Assignment 3. Deep Generative Models

Max Bos
10669027
University of Amsterdam
maxn.bos@gmail.com

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

1. The VAE and standard autoencoder are similar in the sense that they both can be used to autoencode a given input, so to map an input space to a lower-dimensional latent space.
2. A standard autoencoder is not generative, since it only learns latent representations of real input data, and learns to decode this latent representation to look exactly like the original input.
3. A variational autoencoder can be used in place of a standard autoencoder. But since the variational autoencoder has a distribution from which the latent space vector can be sampled, one should make it deterministic by getting the mean values from the distribution.
4. The VAE learns a variance and standard deviation parameter which can be used to create a distribution from which a latent space vector can be sampled. This sampled vector can be fed to the decoder network of the VAE to generate new images.

1.2 Decoder: The Generative Part of the VAE

Question 1.2

To sample from a model of Bernoulli distributed to construct an image x_n in $\mathcal{D}$ we can use ancestral sampling. First, the noise z_n is sampled from a Normal distribution and subsequently fed into a Neural Network parameterized by θ to calculate a mean vector. The m-th value in the mean vector is used as the mean value for the Bernoulli distribution from which the m-th value of the x_n image is sampled independently.

Question 1.3

The noise is sampled from a standard normal distribution but fed into a function that maps the latent space to a data space, where the mapping function is parameterized. The mapping function will learn the correct parameter values given the fixed latent space.

Question 1.4

(a)

$$\log p\left(\mathbf{x}_n\right) \approx \log \frac{1}{N} \sum_{n=1}^{N} p(x_n|z_n), z_n \sim P\left(z_n\right)$$

(b) Above approach is inefficient since it tries to approximate the log probability of the dataset by approximating the estimated probability of each image in the dataset by sampling

from the distributions. When the dimensionality of the noise increases, the complexity of approximating the estimate increases, so then the number of samples should be increased.

1.3 The Encoder: $q_\phi(z_n|x_n)$

Question 1.5

(a) For $\mu_q = 0$ and $\sigma_q^2 = 1$, the KL-divergence $D_{\mathrm{KL}}(q\|p)$ would be very small, since then $q = \mathcal{N}\left(\mu_q, \sigma_q^2\right)$ would be equal to p. For $\mu_q = 100$ and $\sigma_q^2 = 100$, $D_{\mathrm{KL}}(q\|p)$ would be very large, since the two distributions are highly dissimilar.

(b) A closed form formula for $D_{KL}(q\|p)$, where $q = \mathcal{N}\left(\mu_q, \sigma_q\right)$ and $p = \mathcal{N}(\mu_p, \sigma_p)$, would be:

$$D_{KL}(q\|p) = \frac{(\mu_p - \mu_q)^2 + \sigma_p^2 - \sigma_q^2}{2\sigma_q^2} + \ln\left(\frac{\sigma_q}{\sigma_p}\right)$$

A closed form formula for $D_{KL}(q\|p)$, where $q = \mathcal{N}\left(\mu_q, \sigma_q^2\right)$ and $p = \mathcal{N}(0, 1)$, would then be:

$$D_{KL}(q\|p) = \frac{(-\mu_q)^2 + 1^2 - \sigma_q^4}{2\sigma_q^4} + \ln\left(\frac{\sigma_q^2}{1}\right)$$

Above answer is taken from http://allisons.org/ll/MML/KL/Normal/.

Question 1.6

Since the log probability of the data can be calculated by adding the KL-divergence to the ELBO, and the KL-divergence always is greater or equal to zero, the ELBO can be seen as a minimum/bias of the log probability of the data.

Question 1.7

We must optimize the lower-bound instead of the log probability of the data since it is intractable to optimize the log-probability directly.

Question 1.8

When the lower-bound is pushed up, the log probability of the data needs to increase or the KL-divergence between the variational posterior $q(z|x)$ and the true posterior $p(z|x)$ needs to decrease.

1.4 Specifying the encoder $q_\phi(z_n|x_n)$

Question 1.9

The name reconstruction loss for $-\mathbb{E}_{q_\phi(z|x_n)}\left[\log p_\theta\left(x_n|Z\right)\right]$ is appropriate since this calculates the negative log probability of the data x_n given a latent vector z that is sampled given the data x_n, which would be smaller if the probability of the data given z is greater. The name regularization for $D_{\mathrm{KL}}\left(q_\phi\left(Z|x_n\right)\|p_\theta(Z)\right)$ is appropriate since it calculates the difference between a desired distribution and the current encoder distribution, and it is aimed to have a small difference, i.e. yielding an ecoder distribution that is similar to the desired distribution, this way it regularizes the variance and bias of the encoder distribution.

Question 1.10

For the reconstruction loss we can use the Monte Carlo integration answer to question 1.4, to write out a tractable expression:

$$\mathcal{L}_n^{\text{recon}} = -\mathbb{E}_{q_\phi(z|x_n)}\left[\log p_\theta\left(x_n|Z\right)\right] \tag{1}$$

$$= -\log\frac{1}{N}\sum_{n=1}^{N} p(x_n|Z), Z \sim P\left(Z\right) \tag{2}$$

For a tractable version of the KL-divergence we can use the closed-form solution as given in question 1.5.

$$\mathcal{L}_n^{\text{reg}} = D_{\text{KL}}\left(q_\phi\left(Z|x_n\right)\|p_\theta(Z)\right) \tag{3}$$

$$= \frac{(\mu_p - \mu_q)^2 + \sigma_p^2 - \sigma_q^2}{2\sigma_q^2} + \ln\left(\frac{\sigma_q}{\sigma_p}\right) \tag{4}$$

1.5 The Reparameterization Trick

Question 1.11

(a) We need the gradient of the loss function with respect to ϕ because we want to update the parameters ϕ of the encoder network using the gradient.

(b) The sampling is done from a continuous distribution from which we can not compute the gradient when using backpropagation, since when backpropagating we cannot compute a gradient of that distribution for that specific outcome.

(c) The reparameterization trick is a way of re-writing the method for generating samples from $q_\phi(\mathbf{z}|\mathbf{x})$. This is done by constructing the sampling expression using a mean μ, standard deviation σ, and random noise ϵ parameter: $z = \mu + \sigma\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$. Now, using this expression, a gradient can be calculated with respect to μ and σ, so we can perform backpropagation through the encoder network.

1.6 Putting things together: Building a VAE

Question 1.12

The implementation of the VAE consists of an Encoder and Decoder network. The Encoder network first receives data in a first linear layer with ReLU activation, subsequently the result is fed into two different linear layers, for the mean and log-variance. The Decoder network first has a linear layer with ReLU activation and a final linear layer with sigmoid activation. The forward functionality of the VAE first feds the input to the Encoder network, then uses the resulting mean and log-variance, calculates the standard deviation, to sample using the reparameterization trick. The resulting noise is fed into the Decoder network after which the reconstruction is used to calculate and return the average negative ELBO loss, based on the reconstruction loss and regularization term. Only when in training, we perform backward propagation on the ELBO loss and use the Adam optimizer to update the parameters in the encoder and decoder network.

Question 1.13

Figure 1 shows the estimated lower-bounds of the training and validation set on the binary MNIST dataset as training progresses over 40 epochs, using a 20-dimensional latent space, and default Adam optimizer learning rate of $1e-3$.

Question 1.14

Figure 4 depicts samples from the VAE model at three different stages in the training process, namely before, halfway, and at the end. The sampling is performed by first generating standard normal distributed random noise, subsequently feeding this into the Decoder network, and finally using the resulting output value for each pixel as the mean value for a Bernoulli sampled value. At the first epoch, the model already seems to have learned some characteristics of the MNIST dataset, as a some shapes are somewhat recognizable as a 1 or 0. Halfway, at epoch 20, the model has clearly learned characteristics of the digits. The samples at the last epoch show only a slight improvement over the samples at epoch 20.

Question 1.15

Figure 3 depicts the data manifold of a VAE with a 2-dimensional latent space.

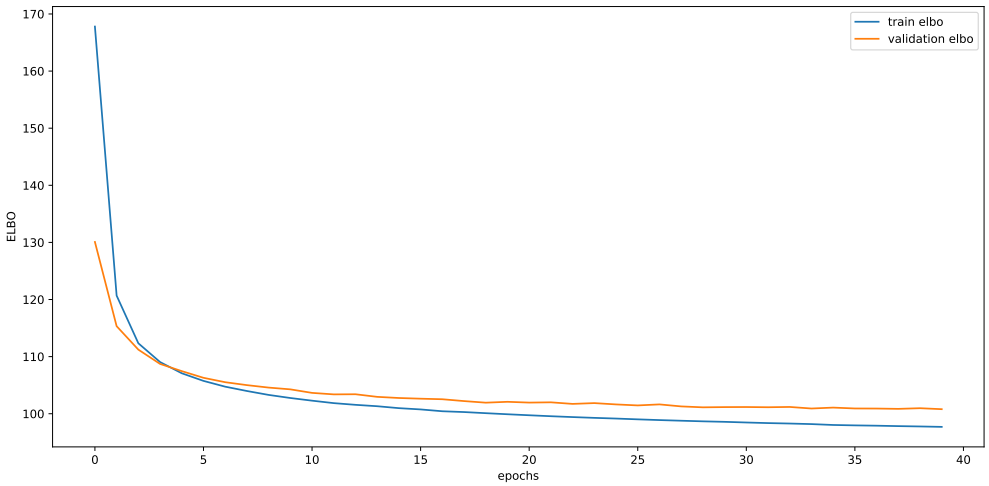

Figure 1: The estimated lower-bounds of the training and validation set as training progresses over 40 epochs — using a 20-dimensional latent space.

(a) Samples at epoch 1 (b) Samples at epoch 20 (c) Samples at epoch 40

Figure 2: Samples from the VAE model at three different stages in the training process, before, halfway, and at the end.

2 Generative Adverserial Network

Question 2.1

For the generator component in the GAN the input would be a matrix $\mathbf{z}$ of random noise sampled from a normal or uniform distribution and the output would a matrix $\mathbf{G}$, e.g. an image, that is generated from the input noise. For the discriminator component in the GAN the input would be a matrix $\mathbf{G}$, i.e. the output from the generator component, or a matrix $\mathbf{x}$ sampled from real input data, e.g. real images and the output would be a probability p of the input being real.

2.1 Training objective: A Minimax Game

Question 2.2

The GAN training objective as defined in Equation 5, consists of two entities with conflicting objectives. The discriminator D aims to maximize the objective function, $\max_D \mathbb{E}_{p_{\text{data}}(x)}[\log D(X)] + \mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$ while the generator G aims to minimize the objective function, $\min_G \mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$. The first term of the objective function, $\mathbb{E}_{p_{\text{data}}(x)}[\log D(X)]$ is explained as the expected log probability for the discriminator predicting that a given real input image $\mathbf{X}$ is real. The second term of the objective, $\mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$, is explained as the

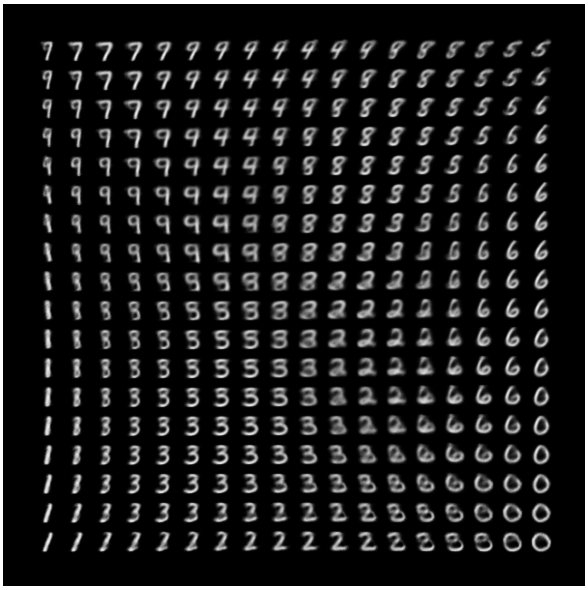

Figure 3: Data manifold of a VAE with a 2-dimensional latent space.

expected log probability of the discriminator predicting that a given fake input image $G(Z)$ is fake, since $1 - D(G(Z))$ is the probability of the input image not being real.

$$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{p_{\text{data}}(x)}[\log D(X)] + \mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))] \tag{5}$$

Question 2.3

Since the GAN is based on the minimax game, the GAN converges when the discriminator and the generator reach a Nash Equilibrium. So the value of $V(D,G)$ would be the optimal point for Equation 5, which would be $log(0.5) + log(1 - 0.5) \approx -0.602$.

Question 2.4

Early on in training the fake input images are likely to be predicted to be not real, so in the extreme cases the result of $D(G(Z))$ would then be 0, yielding a loss value of $log(1 - 0) = 0$, which entail flat gradients from which the generator cannot learn. This problem can be solved by changing the generator objective from minimizing the expected probability of the discriminator predicting a fake image to be fake to maximizing the expected probability of the discriminator predicting a fake image to be real: $\max_G \mathbb{E}_{p_z(z)}[\log(D(G(Z)))]$.

2.2 Building a GAN

Question 2.5

The Discriminator takes a 1-dimensional representation of an image and outputs a value between 0 and 1, and is defined by a sequence of LeakyReLU activated linear layers ending with a sigmoid activation. The Generator input size is equal to the number of latent dimensions and output size equal to the input size of the Discriminator. The Generator is defined by a combination of LeakyReLU batch normalized linear layers, ending with a tanh activated linear layer. When training, the Discriminator is first trained by feeding a batch of real images, and subsequently calculating the loss on real labels. Second, the Discriminator is trained by feeding a batch of detached fake images generated by the Generator after which the Discriminator loss is calculated on fake labels. Total loss for the Discriminator is $loss\_real + loss\_fake$. Lastly, the Generator is trained by forward passing the fake images through the Discriminator and calculating a loss based on the real labels, since the Generator wants to maximize the probability that the Discriminator classifies a fake image as real.

(a) Samples at start of training (b) Samples at halfway of training (c) Samples at end of training

Figure 4: Samples from the GAN model at three different stages in the training process, start, halfway, and at the end.

Question 2.6

Figure 4 depicts samples from the GAN model at three different stages in the training process.

Question 2.7

Figure 5 shows a 7-step interpolation between 2 images from different classes, sampled from the final GAN model. The images are sampled by generating random noise and feeding this into the Generator network.

Figure 5: 7-step interpolation between 2 GAN sampled images from different classes.

3 Generative Normalizing Flows

3.1 Change of variables for Neural Networks

Question 3.1

$$\begin{gathered} \mathbf{z} \sim \pi(\mathbf{z}), \mathbf{x} = f(\mathbf{z}), \mathbf{z} = f^{-1}(\mathbf{x}) \\ p(\mathbf{x}) = \pi(\mathbf{z}) \left|\det \frac{d\mathbf{z}}{d\mathbf{x}}\right| = \pi\left(f^{-1}(\mathbf{x})\right) \left|\det \frac{df^{-1}}{d\mathbf{x}}\right| \end{gathered} \tag{6}$$

Question 3.2

The transformation function f should be easily invertible. So the output of the transformation function f should be a square invertible matrix. $\mathbf{x}$ and $f(\mathbf{z})$ should have equal dimensions.

Question 3.3

The Jacobian determinant of the function f must be easy to compute.

Question 3.4

A degenerate solution with all probability mass placed on discrete datapoints will be the result of fitting a continuous density model, e.g. change-of-variables formula, to discrete integer data [1]. One could fix this problem by first converting the discrete integer data distribution into a continuous distribution using "dequantization". Then, model the resulting continuous distribution using the continuous density model, e.g. change-of-variables formula.

3.2 Building a flow-based model

Question 3.5

The input would be a sample from a dataset, and the output would be the log-likelihood of the sample. During inference time we care about sampling data from the trained model, by first sampling random noise z as $\mathbf{z} \sim \pi(\mathbf{z})$ and then calculating a data image $\mathbf{x} = f(\mathbf{z})$.

Question 3.6

The steps one needs to take when training a flow-based model:

1. Perform dequantization and logit normalization on the input to convert the discrete data to a continuous distribution in the range 0 and 1,
2. Perform forward pass through the Flow network
3. Compute $\log p(\mathbf{z})$ and $\log p(\mathbf{x})$

The steps one needs to take to sample from a flow-based model:

1. Sample noise z from a standard normal distribution,
2. Zero-initialize a log Jacobian determinant vector,
3. Invert the flow by performing a reverse propagation through the Flow network,
4. Invert the logit normalization to convert back the continuous data distribution to a discrete data distribution.

Question 3.7

See file `a3_nf_template.py` in the source code. The training implementation consists of feeding image data into a flow-based model and optimizing for the returned bits per dimension. A forward pass of the flow-based model consists of first dequantizing the input data and performing logit normalization to scale the range of the values to be between 0 and 1. The resulting data is fed to a flow network of 4 layers, from which the final log-likelihood of the data can be calculated.

Question 3.8

Figure 6 shows the training and validation performance in bits per dimension (negative log2 likelihood divided by the size of the image). After 27 epochs the model was able to reach a validation bpd below 1.85, and after 40 epochs the validation bpd was at around 1.83.

4 Conclusion

Question 4.1

From this assignment I have learned the definitions of three different generative models, namely VAEs, GANs and flow-based models. The three models differ in their approach of learning to reconstruct data from random input noise sampled from a latent space. VAEs aim to learn parameters for an encoder, latent distribution and decoder network. It does this by inexplicitly optimizing the log-likelihood of the data by maximizing the evidence lower bound. Flow-based type models are similar to VAEs since they both try to model the likelihood of the data. Though, flow-based models explicitly learn the mentioned $p(x)$ distribution and therefore simply has an objective of maximizing the likelihood. GANs are different from VAEs and Flow-based models in that they do not aim to model the likelihood of the data but model the data generation by converting it from an initially unsupervised problem to a supervised problem by, next to learning a generator network, simultaneously learning a network that aims to label fake images, generated by the generator network, and real images. A drawback of GANs is that they are unstable in training since the objective of the model is designed as a zero-sum game. On other hand, VAEs and flow-based models are more stable during training time.

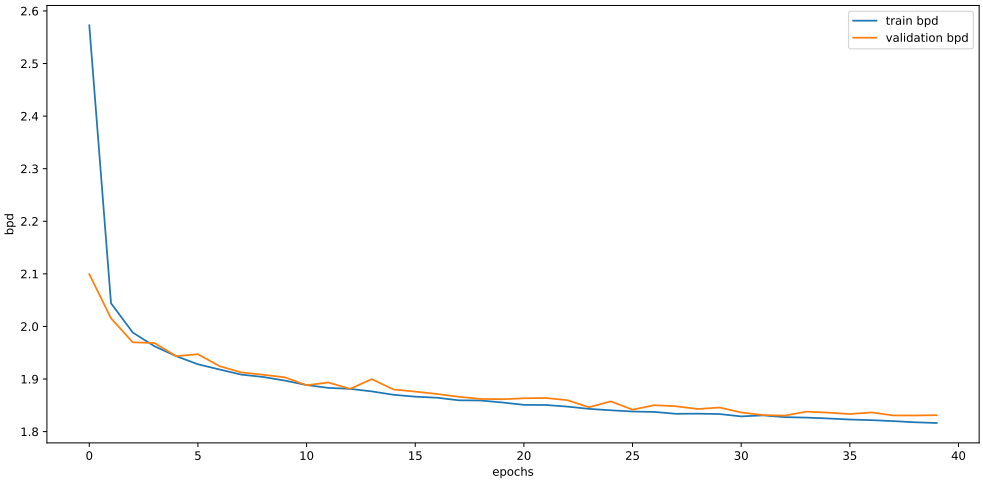

Figure 6: Training and validation performance in bits per dimension (negative log-2 likelihood divided by the size of the image). After 40 epochs the model has a validation bits per dimension of around 1.83.

References

[1] Uria, Benigno, Iain Murray, and Hugo Larochelle. "RNADE: The real-valued neural autoregressive density-estimator." Advances in Neural Information Processing Systems. 2013.