Deep Learning Assignment 3

Nils Lehmann
12868175
nils.lehmann@student.uva.nl

1 Variational Auto Encoders

1.1

In order to obtain samples from our VAE model, we first sample a z vector from our latent distribution, which in our case is a Normal distribution. Subsequently, we use the decoder neural network to obtain an image representation that holds Bernoulli means. Then, we can use those per pixel means to sample from a Bernoulli distribution to obtain a final binary image. Thus, we first sample from our latent distribution and then obtain a sample conditioned on that latent distribution.

1.2

Monte-Carlo integration is inefficient because it would require a large number of samples per single data point and have a high variance. In a high dimensional space there can be vast areas with no significant probability mass and hence we need a large number of samples to obtain a reasonable estimate.

1.3

Since, KL-divergence measures the dissimilarity of probability distributions, a very small divergence value would for example occur when univariate gaussians q and p have a mean that is slightly different with the same standard deviation. So for example q = $\mathcal{N}(0, 1)$ and p = $\mathcal{N}(0.01, 1)$. On the other hand two distributions that have very different means, will yield a very large KL-divergence. For example, q = $\mathcal{N}(0, 1)$ and p = $\mathcal{N}(100, 1)$.

1.4

We cannot optimize log$p(\mathbf{x_n})$ directly, because computing it involves an intractable integral and Monte-Carlo integration is too expensive. Therefore, we must optimize the lowe-bound instead. In equation 14, we see the answer to the question how large the gap between $\log p(x_n)$ and the ELBO is. Since the left hand side subtracts the KL divergence term from the data likelihood probability, we see that the right hand side has to be smaller than than the term $\log p(x_n)$

1.5

When the lower bound is pushed up, either log$p(\mathbf{x_n})$ increases or $KL(q(Z|\mathbf{x_n})||\mathbf{p}(\mathbf{Z}|\mathbf{x_n}))$ decreases. In the first case we improve the likelihood of our data, and in the second case we minimize the dissimilarity between the true poster $p(Z|\mathbf{x_n})$ and our approximate posterior $q(Z|\mathbf{x_n})$.

1.6

The name reconstruction loss is appropriate because $\mathcal{L}_n^{recon}$ captures how close our reconstructed image from the latent distribution z is to our input image $\mathbf{x_n}$. The name regularization loss is also

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

appropriate because $\mathcal{L}_n^{reg}$ ensures that our approximate posterior distribution q is not too far away from the true poster distribution p(z).

1.7

Using equation 11, with the multivariate case for computing the regularization Loss:

$$\begin{aligned}
D_{KL}(q_\phi(Z|x_n), p_\theta(Z)) &= \log \frac{1}{|diag(\Sigma_q(x))|} + \frac{diag\Sigma_q(x)) + \mu_\phi(\boldsymbol{x})^T \mu_\phi(\boldsymbol{x})}{2} - \frac{1}{2} \\
&= -\log |diag(\Sigma_q(x))| + \frac{diag(\Sigma_q(x)) + \mu_\phi(\boldsymbol{x})^T \mu_\phi(\boldsymbol{x}) - 1}{2} \\
&= -\frac{2}{2} \log |diag(\Sigma_q(x))| + \frac{diag(\Sigma_q(x)) + \mu_\phi(\boldsymbol{x})^T \mu_\phi(\boldsymbol{x}) - 1}{2} \\
&= -\frac{1}{2} \log diag(\Sigma_q(x))^2 + \frac{diag(\Sigma_q(x)) + \mu_\phi(\boldsymbol{x})^T \mu_\phi(\boldsymbol{x}) - 1}{2} \\
&= \frac{diag(\Sigma_q(x)) + \mu_\phi(\boldsymbol{x})^T \mu_\phi(\boldsymbol{x}) - 1 - \log diag(\Sigma_q(x))^2}{2}
\end{aligned}$$

For the reconstruction loss:

$$\begin{aligned}
\mathcal{L}_n^{recon} &= -E_{q_\phi(z|x_n)}[\log p_\theta(x_n|Z)] \\
&= -E_{q_\phi(z|x_n)} \log \prod_{m=1}^{M} Bern(x_n^m | f_\theta(z_n)_m) \\
&= -E_{q_\phi(z|x_n)} \sum_{m=1}^{M} x_n^m \log f_\theta(z_n)_m + (1 - x_n^m) \log f_\theta(z_n)_m
\end{aligned}$$

1.8

The act of sampling is a stochastic and non-differentiable operation, which means that we cannot backpropagate through the module that does this operation. However, the reparameterization trick changes the sampling operation into operations of addition and multiplication by a random ϵ for which we do not need a gradient. Since the mean and standard deviation output from the encoder are now only forwarded with addition and multiplication, we can easily obtain gradients and backpropagate through this module.

1.9

The MLP encoder architecture has an input of size 784, which is the streched out vector of the image and subsequently several hidden layers of varying dimension, with the default value of a single layer with 512 activations. All hidden layers are followed by an ELU activation module. The output layer is twice the size of the chosen latent dimensionality, which in the default case is 20, yielding an output layer of size 40. The first half of the output vector is chosen to be the mean, whereas the second half is the log standard deviation. The decoder architecture is set to be the same as the encoder, however, the input is the latent vector and the output vector is reshaped into an image. Training was conducted with the default parameters: 80 epochs, with a batch size of 128 and a learning rate of 1e-3. Figure 1 shows the training and validation bpd score for the VAE MLP model. The final test set bpd loss came out to be 0.1531.

1.10

Figure 2 shows the sampled images before training, after 10 epochs and after 80 epochs. The largest training improvements already occur in the first 10 epochs. From the beginning of training to 10 epochs, we can notice that the pattern of a white digit on a black background has been learned by the

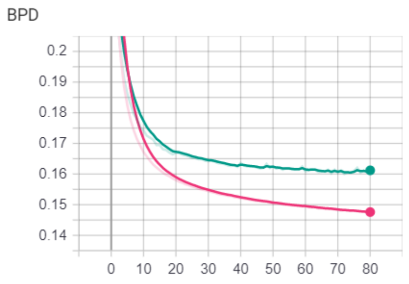

Figure 1: BPD Scores for VAE model with MLP, for training (pink), and validation (green)

VAE. In subsequent training epochs, we can see that some digits become more refined with less noise around them, albeit, the digits after 80 epochs of training are still not very crisp.

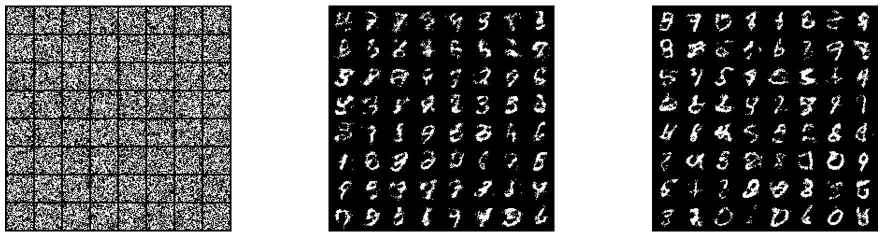
Figure 2: Samples VAE with default parameters before Training, after 10 epochs and after 80 epochs

1.11

Figure 3 shows the manifold of a VAE with two dimensional latent space, where we can observe smooth transitions between all digits in the dataset. With respect to the placement of digits on the manifold, one could say that the corners of the manifold make up some more basic properties of digits such as vertical and horizontal strokes, or loops. However, I am unsure about a specific positioning pattern of the digits.

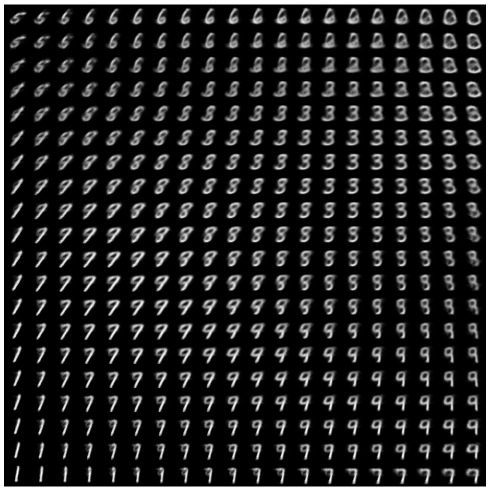
Figure 3: VAE manifold with dimensionality z equal to 2

1.12 Bonus

As an additional experiment, the VAE Encoder and Decoder were modelled with a CNN architecture similar to Tutorial 9. Based on BPD scores, the CNN architecture performs slightly better than the

MLP as Figure 4 shows and has a BPD test score of 0.1325. With respect to generated samples, we can observe in Figure 5 that the samples are also slightly better. For example, the stroke of a "7" is much clearer and the MLP VAE samples seem to exhibit some more noise.

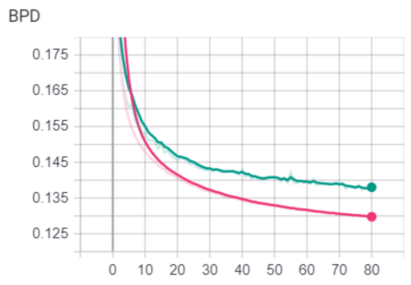

Figure 4: BPD Scores for VAE model with CNN, for training (pink), and validation (green)

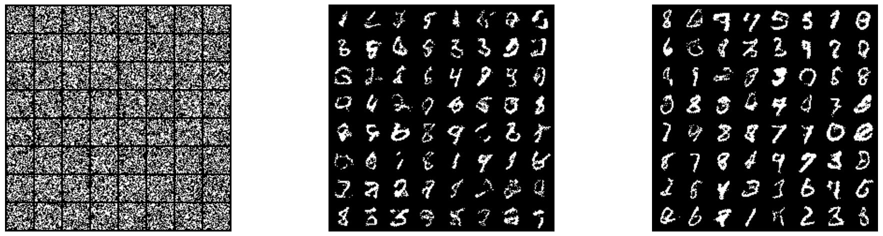

Figure 5: Samples VAE with CNN and default parameters before Training, after 10 epochs and after 80 epochs

2 Generative Adversarial Networks

2.1

2.1.a

The left term of the GAN training objective shows that we want to maximize the expected value of the probability that the Discriminator identifies real data instances as real. The term on the right hand side shows that the Discriminator wants to minimize D(G(z)) to be near zero (correctly identifying the fake), while the Generator wants D(G(z)) to be near 1 (fooling the discriminator). As the Generator cannot affect the term on the left, the loss function from the perspective of the Generator only consists of the right term.

2.1.b

The original paper shows that in theory, V(D,G) would obtain a global maximum, when the probability of the real data is equal to the probability of the generated data (Goodfellow et al. [2014]). This means that the Discriminator cannot distinguish between real and fake images as it just outputs 0.5, instead of something close to 1 (real image) or close to 0 (fake image).

2.2

In the early stages of training, the images that come from the Generator look very different than the training set images and can therefore easily be rejected by the Discriminator. This means, that the gradient signal will not be strong and the Generator does not improve. The solution is to maximize $\log D(G(z))$ instead, which allows for stronger gradients (Goodfellow et al. [2014]).

2.3

The default suggestions and parameters were used for both the Generator and Discriminator. The Generator takes a vector of dimension 32 as input and has five hidden layers with sizes 128, 256, 512, an 1024 respectively. Each hidden layer is followed by a dropout layer with probability 0.1 and a LeakyReLU activation module with alpha 0.2. The output layer has the flattened dimension of the image, so 784 in this case, and a TanH activation because we use normalized images. The Discriminator takes a vector of 784 as input and has two hidden layers of size 512 and 256, which are each followed by a dropout layer with probability of 0.3 and LeakyReLU activation with alpha 0.2. The output of the Discriminator is a single scalar. Training was conducted for 250 epochs with a learning rate of 2e-4 and a batch size of 128. With respect to the generated samples, we can observe in Figure 6 that after 10 epochs, the GAN already shows rough but noisy shapes of digits and understands that it needs to draw white digits on a black background. After 250 epochs we see clearly identifiable digits, that even have some variety in the stroke width and are overall much better than the VAE samples.

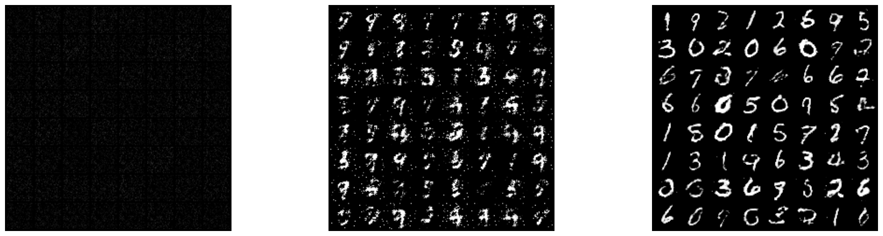

Figure 6: Samples from GAN with default parameters before Training, after 10 epochs and after 250 epochs

2.4

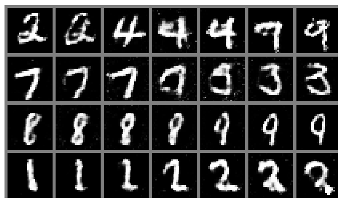

Figure 7: GAN interpolation 4 examples with 5 interpolation steps

The interpolation results can be seen in Figure 7. In the first row, we can observe a "2" being transformed into a "9" via becoming a "4". The other examples also show digit transformations, albeit, there is no clear intermediate digit during the interpolation steps but more a direct transformation from one digit to another. This shows that linear interpolation between two randomly sampled z vectors also leads to smooth transitions in generated images.

2.5

A common problem with the vanilla GAN is instability during training, which could for example be due to low dimensional support, where the distributions of the real and generated data do not overlap and the Jenson Shannon Divergence grows very large. A solution to this problem is using the Wasserstein Distance which yields a meaningful distance metric between distributions even when they are not overlapping.

3 Generative Normalizing Flows

3.1

$$\begin{aligned} p_x(x) &= p_z(z)|\frac{df(x)}{dx}| \\ &= p_z(x^3)|3x^2| \\ &= \frac{1}{b-a}|3x^2| \end{aligned}$$

When integrating the result, we have the relationship that $z = x^3$ and because the Uniform distribution z has support from a to b, we have to change the integration boundaries accordingly. We can drop the absolute value, because x^2 is always positive.

$$\begin{aligned} \int_{-\infty}^{\infty} p_x(x)dx &= \int_{a^{\frac{1}{3}}}^{b^{\frac{1}{3}}} \frac{1}{b-a} 3x^2 dx \\ &= [\frac{1}{b-a}x^3]_{a^{\frac{1}{3}}}^{b^{\frac{1}{3}}} \\ &= \frac{(b^{\frac{1}{3}})^3}{b-a} - \frac{(a^{\frac{1}{3}})^3}{b-a} = \frac{b-a}{b-a} = 1 \end{aligned}$$

3.2

3.2.a

The Jacobian matrix J has to be square because the determinant operation requires square matrices.

3.2.b

For each sample, we have to compute the determinant of the Jacobian matrix, which is an expensive operations. Additionally, unlike in Autoencoders, there is no bottleneck latent space but instead the input and latent space dimensionality have to be the same meaning that the number of dimensions can grow large.

3.3

3.3.a

Since Normalizing flows assume a continuous probability space. One method to turn a discrete space into a continuous one is dequantization. However, as Tutorial 10 demonstrates, simple dequantization is not sufficent for images because it uses a uniform distribution. However, when we drop the uniform assumption, which we have to do for modelling images, simple dequantization leads to a distribution with sharp, non-smooth edges which cannot be transformed to the Gaussian distribution that we desire. The solution to this problem is Variational Dequantization where we use another flow that learns a distribution over the random variable u and replaces the uniform distribution used in simple dequantization.

3.3.b

Flow-based models can be trained by using variational dequantization, coupling layers and a neural network architecture like MLPs or CNNs. During training, the input consists of our training data X and the output is the density of the corresponding latent space representation. The model can be optimized and evaluated via the negative log likelihood loss. As Tutorial 11 suggests, after training a model can be evaluated based on validation and test scores, the inference and sampling time (how long it takes to calculate the probability of an image and how long it takes to sample an image respectively), as well as the number of total parameters. We can also evaluate the model qualitatively by sampling from the prior distribution and inverting the representation via a reversed flow transformation to a data example. So here the input is a sample from out latent space density and the output is a data representation resembling the training data. Additionally, we can visualize the smoothness of the latent space via interpolation.

3.4 Bonus

Although the BPD score is significantly lower than the suggested score and I am not sure where I made the mistake, the Flow model does learn the bimodal distribution as Figure 9 shows. The variance of the learned distribution is lower than the target distribution. The means seem to be centered correctly, but the deviating variance is likely the source of my error.

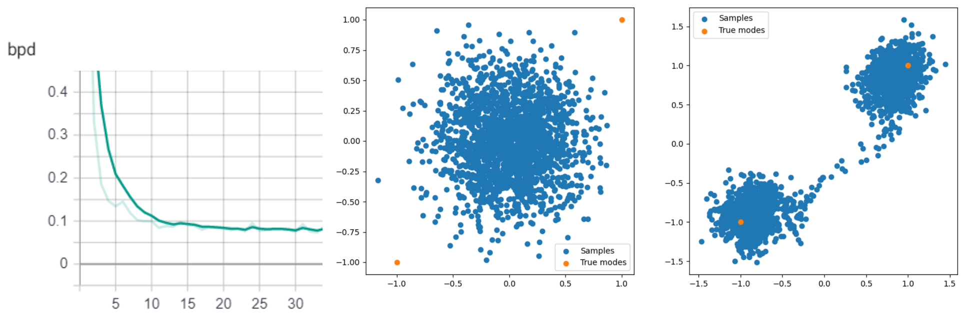

Figure 8: BPD score, and samples prior to training and after training

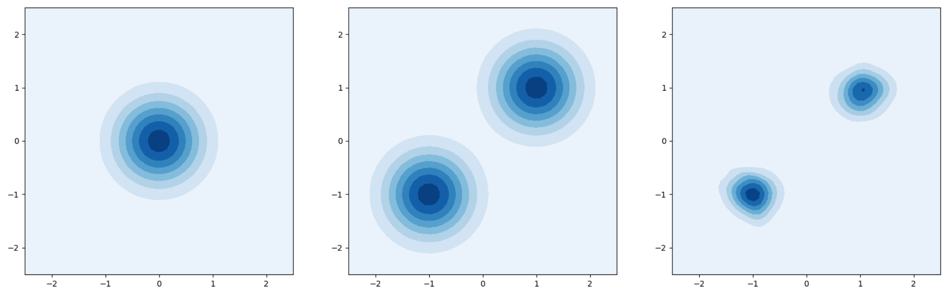

Figure 9: Base density, target density and learned density of Normalizing Flow Model

4 Conclusion

VAEs, GANs and Flow-based models all belong to the class of generative models that are trained to generate new data which resembles the training data. However, there are also noticable differences between them. In VAEs we map the input to a bottleneck latent distribution, with considerable fewer dimensions than the input. This learned latent distribution allows us to sample from it and generate new images. In Flow-based models we also learn a latent distribution, however, it has to have the same dimension as the input space because we are learning a one-to-one mapping from the data to the latent space and require invertibility. The model architecture is therefore limited and the generated sample we obtain, even though the model optimizes the likelihood of the data exactly, are worse than VAEs or GANs. In contrast, GANs use a Generator that maps random noise to an image that resembles true images from the training set and we do not obtain a distribution over latent variables. While GANs generally generate better quality images than VAEs and Flow-based models, training them is much more difficult as the minimax game approach is harder to optimize.

References

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.