2d latent space

The training result using Fashion MNIST is as follows. The final training loss is 229, where reconstruction term is 216 and KL divergence term is 13.4.

Figure 6: Generation using MLP architecture on Fashion MNIST (final loss: 229)



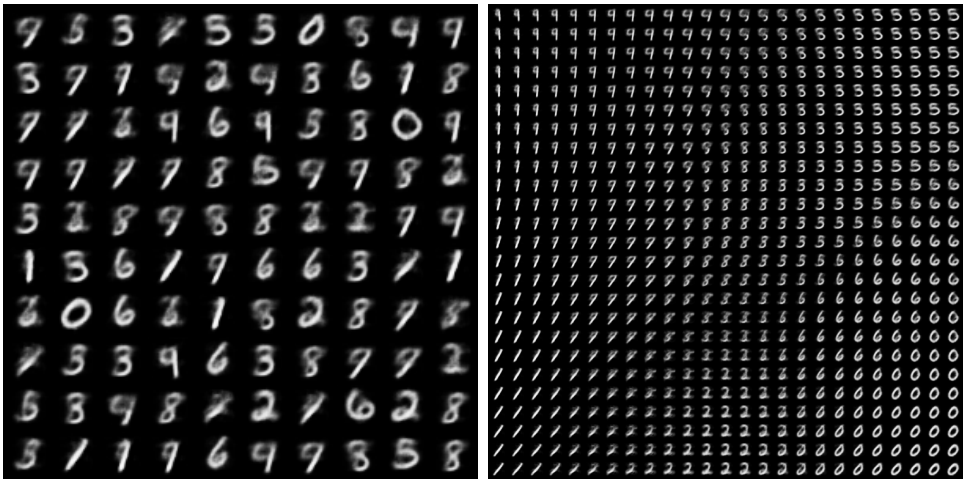
(a) Samples from latent space (b) The structure of 2d latent space

4 Comparative Analysis

4.1 MLP-based vs CNN-based Architectures

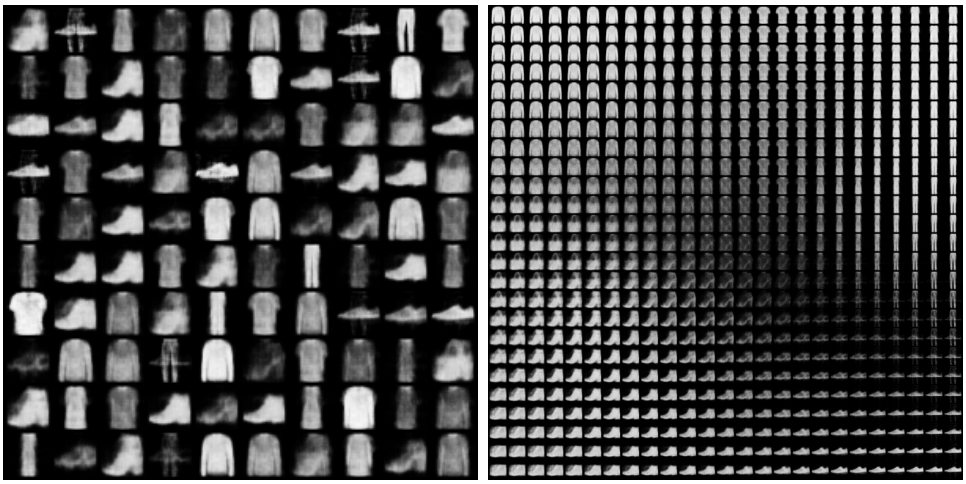
The training result using CNN architecture on MNIST dataset for latent dimension 2 is as follows. The generation quality is worse than that of MLP architecture, and the final training loss is greater than MLP. However, as we will see later, as latent dimension increases, CNN performs significantly better.

Figure 7: Generation using CNN architecture on MNIST (final loss: 322)



(a) Samples from latent space (b) The structure of 2d latent space

Figure 8: Generation using CNN architecture on Fashion MNIST (final loss: 246)



(a) Samples from latent space (b) The structure of 2d latent space

4.2 Latent Dimensions

Latent dimensions dictate the capacity of the latent space. In essence, a higher dimensionality means the latent space can capture more complex variations and nuances of the data. As we increase the latent dimension, the final convergence loss is decreasing, however, the generated result is not always being better. When $z = 1$, the latent space is too simple and the model cannot capture the structure of the data, so the generated samples are not diverse and less authentic. When $z = 3$, the model properly captures the underlying structure. In this case, different classes of data are clearly separable, and the generated samples are diverse and real. When $z = 20$, the loss is less than that of $z = 3$, but the generation result is bad. This is because the latent space is sparse: the KL divergence is inevitably higher as the latent dimension increases, and some spaces are not properly learned. When we randomly sample according to a standard normal distribution, there is less probability to sample near a cluster center.

Figure 9: Generation using different latent dimensions (MLP)

(a) z=1, loss=365 (b) z=2, loss=302 (c) z=3, loss=258 (d) z=5, loss=204 (e) z=20, loss=154

The CNN Encoder-Decoder generally performs worse than MLP in low latent dimensions, however, CNN tends to be more robust and effective in high lent dimensions.

Figure 10: Generation using different latent dimensions (CNN)

(a) z=1, loss=399 (b) z=2, loss=336 (c) z=3, loss=284 (d) z=5, loss=218 (e) z=20, loss=133

4.3 Balance between two losses

In the following, models are trained on different c, which determines the balance between the reconstruction loss and KL divergence loss. Other factors remains the same, using $z = 2$,

and MLP architecture. As shown below, when $c = 0.0001$, the model focuses more on reconstruction rather than regularization. As a result, the generated result is less diverse because some cluster means are far away from the origin. When $c = 1$, the model focuses too much on KL divergence and less on reconstruction, so the generated samples are diverse but less authentic.

Figure 11: Generation using different c

(a) c = 0.0001 (b) c = 0.001 (c) c = 0.05 (d) c = 0.1 (e) c = 1

5 Conclusion

VAE is the basis for understanding diffusion model because diffusion model is just a specific version of hierarchical VAE, so I spent enough time understanding the idea and mathematics behind VAE. The next step I plan to understand VQ-VAE, which is crucial for understanding stable diffusion. By learning VAE, I gained a deeper understanding on latent models, the moral of ELBO and how different architectures and latent spaces affect the performance of a model.