CS552: Generative AI, Homework 2
Will Buchta
2/12/2025

Problem 1

a) For my model architecture, I chose the encoder to have three convolutional layers with batch norm between them. The latent encoded feature map is then flattened and passed through a fully connected layer to 32 dimensions. The decoder is then the inverse of the encoder without batch norm (Figure 1). After training, I generated new faces via "z = torch.randn(num_samples, model.latent_dim).to(device)" (Figure 2).

```
VAE(
  (encoder): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (7): ReLU()
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (fc_mu): Linear(in_features=1152, out_features=32, bias=True)
  (fc_logvar): Linear(in_features=1152, out_features=32, bias=True)
  (fc_dec): Linear(in_features=32, out_features=1152, bias=True)
  (decoder): Sequential(
    (0): ConvTranspose2d(128, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (1): ReLU()
    (2): ConvTranspose2d(64, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (3): ReLU()
    (4): ConvTranspose2d(32, 1, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
    (5): Sigmoid()
  )
)
```

Figure 1: VAE Architecture

Figure 2: Randomly Generated Faces from VAE

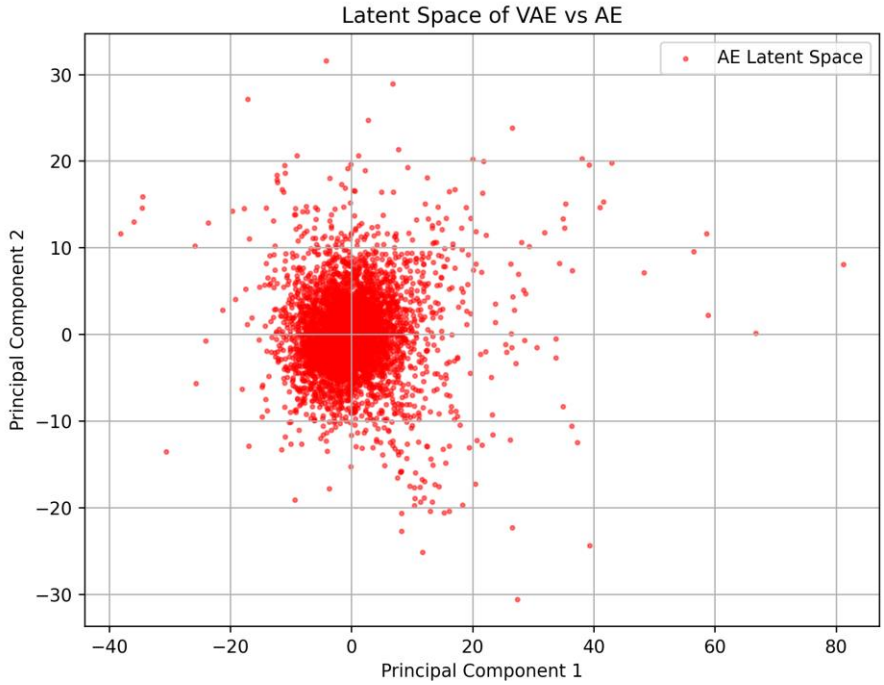
b) I created a new class, AE, that has the same architecture as the VAE from part (a), only without the mu and logvar fully connected layers. I then trained it like a normal autoencoder. When reconstructing faces (Figure 3), the quality is not bad. However, when generating new faces by randomly sampling z (Figure 4), same as in part (a), the results were quite disturbing.

Figure 3: Reconstructed Faces

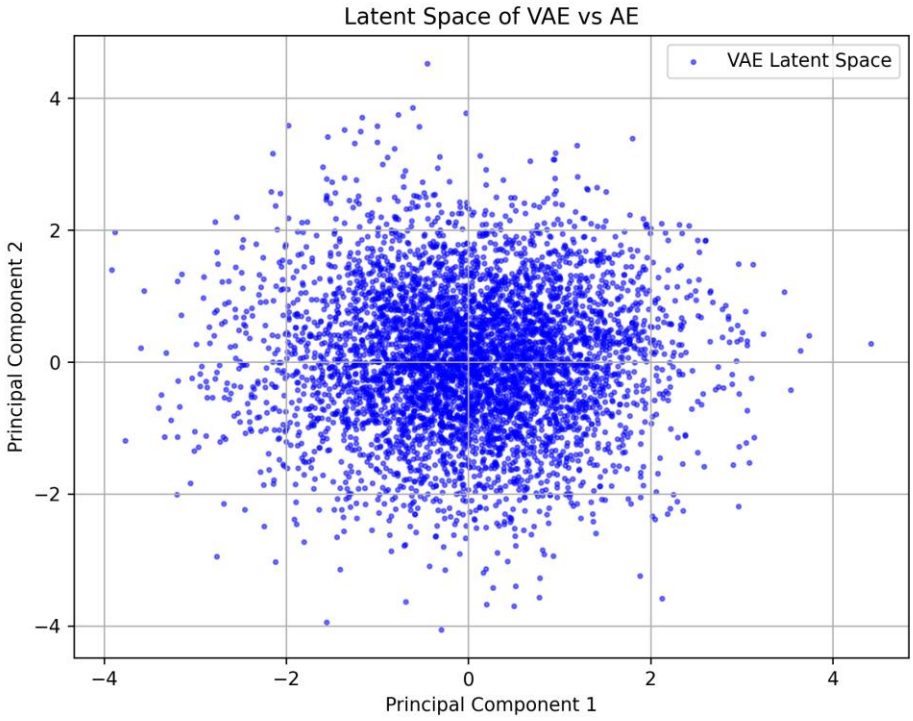
Figure 4: Randomly Sampled Latent Vectors After Decoder

c) Comparing latent spaces of AE and VAE by using PCA. Note the magnitude of the scales. The latent space of the variational autoencoder represents a standard normal distribution much nicer. In fact, when plotting a standard normal distribution on top of the VAE's latent space, the two distributions are nearly indistinguishable (see iv). The distribution of the standard autoencoder has a skew with large magnitude outliers, and the principal components are greater in magnitude as well. This is because the KL divergence term strongly trains the variational autoencoder to map to a standard normal.

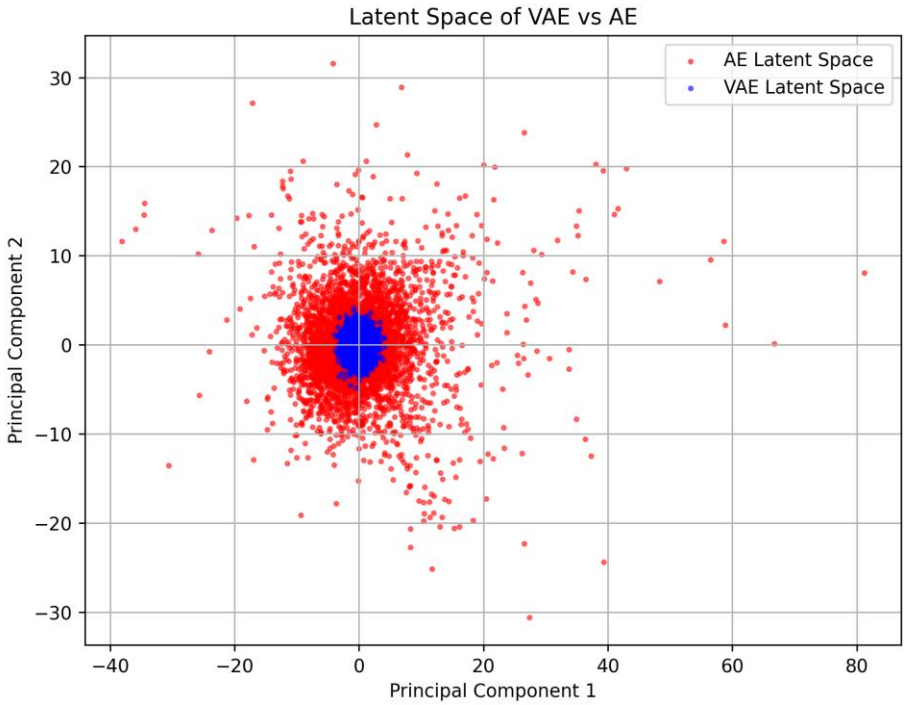
i. Latent space of AE:



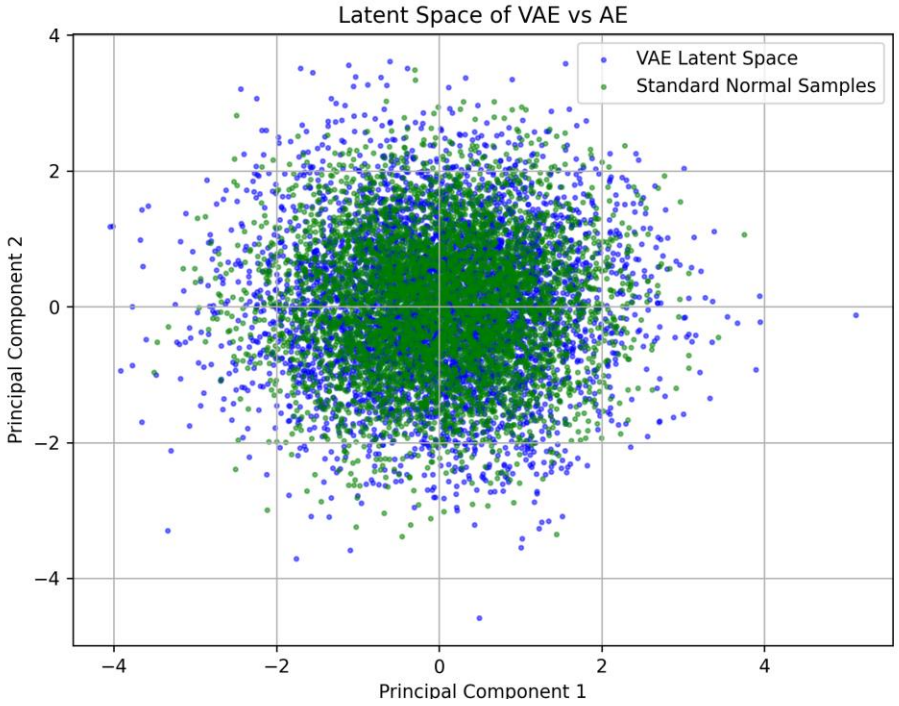
ii. Latent space of VAE:



iii. Overlapped:



iv. Standard normal plotted on top of VAE latent space



d) Figure 5 below shows two different randomly sampled latent vectors that were decoded to faces (top left and bottom right), along with the interpolation of their latent features. As

seen, the faces slowly start to transition into looking like the other one; the top left could be a pale skinned woman while the bottom right is a darker skinned male.

Figure 5: Interpolated Faces

e) See problem_1e() in train_vae.py. I first trained PPCA on the entire dataset with the same latent dimension as the VAE for a fair comparison, then saved the weights for future use as it took a non-zero amount of time to train. I then trained PCA on the X face dataset, generated 20 random vectors, and projected them using the VAE and PPCA. When plotting only 20 datapoints, it is hard to make out the distribution of samples (Figure 6), but when plotting more (200, Figure 7), the distribution is clear. Looking at the VAE's distribution, it's easy to claim that the standard deviation is close to 1, mapping a standard normal very nicely. The PPCA's distribution also matches a gaussian distribution, just not with a standard deviation of 1. PPCA is linear and assumes a zero-mean, while the VAE is non-linear and can accommodate shifts in the mean.

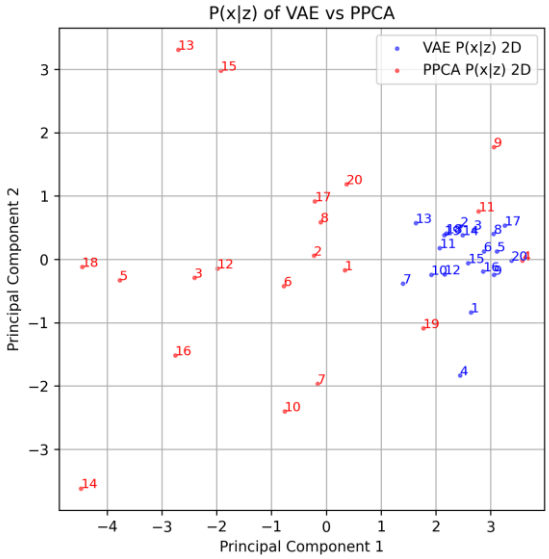


Figure 6: Generated Faces Projected to 2D. 20 Samples

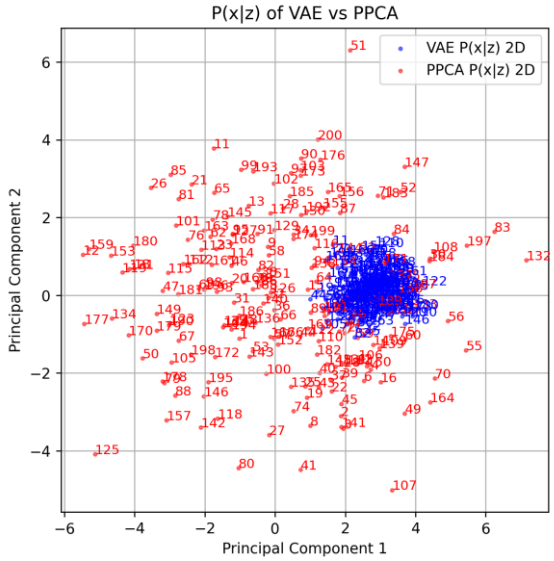


Figure 7: Generated Faces Projected to 2D. 200 Samples

Problem 2

For this problem, I made a simple VAE that predicts a scalar: 0.8. To create a dataset, I sampled from a normal distribution centered around 0.8 with a very small standard deviation. After training, I then chose a sample from the dataset and created the graph below (Figure 8). Black represents the true log likelihood of the sample versus tweaks in the decoder's parameters, and each phi diff is the ELBO computed for small changes in one of the encoder's parameters. It was challenging to get good results for larger changes in the encoder's weight as the ELBO became very noisy. For each theta diff, I sampled the ELBO 200 times. See problem2.py for my implementation.

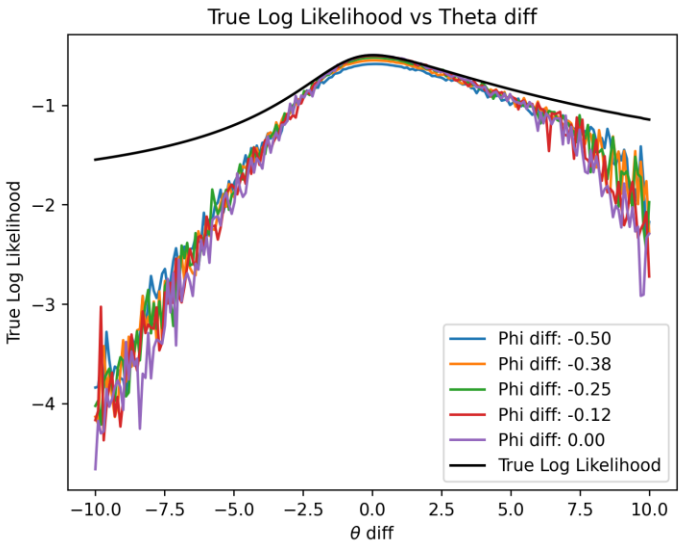


Figure 8: Elbo vs True Log Likelihood