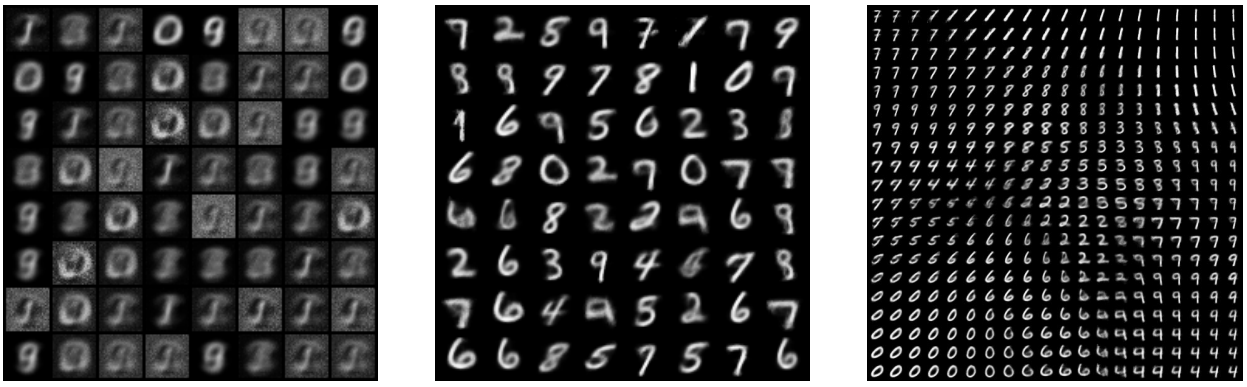
Project 3: Generative Model

1 Introduction

Variational Autoencoders (VAEs) are a type of generative model useful for tasks such as generating new data or encoding data into a lower-dimensional space. The concept behind VAEs involves learning a data representation in a latent space, and generating new data by sampling from this space. They model data x as being generated by a random process involving an underlying latent variable z. The model is trained by maximizing the likelihood of the data given the latent variable, with an added regularization term to enforce smoothness in the latent distribution.



(a) Trained for one epoch (b) Trained for many epochs (c) Learned 2D manifold

Figure 1: Visualizations of samples from Variational Autoencoders

In this assignment, you will first build a simple autoencoder for images. Subsequently, you will progressively build your VAE from the base autoencoder model. We will be using the MNIST dataset of carefully cropped images of hand-written digits. Each image is a 28 x 28 binary image.

The code file `vae.py` contains all the parts that need to be completed, marked as TODO.

2 Detailed Requirements and Points

2.1 Task A: Train a Simple Autoencoder (30 points)

2.1.1 Task A.1: Build the Autoencoder (20 points)

Firstly, we will build a simple autoencoder for images. The architecture of encoder part is: 784(input space) $\rightarrow$ $fc \rightarrow ReLU \rightarrow 512 \rightarrow fc \rightarrow 2$(latent space). The architecture of decoder part is: 2(latent space) $\rightarrow fc \rightarrow ReLU \rightarrow 512 \rightarrow fc \rightarrow Sigmoid \rightarrow$ 784(output space). where fc is fully-connected layer, ReLU and Sigmoid are activation functions. 784, 512 and 2 are the dimensions of the layers.

Your Task: Construct the encoder and decoder layers in `__init__` function of the `Autoencoder` class, and finish the `encode` and `decode` function according to the docstrings.

2.1.2 Task A.2: Train the Autoencoder (10 points)

Perform training for 100 epochs. The results will be saved to `results/AE-arch=ae(512)-lr=0.001-ep=100`. Plot the train & test loss curve (data are in perf_metrics.csv) and attach the reconstructed samples (reconstruction_99.png) and generated samples (sample_99.png).

2.2 Task B: Build your VAE (50 points)

In this task, we will extend the simple autoencoder(AE) to a Variational Autoencoder (VAE).

2.2.1 Task B.1: Derive the Loss Function for VAE(25 points)

The key difference between VAE and AE is their different representation of latent space and output space. In AE, both the latent space and output space are **deterministic**, while in VAE, both the latent space and output space are modeled as **distributions**. However, how can a neural network outputs a (continuous) distribution? In fact, a neural network cannot directly model a continuous distribution. We must **first choose a distribution family**, and then model the **parameters** of the distribution, which is what VAE does. Denote the input as $\mathbf{x}$, when training a VAE for MNIST digit classification, the encoder q_ϕ models the latent space as follows:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_\phi, \boldsymbol{\Sigma}_\phi), \tag{1}$$

and decoder p_θ models the output space as follows:

$$p_\theta(\mathbf{x}|\mathbf{z}) = \text{Bernoulli}(\mathbf{x}; \boldsymbol{\lambda}_\theta). \tag{2}$$

Your task: The ELBO (Evidence Lower BOund) function is defined as:

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||\mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]. \tag{3}$$

Your task is to derive the estimator of ELBO, which is:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) \simeq -\frac{1}{2}\left[\boldsymbol{\mu}_\phi^T \boldsymbol{\mu}_\phi + \text{tr}(\boldsymbol{\Sigma}_\phi) - \log|\boldsymbol{\Sigma}_\phi| - n\right] + \frac{1}{L}\sum_{i=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}^{(l)}), \tag{4}$$

where $|\cdot|$ is the determinant of a matrix, n is the dimension of the latent space, and $\mathbf{z}^{(l)} \sim \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_\phi, \boldsymbol{\Sigma}_\phi)$. The second term is basically the Monte Carlo estimator of $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]$[1].

We would like to maximize ELBO, this is equivalent to minimize the negative ELBO, thus giving our loss function:

$$Loss = \frac{1}{2}\left[\boldsymbol{\mu}_\phi^T \boldsymbol{\mu}_\phi + \text{tr}(\boldsymbol{\Sigma}_\phi) - \log|\boldsymbol{\Sigma}_\phi| - n\right] - \frac{1}{L}\sum_{i=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}^{(l)}), \tag{5}$$

Hint: the KL divergence between two multivariate gaussians is:

$$D_{KL}\big(\mathcal{N}(\mu_\mathbf{1}, \boldsymbol{\Sigma}_\mathbf{1})||\mathcal{N}(\mu_\mathbf{2}, \boldsymbol{\Sigma}_\mathbf{2})\big) = \frac{1}{2}\left[(\mu_\mathbf{2} - \mu_\mathbf{1})^T {\boldsymbol{\Sigma}_\mathbf{2}}^{-1}(\mu_\mathbf{2} - \mu_\mathbf{1}) + \text{tr}({\boldsymbol{\Sigma}_\mathbf{2}}^{-1}\boldsymbol{\Sigma}_\mathbf{1}) - \log\frac{|\boldsymbol{\Sigma}_\mathbf{1}|}{|\boldsymbol{\Sigma}_\mathbf{2}|} - n\right]. \tag{6}$$

2.2.2 Task B.2: Train Your VAE (25 points)

For code implementation, we should make some simplifications:

1. We assume the covariance matrix $\boldsymbol{\Sigma}_\phi$ to be diagonal, i.e., $\boldsymbol{\Sigma}_\phi = \text{diag}(\boldsymbol{\sigma}_\phi^2)$.
2. We make the neural network to model the log-variance (`logvar` in code) $\log \boldsymbol{\sigma}^2$ instead of $\boldsymbol{\sigma}^2$, so we do not need to worry about the negative values.

[1] This finally reduces to a binary cross entropy loss, which is the same as the autoencoder's loss. Think why is that?

3. We simplify the Monte Carlo estimator to only one sample, i.e., $L = 1$.

Your task:

1. Implement the VAE part in the `__init__` function and the `encode` function.

Hint: you only need to add one fc layer ($512 \to 2$) to the encoder network to model the `logvar`), and use `autoencoder_type` to control whether to use VAE or AE. When running the program, you may pass `--autoencoder_type vae` to specify the VAE model.

2. Implement the KL divergence part in the `loss_function` function.

Hint: under our simplifications, $\log|\mathbf{\Sigma}_\phi|$ can be computed by `torch.sum(logvar)`, and $\mathrm{tr}(\mathbf{\Sigma}_\phi)$ can be computed by `torch.sum(logvar.exp())`)

3. Perform training for 100 epochs. The results will be saved to `results/AE-arch=vae(512)-lr=0.001-ep=100`. Plot the train & test loss curve (data are in perf_metrics.csv) and attach the reconstructed samples (reconstruction_99.png) and generated samples (sample_99.png).

2.3 Task C: Visualizing the Latent Space (20 points)

In this task, you will gain a deeper understanding of the latent space of AEs and VAEs by visualizing it.

Your task:

1. Attach the manifold visualization results (manifold_digits.png) and the scatter plot of the latent space (manifold.png) for both VAE and AE.

2. Explain what the `_set_latent_vectors` does in plot_manifold_learning_result function.

3. Compare the latent space of AE and VAE and the generated samples. What do you observe?

2.4 Bonus Task: VAE for Face Generation (20 points)

In this task, you will implement a VAE for face generation. The dataset is the CelebA dataset, which contains 202,999 face images of celebrities. You can download the dataset from `http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`.

Your task: 1. Sample a subset of 50000 images to boost your training speed. You can use the `torch.utils.data.Subset` class to do this.

2. Adopt the CelebA dataset in `vae.py` and train the VAE for 100 epochs.

3. Attach the generated samples (sample_99.png) and the train & test loss curve in your report.