Mahmut Alp Erkent
2019742009

# CMPE 597 Sp. Tp. Deep Learning
# Spring 2021 Project III

## Answers

- **(20 pts) Implement a single layer LSTM as the encoder of the VAE. You may use the LSTM functions provided by the deep learning library. You will determine the dimensionality of the hidden states.**

  I used PyTorch library to implement a single layer LSTM. I chose the number of hidden states to be 128, because I believed this would give me enough dimensionality (64 after sampling) to input for transpose convolutional layers.

- **(20 pts) Implement a convolutional decoder with transpose convolutional layers. This decoder should take a random vector sampled from the distribution that was outputted by the encoder and decodes it to an 28x28 grayscale image. Since you will go to an image from a random vector, you will need transpose convolution. You will decide the number of layers, kernel size, number of filters and the stride value. You are free to add dropout or batch normalization if they are necessary.**

  I tested two different implementations for sampling. First implementation was with two fully-connected layers, one for the mean and one for the variance of the distribution, both taking the 128-dimensional output of LSTM as input and outputting 64 features. As an alternative, I tried sample directly from the LSTM output, taking half of it as the mean and the other half as variance.

  Then, as I implemented the decoder with transpose convolutional layers, I tried many different combinations of architectures with 4, 5, 6 or 8 layers, with ReLU or LeakyReLU with different slopes, and with batchnorm or not.

  Here are the performances of different combinations:

| **4 layer decoder \| leaky(0.01) \| bn- \| FC+:** | | **4 layer decoder \| leaky(0.01) \| bn- \| FC-:** | |
|---|---|---|---|
| Training total loss: | 118.945686 | Training total loss: | 132.336533 |
| Training reconstruction loss: | 91.368050 | Training reconstruction loss: | 108.537201 |
| Training KLD loss: | 27.577604 | Training KLD loss: | 23.799326 |
| Validation total loss: | 117.775909 | Validation total loss: | 130.883347 |
| Validation reconstruction loss: | 89.884445 | Validation reconstruction loss: | 107.122009 |
| Validation KLD loss: | 27.891415 | Validation KLD loss: | 23.761320 |
| **4 layer decoder \| leaky(0.01) \| bn+ \| FC+:** | | **4 layer decoder \| leaky(0.2) \| bn+ \| FC+:** | |
| Training total loss: | 122.204071 | Training total loss: | 121.583725 |
| Training reconstruction loss: | 94.603416 | Training reconstruction loss: | 93.191528 |
| Training KLD loss: | 27.600563 | Training KLD loss: | 28.392277 |
| Validation total loss: | 123.567062 | Validation total loss: | 124.044975 |
| Validation reconstruction loss: | 96.111328 | Validation reconstruction loss: | 95.421844 |
| Validation KLD loss: | 27.455740 | Validation KLD loss: | 28.623117 |
| **4 layer decoder \| leaky(0.2) \| bn- \| FC+:** | | **4 layer decoder \| relu \| bn+ \| FC+:** | |
| Training total loss: | 122.732857 | Training total loss: | 123.085167 |
| Training reconstruction loss: | 93.825417 | Training reconstruction loss: | 95.735008 |
| Training KLD loss: | 28.907507 | Training KLD loss: | 27.350178 |
| Validation total loss: | 121.664612 | Validation total loss: | 125.285271 |
| Validation reconstruction loss: | 92.834892 | Validation reconstruction loss: | 97.902130 |
| Validation KLD loss: | 28.829672 | Validation KLD loss: | 27.383146 |

| **4 layer decoder \| relu \| bn- \| FC+:** | | **5 layer decoder \| leaky(0.01) \| bn- \| FC+:** | |
|---|---|---|---|
| Training total loss: | 120.019218 | Training total loss: | 115.692749 |
| Training reconstruction loss: | 93.494423 | Training reconstruction loss: | 88.207474 |
| Training KLD loss: | 26.524683 | Training KLD loss: | 27.485247 |
| Validation total loss: | 119.368660 | Validation total loss: | 114.255844 |
| Validation reconstruction loss: | 92.944504 | Validation reconstruction loss: | 86.700783 |
| Validation KLD loss: | 26.424137 | Validation KLD loss: | 27.555090 |
| **6 layer decoder \| leaky(0.01) \| bn- \| FC+:** | | **8 layer decoder \| leaky(0.01) \| bn- \| FC+:** | |
| Training total loss: | 113.717278 | Training total loss: | 116.174988 |
| Training reconstruction loss: | 86.138268 | Training reconstruction loss: | 89.589226 |
| Training KLD loss: | 27.578886 | Training KLD loss: | 26.585693 |
| Validation total loss: | 112.770195 | Validation total loss: | 115.182693 |
| Validation reconstruction loss: | 85.135307 | Validation reconstruction loss: | 88.377838 |
| Validation KLD loss: | 27.634918 | Validation KLD loss: | 26.804855 |

In the end, I chose to use LeakyReLU and fully-connected sampling layers with 6-layer decoder and without batch normalization, because this combination gave the best performance. Here are the kernel sizes and the feature numbers in the decoder layers:

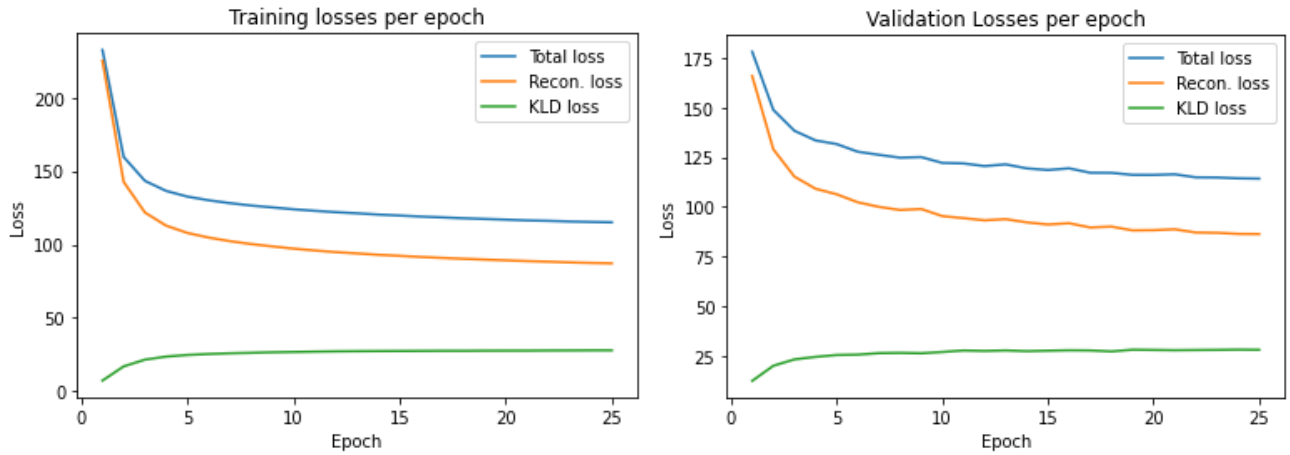
```
self.conv1 = nn.ConvTranspose2d(16, 13, 9)   # (2, 2)   -> (10, 10)
self.conv2 = nn.ConvTranspose2d(13, 10, 7)   # (10, 10) -> (16, 16)
self.conv3 = nn.ConvTranspose2d(10, 7, 5)    # (16, 16) -> (20, 20)
self.conv4 = nn.ConvTranspose2d(7, 4, 5)     # (20, 20) -> (24, 24)
self.conv5 = nn.ConvTranspose2d(4, 2, 3)     # (24, 24) -> (26, 26)
self.conv6 = nn.ConvTranspose2d(2, 1, 3)     # (26, 26) -> (28, 28)
```

- **(20 pts) Implement the reconstruction loss with binary cross-entropy and the regularization term with KL-divergence. You may check the closed form of KL divergence between two univariate Gaussians (you may come across implementations using the closed form on the internet) or you may use a built-in function.**

I used PyTorch's built-in binary cross entropy function for reconstruction loss. I looked up the closed form of KL divergence between two univariate Gaussians[1,2]. Here is the code for the loss calculation:

```
recon_loss = F.binary_cross_entropy(outputs, inputs, reduction='sum')
kld_loss = -0.5 * torch.sum(1 + log_var - mu ** 2 - log_var.exp())
loss = recon_loss + kld_loss
```

- **(15 pts) Train your model and plot the values of the loss function and the regularization term during training. Comment on their behaviors.**



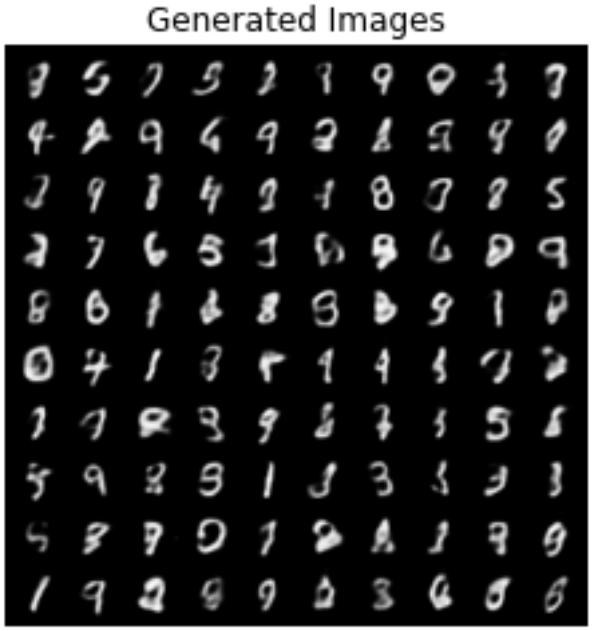

I used the test MNIST data as validation since VAE is an unsupervised learning technique. Value of the regularization term increased as reconstruction term and total loss decreased over epochs. Since the

initialization was very close to normal distribution, regularization term started very small, but it naturally increased as the model started to fit to training data. On the other hand, the reconstruction term started decreased as the model learned to represent the digits in its latent space.

- **(5 pts) Save the trained model and include it in your submission folder. If you don't provide the saved model, 5 points will be deducted.**

  See “model.pk” file.

- **(15 pts) Load the trained decoder and generate 100 images from 100 random vectors. Visualize the generated samples. Comment on your performance.**




  I believe that overall, the system worked well. Many of the images resembled a digit, while there were some images that were missing only a portion from a distinguishable digit. On the other hand, some images were just some unidentifiable scribbles, possibly due to the combination of some unique outlier inputs that created areas in the latent space that were not specific to a certain digit.

- **(15 pts) Provide a README file including the information about how to train your VAE and how to generate samples. If you don't provide a README, 5 points will be deducted.**

  See “README.txt” file.

## References


1. *KL divergence between two univariate Gaussians*. (2021, Jun 15). Stack Exchange. https://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians
2. Krishnamoorthy, A. (2021, Jun 15). *Vanilla VAE*. GitHub. https://github.com/AntixK/PyTorch-VAE/blob/master/models/vanilla_vae.py