Pattern Recognition and Machine Learning: Project 3, Zhengzuo Liu

1 Introduction

2 Detailed Requirements and Points

2.1 Task A: Train a Simple Autoencoder

2.1.1 Task A.1: Build the Autoencoder

Completed codes are as follows:

1) Encoder and decoder layers in ___init___ function of the "Autoencoder" class.

```
self.encoder = nn.Sequential(
    nn.Linear(784,512),
    nn.ReLU(),
    nn.Linear(512,2)
)

self.decoder = nn.Sequential(
    nn.Linear(2,512),
    nn.ReLU(),
    nn.Linear(512,784),
    nn.Sigmoid()
)
```

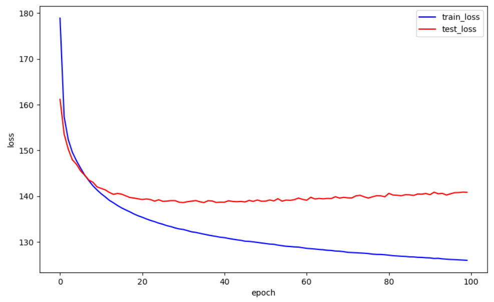
2) Encode and decode function.

```
if self.autoencoder_type == "ae":
    return self.encoder(x)
```

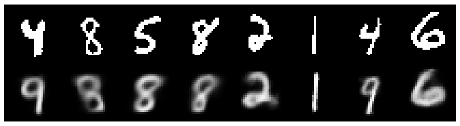
```
return self.decoder(z)
```

2.1.2 Task A.2: Train the Autoencoder

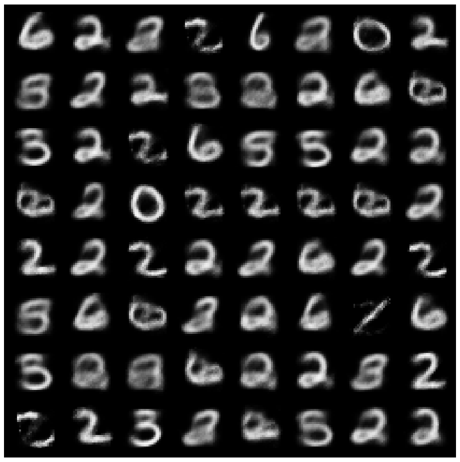
Train & test curve:



Reconstructed sample:



Generated sample:



2.2 Task B: Build your VAE

2.2.1 Task B.1: Derive the Loss Function for VAE

Let $D_{KL}(q_\phi(z|x) \| \mathcal{N}(z;0,I)) \simeq A$, $-E_{q_\phi(z|x)}[\log p_\theta(x|z)] \simeq B$.

Then $\mathcal{L}(\theta,\phi;x) \simeq -A-B$, $Loss = A+B$.

$$\begin{aligned} A &= D_{KL}(\mathcal{N}(z;\mu_\phi,\Sigma_\phi)) \| \mathcal{N}(z;0,I)) \\ &= \frac{1}{2}\left[(0-\mu_\phi)^T I^{-1}(0-\mu_\phi) + tr(I^{-1}\Sigma_\phi) - \log\frac{|\Sigma_\phi|}{|I|} - n\right] \\ &= \frac{1}{2}\left[\mu_\phi^T\mu_\phi + tr(\Sigma_\phi) - \log|\Sigma_\phi| - n\right]. \end{aligned}$$

$B = \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x|z^{(l)})$, where $z^{(l)} \sim q_\phi(z|x)$. L is the number of MC samples.

Therefore.

$$\mathcal{L}(\theta,\phi;x) \simeq -\frac{1}{2}\left[\mu_\phi^T\mu_\phi + tr(\Sigma_\phi) - \log|\Sigma_\phi| - n\right] + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x|z^{(l)})$$

$$Loss = \frac{1}{2}\left[\mu_\phi^T\mu_\phi + tr(\Sigma_\phi) - \log|\Sigma_\phi| - n\right] - \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x|z^{(l)})$$

2.2.2 Task B.2: Train Your VAE

Completed codes are as follows:

1) Modified encoder layer.

```
self.encoder_shared = nn.Sequential(
    nn.Linear(784,512),
    nn.ReLU(),
)
self.encoder_mu = nn.Linear(512,2)
self.encoder_logvar = nn.Linear(512,2)
```

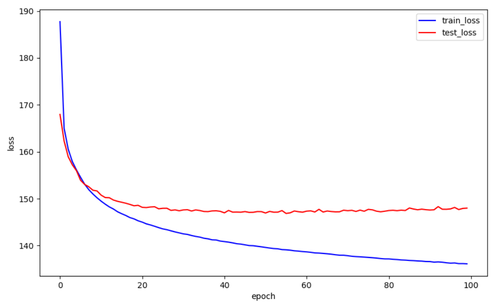
2) Modified encode function.

```
share_encoded = self.encoder_shared(x)
if self.autoencoder_type == "ae":
    return self.encoder_mu(share_encoded)
elif self.autoencoder_type == "vae":
    mu = self.encoder_mu(share_encoded)
    logvar = self.encoder_logvar(share_encoded)
    return mu, logvar
```

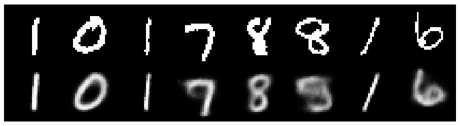
3) KL divergence part in loss_function function

```
share_encoded = self.encoder_shared(x)
if self.autoencoder_type == "ae":
    return self.encoder_mu(share_encoded)
elif self.autoencoder_type == "vae":
    mu = self.encoder_mu(share_encoded)
    logvar = self.encoder_logvar(share_encoded)
    return mu, logvar
```

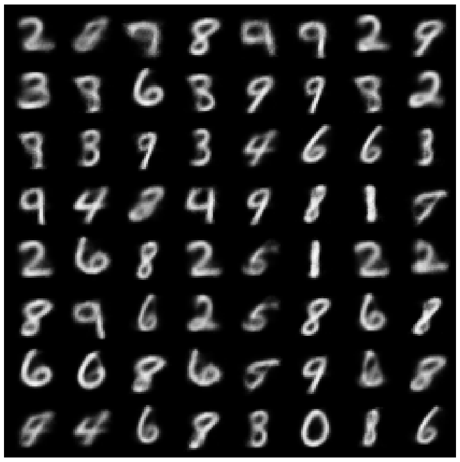
The results are as follows:
Train & test curve:



Reconstructed sample:



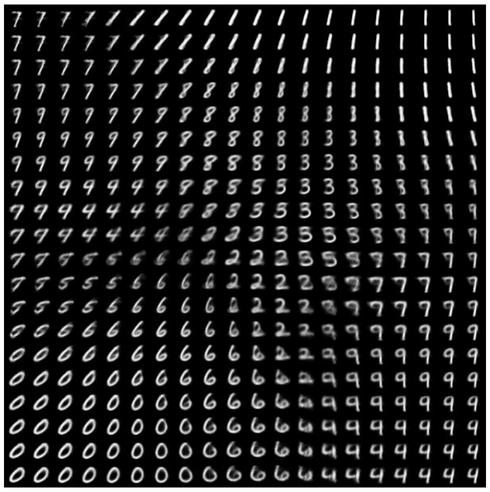
Generated sample:



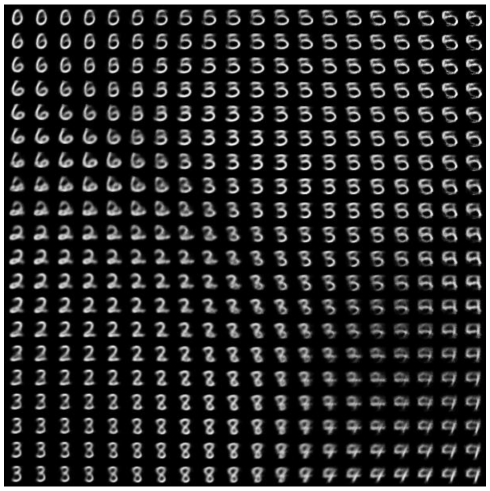
2.3 Task C: Visualizing the Latent Space

Manifold visualization results:

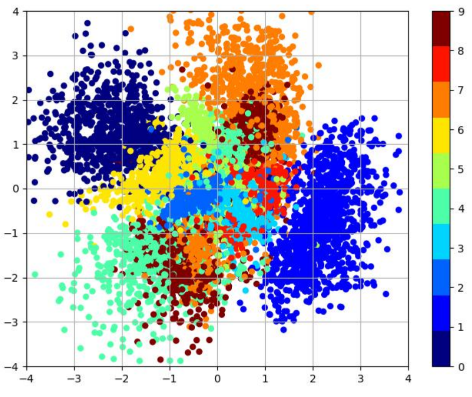
VAE:



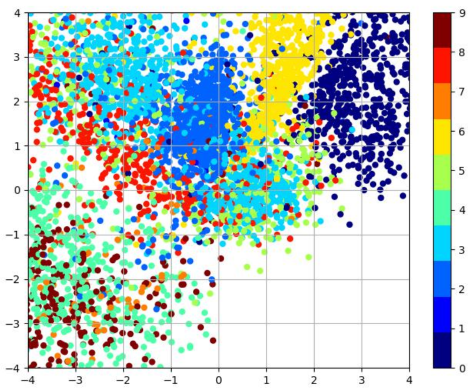
AE:



Scatter plot:
VAE:



AE:



The ‘_set_latent_vectors()‘ function is used to create a grid of points in the latent space of the Variational Autoencoder (VAE). This grid is defined in a two-dimensional space, since the latent vectors in the VAE have been assumed to be 2-dimensional (‘model.n_dims_code == 2‘).

When this grid of points is passed through the decoder of the VAE, the decoder generates an image for each point in the grid. These images can be viewed as a visualization of how the VAE ”imagines” the data space. It’s a way of exploring what the VAE has learned about the data.

The latent space of VAE is more clearly divided than that of AE, that is, points of different colors from different digits are better clustered in there own groups, and better separated from other groups. The generated examples of VAE also include less ”meaningless” digits than AE.