Hannah Lim
10588973
hannah_lim@outlook.com

Assignment 3
Deep Generative Models
Deep Learning, 2019

December 16, 2019

1 Variational Auto Encoders

1.1 Latent Variable Models

1.1)

A standard autoencoder consits of two principle components in the network: the encoder and the decoder. The encoder compresses the input to lower dimensional latent data, which is called the bottleneck. The decoder uses the lower dimensional data to reconstruct it to the original data. The loss is computed by comparing the original data with the reconstructed data. With this loss the network is learned.
The Variational Autoencoder (VAE) has a similar structure (it also consists of a an enconder and a decoder), but the encoder of the VAE maps the input to a distribution latent space instead of a vector (like the standard auto-encoder does). So for VAE's the bottleneck is replaced with two vectors: the mean of the distribution and the variance of the distribution. Then a sample is taken from the distribution, which is passed on to the decoder.
A VAE is generative because of its random sampling. The standard autoencoder is not generative, and also does not have the intention. The main goal of the autoencoder is to simply compress the input data to a lower dimension and then later decode it back to the original data. So therefore the standard autoencoder is only able to replicate input images and is not generative.

1.2 Decoder: The Generative Part of the VAE

1.2)

To sample from a VAE model ancestral sampling is used. This means that we first sample the variable that has no parents. Here that is the latent variable z_n. Now we have the probability distribution for x. Then x_n can be sampled from this distribution. In other words:
1. $z_n \backsim \mathcal{N}(0, \mathcal{I_D})$
2. $x_n \backsim Bern(x_n|f_\theta(z_n))$

1.3)

It is not such a restrictive assumption in practice that p(Z) does not have a trainable parameter because we are not interested in this latent space. Just in the function f_θ on that latent space. The decoder is able to learn a mapping from the latent space z_n to x_n. If p(z) is normally distributed and we train the decoder to sample from the distribution and from there on sample x_n, we do not need trainable parameters in the p(z).
Backpropagation could be made very difficult with VAE because how do you

backpropagate a derivative to a sampling function node. Here the reparamitization trick is used. The latent vector that we're sampling from is: $z = \mu + \sigma\dot{\epsilon}$. The only stochastic part of this derivation is the ϵ, but since this is fixed and we are not interested in this part we do not take the derivative here. We only need to train the parameters μ and σ.

1.4a)

$$log\ p(\mathcal{D}) = \sum_{n=1}^{N} log\, \mathbb{E}_{p(z_n)}[p(x_n|z_n)] = \sum_{n=1}^{N} log \int p(x_n|z_n)p(z_n)dz_n$$
$$\approx \sum_{n=1}^{N} log \frac{1}{J} \sum_{j=1}^{J} p(x_n|z_n^{(j)}) \text{ with } z_n^{(j)} \backsim p(z_n)$$

1.4b)

This approach to train VAE models is inefficient because we would require many samples to get a good estimate of the prior distribution ([1]). As can be seen in figure 2 of the assignment, samples taken from p(z), will mostly result in $p(x_n|z_n)$ to be zero. In addition, if the dimensionality of z increases, even more samples are needed to estimate and therefore will be more inefficient.

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

1.5a)

D_{KL} will be very small if μ_q is close to zero and σ_q^2 is close to 1. If $\mu_q = 0$ and $\sigma_q^2 = 1$, the D_{KL} is actually 0.
D_{KL} will be very large if μ_q is very big and σ_q^2 is very small. For example, if $\mu_q = 100$ and $\sigma_q^2 = 0.1$.

1.5b)

$D_{KL}(q||p) = log\frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2+(\mu_q-\mu_p)^2}{2\sigma_p^2} - \frac{1}{2}$
In this case where $\mu_p = 0$ and $\sigma_p = 1$:
$D_{KL}(q||p) = \frac{1}{2}(\sigma_q^2 + \mu_q^2 - log(\sigma_q) - 1)$

1.6)

The right hand side is called the lower bound, because it will always be smaller then the left-hand side. Because:
A property of D_{KL} is, that is always bigger or equal to 0. Therefore:
$logp(x_n) \geq logp(x_n) - D_{KL}(q(Z|x_n)||p(Z|x_n))$
$logp(x_n) - D_{KL}(q(Z|x_n)||p(Z|x_n)) = \mathbb{E}_{q(Z|x_n)}[logp(x_n|Z)] - D_{KL}(q(Z|x_n)||p(Z))$
so:
$logp(x_n) \geq \mathbb{E}_{q(Z|x_n)}[logp(x_n|Z)] - D_{KL}(q(Z|x_n)||p(Z))$

1.7)

Optimizing the log probability is not doable because for optimizing the log probability we need to compute the truee posterior distribution, wherefore we need to compute the integral $\int p(x_n|z_n)p(z_n)dz_n$. This integral is intractable.

Therefore if we optimize the lower bound, we will also be optimizing for the log probability since this is always higher than the lower bound.

1.8)

When the lower bound is pushed, the log probability will also be pushed up. This means that the KL divergence decreases and that the distributions look more alike. The lowerbound is pushed up to optimize the loss (between the distributions- that the distribution q matches more to the true distribution p). When the lower bound is pushed up two things can happen.

1. The log$p(x_n)$ increases.
2. The $D_{KL}(q(Z|x_n)||p(Z|x_n))$ became smaller and therefore the approximated posterior is more similar to the true posterior.

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

1.9)

The reconstruction loss describes how well the decoder is in reconstructing the original data. This contains on an expectation operator because its based on a sample from the distribution.
The regularization loss describes how close the posterior distribution is to the prior distribution. This is useful so the autoencoder will not overfit and it generalizes well enough.

1.10)

We want to minimize $\mathcal{L} = \sum_{n=1}^{N} \mathcal{L}_n^{recon} + \mathcal{L}_n^{reg}$

Constructing $\mathcal{L}_n^{recon}$:
$\mathcal{L}_n^{recon} = -\,\mathbb{E}_{q_\phi(z|x_n)}[log p_\theta(x_n|Z)]$
We know: $p_\theta(x_n|Z) = \prod_{m=1}^{M} Bern(x_n^{(m)}|f_\theta(z_n)_m$ (equation 4 of assignment)
Filling in:
$\mathcal{L}_n^{recon} = -\,\mathbb{E}_{q_\phi(z|x_n)}[log \prod_{m=1}^{M} Bern(x_n^{(m)}|f_\theta(z_n)_m]$

Constructing $\mathcal{L}_n^{reg}$:
$\mathcal{L}_n^{reg} = D_{KL}(q_\phi(Z|x_n)||p_\theta(Z)) = log\frac{\sigma_{p_\theta(Z)}}{\sigma_{q_\phi(Z|x_n)}} + \frac{\sigma^2_{q_\phi(Z|x_n)}+(\mu_{q_\phi(Z|x_n)}-\mu_{p_\theta(Z)})^2}{2\sigma^2_{p_\theta(Z)}} - \frac{1}{2}$

1.5 The Reparametrization Trick

1.11)

We need $\nabla_\phi \mathcal{L}$ to be able to optimize the parameters of the encoder. The loss is computed based on one or more randomly chosen samples. Obtaining the gradient analytically is impossible due to the act of sampling in the autoencoder. Sampling is a nondifferentiable operation. To still be able to derive the gradient we use the reparametrization trick.

The reparamatrization trick makes use of external sampling instead of sampling directly from the complex probability distribution. This sample (gained with external sampling) undergoes a deterministic transformation by multiplying with the standard deviation and adding the mean of the distribution into a sample of the modeled distribution.
Now the only part that contains stochastic units is the external sampling part. Since we do not need to update this (usually just $\mathcal{N} \backsim (0, 1)$), we are able to compute $\nabla_\phi \mathcal{L}$ since it is does not require differentating over a sampling operation.

1.6 Putting things together: Building a VAE

1.12)

The VAE model implemented consists of the following structure:

- Encoder The encoder is a MLP with a single hidden layer containing 500 hidden units. This is followed by the non-linearity function ReLU. The output is obtained with two seperate layers computing the mean and the variance. Here we used the log of the variance to obtain numerical stability of the approximate distribution.
- Sampling process With the reparametrization trick a sample is taken from the approximate distribution as explained in question 1.11.
- Decoder The decoder is also a MLP with a single hidden layer 500 hidden units, followed by a ReLU. The output is a linear layer, generating a vector that represents the constructed image of size 28x28.
- Loss The ELBO is computed as shown before (the reconstruction loss + the KL-divergence).

The model is trained on the MNIST dataset, which contains images of numbers with size 28x28. The optimizer used in this task is the Adam optimizer with the default parameters.

1.13)

As can be seen in figure 1 the ELBO of the training and the validation set decreases during training. The ELBO quickly decreases the first epochs and hereafter only slowly decreases. This means that the model is quickly able to generate images that begin to look like the original images and hereafter only slowly improves in this task.

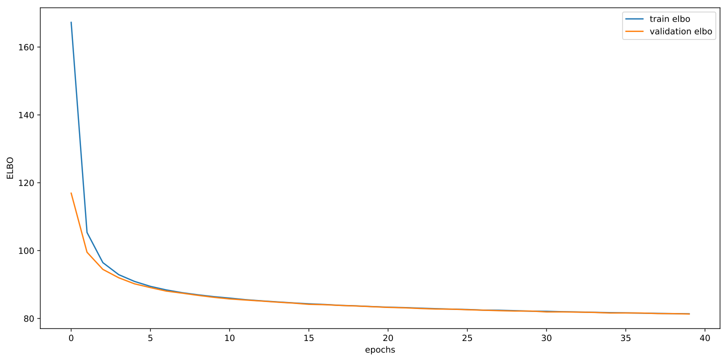
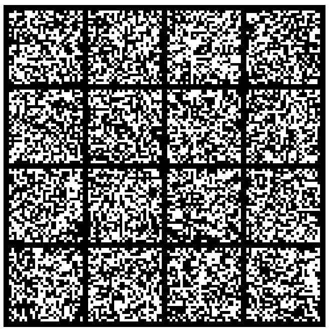


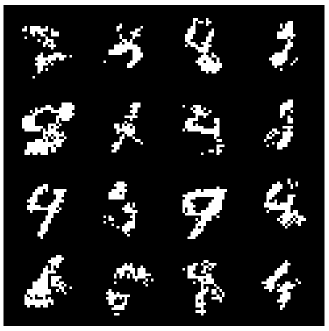
Figure 1: Estimated lower-bound of train and validation set during training using a 20-dimensional latent space

1.14)

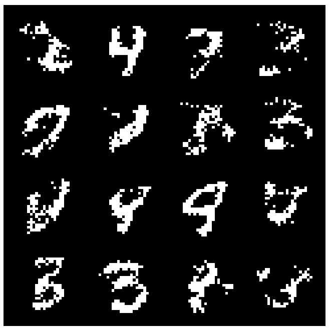
As can be seen in figures 2a, 2b and 2c the model improves a lot during training. Before training the model just generates random noise. During training (after 20 epochs) the model is able to generate images that begin to look like numbers. After training (40 epochs) the model has generated images that look more like numbers but are not always clear enough yet. The ELBO converges to approximately 90 for bot the validation and training set.



(a) Before training



(b) Training halfway (20 epochs)



(c) After training (40 epochs)

Figure 2: Samples from the VAE model during different timesteps

1.15)

2 Generative Adversarial Networks

2.1)

The input for the generator is random noise and the output will be an image. Because the input is randomized the generator is able to generate varying new images the model has never seen before. The discriminator will acts as a classifier. The input for the discriminator will be an image and the output will be a classification (between 0 and 1) of the image being real or fake. The discriminator is trained on true images and images generated by the generator and tries to detect if the input image is real or fake.

2.1 Training objective: A Minimax Game

2.2)

In the minimax game the discriminator tries to predict the classification of an image being real or fake as good as possible. The minimax game is further explained by inspecting the following equation:

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

The first term in the equation ($\mathbb{E}_{p_{\text{data}}x}[logD(X)]$) represents the performance of the discriminator on images sampled from the database containing real images. Maximizing this term the discriminator will correctly classify real images as being real.
The second term in the equation ($\mathbb{E}_{p_z(z)}[log(1-D(G(Z)))]$) represents the performance of the discriminator on images generated from the generator. Maximizing this term will result in a discriminator that will correctly classify these fake images as being fake. But in contrast the generator tries to fool the discriminator, as it wants the discriminator to predict a fake image, generated by the generator, to be classified as a real image.

2.3)

When the GAN has converged (to the Nash equilibrium) the generator is able to generate images that are indistinguishable from real images from the dataset. Then the discriminator is unable to distinguish between the real and the fake images and therefore the D(X) and the D(G(Z)) will be $\frac{1}{2}$. This results in a value of V(D,G) = $\log\frac{1}{2} + \log\frac{1}{2} = -2\log 2$

2.4)

Early in the training the generator is not able to generate images that approach the quality of a real image of the dataset. At this point the discriminator will be pretty good in classifying these fake images as fake. Therefore the $log(1 - D(G(Z)))$ term will be close to 0. This results in vanishing gradients for the generative model and therefore the generative model won't optimize properly and will become very bad.

2.2 Building a GAN

2.5)

The implemented GAN model has the following structure:

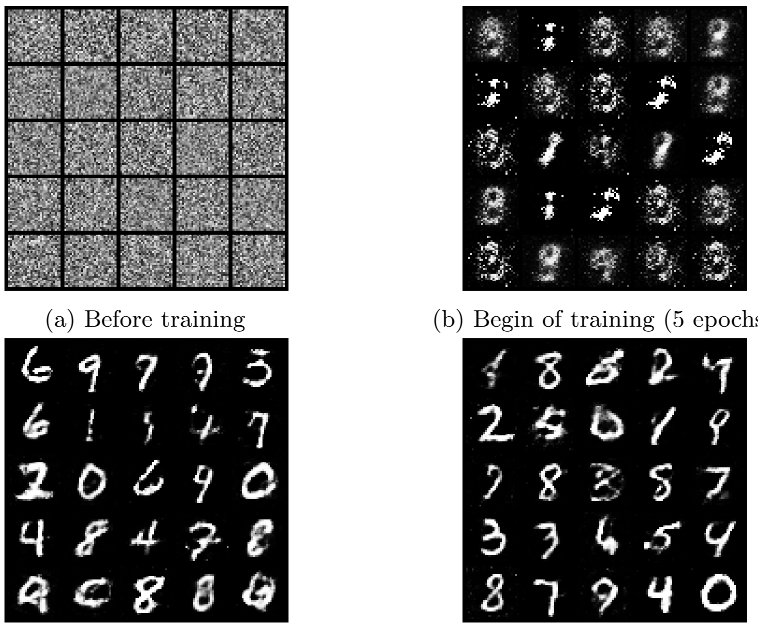
- Generator
  The input of the generator is a latent vector with a dimensionality of 100. The generator consists of five linear layers followed by Batch Normalization (except the first layer) and by Leaky ReLUs (slope = 0.2). The final layer is a non-linearity Tanh layer. The output of the generator is a vector representing an image (28x28).

- Discriminator
  The discriminator also consists of three linear layers followed by Leaky ReLUs (slope = 0.2). In the discriminator we made use of the Dropout function of 0.3. The final layer is a non-linearity function Sigmoid (containing the information of the classification of the input being real or fake).

In the experiments we trained on the dataset MNIST and used a batch size of 64. We trained with the optimizer Adam, with learning rate 0.0002.

2.6)

In figure 3 the improvement of the GAN can be seen. Before training, begin of training, halfway trough training and after training samples are taken by randomly passing a latent variable to the generator.



(a) Before training

(b) Begin of training (5 epochs)

(c) Halfway training (100 epochs)

(d) After training (200 epochs)

Figure 3: Samples from the GAN model during different timesteps

2.7)

3 Generative Normalizing Flows

3.1 Change of variables for Neural Networks

3.1)

Invertibility of f:
$z = f(x); x = f^{-1}(z); p(x) = p(z)|\det(\frac{df(x)}{dx})|$
Log-likelihood:
$\log\, p(x) = \log\, p(z) + \sum_{l=1}^{L} \log\, |\det(\frac{df(x)}{dx})|$

3.2)

The constraints are:

1. h_l and h_{l-1} need to have the same dimension [2], otherwise invertibility is not possible.
2. the transformation must be invertible
3. Jacobian matrix needs to be square, otherwise we are not able to compute the determinant. This will automatically be square if h_l and h_{l-1} have the same dimensions.

3.3)

This network will be computationally expensive because the computational cost of the inverse of the gradient of the Jacobian matrix during backpropagation.

3.4)

Because the change-of-variable formula assumes continuous random variables, giving it discrete variables will cause the model to produce a distribution around the discrete variables instead of a smoother distribution that fits more to the true distribution. We could fix this problem by making the discrete data continuous by adding random noise to the data. This is for example done in the function 'dequantize' of the flow-based model in question 3.6.

3.2 The coupling-layers of Real NVP

3.3 Building a flow-based model

3.6)

During training:

- The input is a flat image vector and is fed in the forward model.
- The model consists of two coupling layers; each consisting of two linear layers followed by a ReLU and a scale and translation function to get the latent vector z. In addition, the log Jacobian determinant is computed which is later needed to compute the log $p(x)$

- The log $p(x)$ is computed by computing log $p(z)$ and using the log Jacobian determinant. This log $p(x)$ is used as our negative loss function, which is used for backpropagation to update the parameters. We used the optimizer Adam, with learning rate 1e-3

After training:

- The input is a latent vector sampled from the prior distribution $p(z)$ and is fed in the reverse model.
- The output is a generated image.

In the experiments we trained on the dataset MNIST and used batches of size 128.

3.7)

See code.

3.8)

As can be seen in figure 4 the loss decreases very fast during training (after 5 epochs already converged) to around the 1.8 bits per dimension.

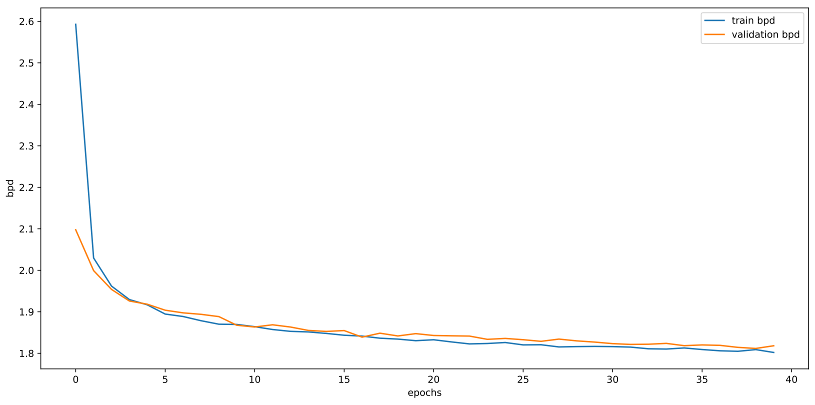
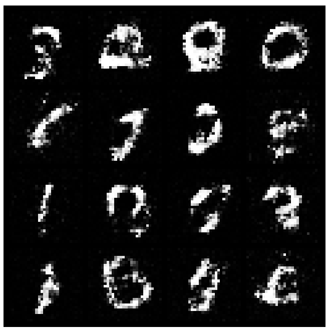
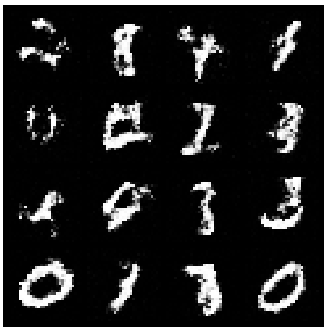


Figure 4: BPD during training


(a) Begin training (5 epochs)


(b) Training halfway (20 epochs)

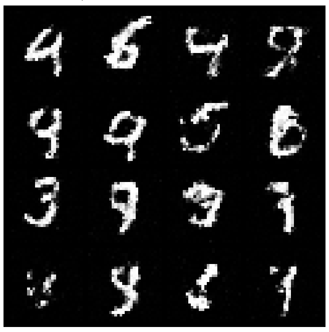

(c) After training (40 epochs)

Figure 5: Samples from the NF model during different timesteps

3.4 Conclusion

4.1)

In this report we have investigated three generative models; Variational Auto Encoders, Generative Adversarial Networks and Generative Normalizing Flows. We compared these three models based on their qualitative ability to generate images, on their computational cost and on their ability to model the data distribution.
The GAN model generates the most qualitative images as can be seen in the figures 2, 3 and 5. Thereafter the flow-based model and last the VAE. I believe that GANs are the best model for generating images because they are able to generate an image from random noise, while the VAE learns to generate images from latent space which is based on a real image.
Flows are computationally most expensive, because it trains sequentially and also generates sequentially. GANs and the VAEs are able to sample parallel. Moreover, the VAE and the Normalizing Flow are able to explicitly approximate (VAE) or compute (Normalizing Flow) the data distribution, while GANs cannot. Finally, we can conclude that GAN's are the best model to generate qualitative images but in other work this could lead to limitations because we are not able to explicitely derive the maximum likelihood where the VAE (approximates) and the NFs can.

References

[1] Carl Doersch. *Tutorial on Variational Autoencoders*. Carnegie Mellon / UC Berkeley, August 16, 2016.

[2] `https://deepgenerativemodels.github.io/notes/flow/`