Stephan Grzelkowski
10342931

1 Variational auto encoder

1.1

(1)
The Variational auto encode is a specialized form of the standard auto-encoder. The main function can be viewed as similar. Both auto-encoder and VAE learn the latent space representations of some data and perform this through an encoding step, from original sample to representation, and an decoding step, from representation to reconstruction. The difference between the two is how the latent space representation is build. The VAE adds distributional constraints to the latent space construction in the encoding step. The latent space is build as a sample from a pre-determined distribution whose parameters are learned by the network. Both are forms of unsupervised learning and can be used to learn data structures without the need for labels. The VAE has an additional functionality as it allows to generate new data by drawing samples from the learned distributions of the latent space and subsequently running the decoding process.
(2)
As alluded to in the previous answer, the VAE has generative potential. This is less so the case for a standard auto-encoder as an imput sample is required to build a meaningful representation of the latent variables. So, while it is technically possibly to choose some arbitrary values for the latent space variables and then run the decoding, the resulting output might be far from the training data samples.

1.2 Ancestral sampling

As generation of sampling of x depends on the variable Z we need to perform forward (or ancenstral sampling). We first sample the variable z according to the equation 3 (a standard-normal gaussian). We can now sample the marginal distribution of x given z according to equation 4.

1.3

The answer to this question lies in the mapping of the latent variables z_n through the function f_θ. With complex enough function $f_\theta(z_n)$ we can approximate any underlying representation of z. In the VAE we accomplish this as $f_\theta(z_n)$ is the neural network with parameters θ mapping z to the means of the bernoulli distribution which return pixel values for x. [1]

1.4

(a)
We can approximate log$p(x_n)$ using the monte carlo methods, simply sampling z for enough times and averaging over all samples:

$$\mathbb{E}_{p(z_n)}[p(x_n|z_n) \approx \frac{1}{T}\sum_{t}^{T} p(x_n|z_n^t)$$

Where z_n is a random sample drawn from a distribution and T is the total number of samples drawn.
(b)

The problem with this approximation, is that with high dimensionality of z_n we will need a very large amount of samples. So if we want to generate an accurate approximation of the model parameters we would need a high number of samples [1]. This would result in a high computational load.

1.5

(a)
One easy choice for parameters to create a very small divergence is to choose the same parameters for mean and variance of q as for p:

$$\mu_q = 0$$
$$\sigma_q^2 = 1$$

It is easy to see that that for all values of x the term $\log \frac{p(x)}{q(x)}$ becomes 0 so the integral becomes 0 as well and the KL-divergence is very small (0).
If we take a value for μ_q that is far away from 0, say:

$$\mu_q = 10^{666}$$

the term $\log \frac{p(x)}{q(x)}$ will become very negative so the KL-divergence will become very large. **(b)** I found the closed form solution for the DL divergence of 2 univariate gaussians here:

$$KL(p, q) = \log \frac{\sigma_q}{\sigma_p} + \frac{\sigma_p + (\mu_p - \mu_q)^2}{2\sigma_q^2} - \frac{1}{2}$$

1.6

I found a useful tutorial for answering the next 3 questions here. I have taken the information from there.
As the KL divergence is always non-negative, minimizing the KL means maximizing the lower bound.

1.7

The log-probability is an intractable integral, whereas the lower bound serves as a good approximation. By taking samples of the latent variable we can compute the KL-divergence directly and with this we can draw samples for x given z allowing us to produce estimated values to compute the Loss.

1.8

I don't really understand what is meant by push up. I would assume you mean how to maximize the left hand side? So maximizing the log probability log$p(x_n)$ or miniziming the KL divergence since it is substracted. So that is either creating likely reconstruction of the samples of x or having a prior and the approximated distribution of the latent variable Z be very similar.

1.9

This question is somewhat related to the previous question. The reconstruction loss is the divergence of the new samples from our input - aiming for the best reconstruction of the input space through the latent variable z. The regularization term punishes distributions of Z that are far from our prior distribution of Z ($p_\theta(Z)$. In other words, we are regularization the fitting of the latent space on our input to not diverge too far from our prior beliefs.

1.10

1.11

(a)
We want to find the best parameters Φ for the distribution of the latent variables that are closest to the

distribution underlying our data and we need the gradient for optimization. **(b)**
If the sampling operation is included in the loss computation then we cannot calculate the gradient of the loss as sampling is not a continuous operation and hence not differentiable [1]. **(c)**
The reparametrization trick instead samples from a univariate gaussian (which is a differentiable operation) and then adds the mean and multiplies the standard deviation of the actual target distributions. This yields the same reesults but because we have specific values for the mean and standard deviation for the current samples, we can calculate the gradient over these samples.

1.12

I implemented the VAE using the Doersch tutorial [1]. I also made use of this tutorial for the explicit form of the loss. The model is made up of the encoder and the decoder.

1.13

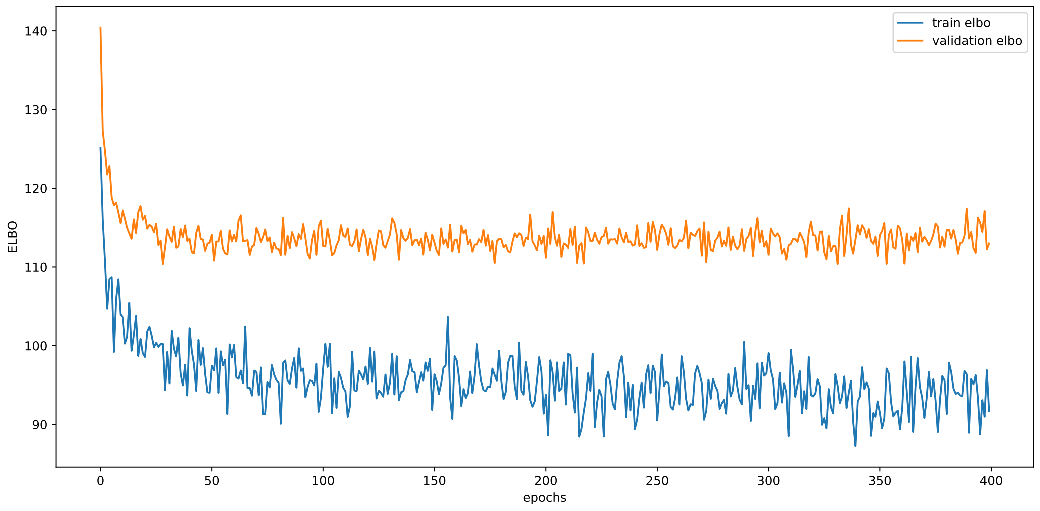

Figure 1: The lower bound of training and testing loss in the VAE

We can see that after about 50 training iterations the testing loss does no longer improve much more.

1.14

I chose to plot the means of the output instead of running them through the bernoulli distribution. I also made plots with bernoulli sampling initially but I found the means gave better looking images. Although i trained for longer, since the validation set did not improve much after he first couple hundred epochs, I plotted images generated after 0, 100 and 200 epochs: We see that image qualitiy improves. The first images are very unclear and it is very difficult to recognize any digits. After the first one hundred epochs we see clear improvement but some images are clear. After 200 epochs most images have identifyable digits with some exceptions.

2 GAN

2.1

The generator takes as input some noise. It's goal is to create real looking images from this noise. It outputs the generated image to the discriminator network. So it maps some input vector ρ to an output image. The discriminator takes an input image and aims to classify it correctly as a fake or real image. Hence the output of the discriminator network is either 1 for a real image or 0 for a fake image. During training it takes either real images or images generated by the generator network.

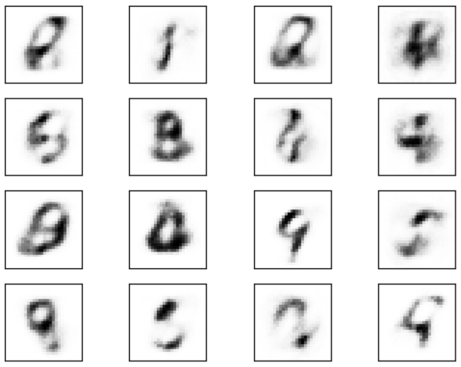

Figure 2: VAE generated images after the first epoch

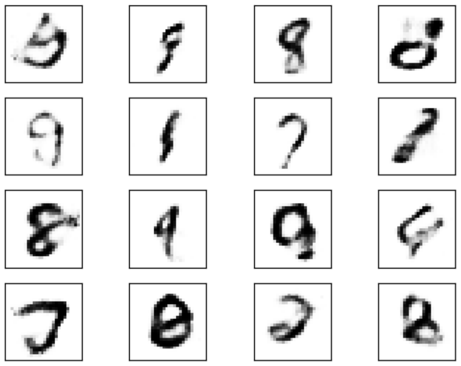

Figure 3: VAE generated images after 100 epochs

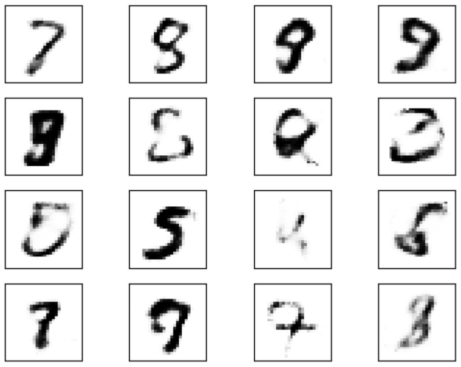

Figure 4: VAE generated images after 200 epochs

2.2

The first term describes maximizing the likelihood of D classifying an image as real that is in fact a real image from the dataset. The second term describes miniziming the likelihood $(1 - ...)$ of the discriminator D classifying an image as real $(D(...))$ that

2.3

We can find this information on the slides:

$$D(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

At convergence $p_g(x) = p_r(x)$, so above equation becomes:

$$D(x) = \frac{p_r(x)}{2p_r(x)} = \frac{1}{2}$$

This leaves us with Loss at convergence:

$$L(G, D) = -2log2$$

2.4

This is problematic because at the start of the training the generator will have troubles generating good fake images. This means the discriminator will easily identify these images and the gradient for the generator will vanish.
This can be solved using the Heuristic non-saturating game. The generator no longer does not care about the real images. The generator loss is now only on generating good fake images.

2.5 Implementation

Unfortunately, I couldn't get the model to train properly. At first I managed to generate some images but all generated images were very similar and did not appear to be digits:

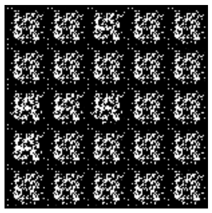

Figure 5: Example of bad gan generated image after 150 epochs

After trying to fix it the generator came only up with black images.

3 Flow-based models

3.1

3.2

The hidden layers need to the have the same dimensions. This way we can ensure that we have a quadratic weight matrix that is invertible.

3.3

3.4

The problem lies in that we are trying to model a discrete input space over a continuous density model. Any direct solution of this modeling is invalid as the probability masses will be accumulated on discrete datapoints. We can solve this issue by dequantisizing the input data into a continuous distribution with which it is then possible to model a continuous density model [2]. Ho et al. suggest two options for dequantization: the uniform dequantization or the variational dequantization.

4

As I didn't have the time to implement the GAN and flow based models correctly, I cannot compare the models. The VAE came up with acceptable images but not all generated images contained identifyable digits. I would expect the GAN to come up with more realistic looking digits as that is after all the direct training objective. However, the GAN might have trouble coming up with covering all relevant digits (if it comes up with one image that it can use to perfectly fool the decoder it might always resort to that image).

References

[1] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[2] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.