Deep Learning Practical 3

Longxiang Wang
Master of Artificial Intelligence
University of Amsterdam
`wlongxiang1119@gmail.com`

1 Variational auto encoders

Q1.1: How does the VAE relate to a standard autoencoder?

a) Are they different in terms of their main function or intended purpose? How so?

In terms of functionality, both autoencoder and VAE are used to learn a dense representation of data which is sparse originally. In that sense, they are no different. However, autoencoder learns a deterministic representation with the goal of minimizing reconstructing loss. Thus it does not have a probabilistic interpretation and it only allows us to reconstruct original inputs, which does not have many practical applications. But VAE takes a bayesian approach and learns a joint distribution which not only has a probabilistic interpretation and also allows us to sample and generate new data.

b) A VAE is generative. Can the same be said of a standard autoencoder? Why or why not?

No, a standard autoencoder is not generative, as it does not learn the underlying distribution of the data. It only learns a deterministic dense representation.

Q1.2: Describe the procedure to sample from such a model. (Hint: ancestral sampling)

In ancestral sampling we suppose that each node has a higher number than any of its parents nodes. Thus, we can start sampling from the lowest number node (root node), and then we walk through nodes in numbering order until we reach out target node. In this case, z_n is the root node which follows a standard normal distribution which we can sample $\hat{z_n}$ from. Then with the sample from root node, we can derive the mean of Bernoulli distribution, and sample the conditional distribution to obtain the sample of $\hat{x_n}$.

Q1.3: Why the latent space standard distribution is not a restrictive assumption?

As said in Carl Doersch's tutorial, the key is to notice that any distribution in d dimensions can be generated by taking a set of d variables that are normally distributed and mapping them through a sufficiently complicated function. In the VAE model, we can sample d variables from the standard distribution then passing through a complicated neural network denoted by f_θ, which gives the model the power to approximate any distribution.

Q1.4: Write down approximation of $logp(x_n)$

$$logp(x_n) \approx \frac{1}{S}\sum_{i=1}^{S} p(x_n|z_i) \tag{1}$$

Using the sampling approach mentioned previously, we can sample a large number of samples S from the z, then get the mean of the conditional distribution.

However, in practice this approach is very inefficient. As the figure shows, the conditional distribution $p(x_n|z_n)$ might lie in a small region, which is mostly likely the case due to the difficulties of acquiring large number of data in reality. therefore, When we get samples from $p(z_n)$, we will need to have a very large of samples to effectively estimate the integral, since most of samples will have $p(x_n|z_n)$ close to 0. This situation get worse as the dimension of $p(z_n)$ increases, as for each dimension d of z_n we need to sample a variable.

Q1.5: KL divergence

We can get the smallest divergence when two distributions are almost identical everywhere. Therefore, when $\mu_q = 0, \sigma_q^2 = 1$, we have the smallest KL divergence.

We can get the largest divergence when two distributions are infinitely far from each other. Therefore, when $\mu_q = \infty, \sigma_q^2 \to 0$, we have the largest KL divergence.

According to the link given in the assignment, the KL divergence between 2 univariate Gaussian is given by:

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

Q1.6: Lower bound of log probability

Since KL divergence are non-negative, we have:

$$D_{KL}(q(Z|x_n)||p(Z|x_n)) \geq 0 \tag{3}$$

Therefore, according to equation 11, the right hand side is the lower bound $logp(x_n)$.

Q1.7: Why not optimize log probability directly?

From euqation 11, we see that if we want to optimize the log probability directly, we will need to calculate $D_{KL}(q(Z|x_n)||p(Z|x_n))$, which means we need to calculate the posterior distribution. According to Bayes rule, posterior distribution is expensive to calculate, because that normally involves treating an integral of marginal distribution of $p(x) = \int p(z)p(x|z)dz$.

Instead, we can optimize the more tractable lower bound.

Q1.8: What happens when lower bound pushes up?

From equation 11, we can derive that if lower bound increases, the $logp(x)$ increases and $D_{KL}(q(Z|x_n)||p(Z|x_n))$ decreases. The former is similar to the effect of max likelihood, which tries to find a distribution that assigns maximal probability to observational data.The decrease of KL divergence optimize the approximated distribution to be close to the underlying true posterior distribution.

Q1.9: Explain reconstruction and regularization loss used in VAE

Minimizing the construction loss is equivalent to maximizing the the log likelihood. For a sigle sample when $N = 1$, we have

$$\mathcal{L}^{recon} = -q_{\phi(z|x)} logp_\theta(x|z) \tag{4}$$

This loss takes the same form as a cross entropy loss, whereas it is the cross entropy loss between the encoder and decoder. Therefore, while minimizing the loss, we are also minimizing the reconstruction error.

For the regularization term, it is actually the KL divergence between the posterior distribution and the prior distribution of the latent variable. While minimizing the KL divergence, we are penalizing the posterior distribution to drift too far away from the prior. In other words, it is preventing the model to overfit on the observed data x.

Q1.10: Write down the loss function form of VAE

The loss function comprises two parts as explained previously, the reconstruction loss and the regularization loss.

For reconstruction loss, we can approximate it by sampling as follows:

$$\mathcal{L}^{recon} = -q_{\phi(z|x)} log p_{\theta(x|z)} \approx -\frac{1}{S}\sum_{i=1}^{S} log p_{\theta}(x|z) = -\frac{1}{S}\sum_{i=1}^{S} log \prod_{m=1}^{M} Bern(x^m|\mu^m) \tag{5}$$

Where the Bernoilli mean μ^m can be obtained by a forward pass of input x.

For the regularization loss, we have seen how to calculate the KL divergence for two univariate Gaussian distributions. Here we extend it to multivariate case, and obtained the following:

$$\mathcal{L}^{reg} = \sum_{i=1}^{D} \big(-\log \sigma_{qi} + \frac{\sigma_{qi}^2 + \mu_{qi}^2}{2} - \frac{1}{2} \big) \tag{6}$$

Q1.11: Write down the loss function form of VAE

$\nabla_{\phi}\mathcal{L}$ represents the gradient of loss w.r.t. to parameters of the encoder network. This is required because:

- We need to learn the encoder network in order to minimize the reconstruction loss
- We need to learn the encoder network to approximate the KL divergence.

However, due to the sampling from encoder probabilistic distribution, it makes the loss not differentiable anymore on the parameters of the encoder.

We can circumvent the non-differentiable difficulty introduced by sampling by using the so-called *reparametrization trick*. We first sample an initial seed ϵ from a standard distribution, then compute $z = \mu(x) + \Sigma^{\frac{1}{2}}(x) * \epsilon$. Basically by moving the sampling step out of the backpropagation path, we are able to achieve the same effect of sampling and have a continuous and differentiable space.

Q1.12: VAE implementation in Pytorch

By following Kingma paper, MLPs with Gaussian outputs are used as probalistic encoders (see C.2 section of the paper). More specifically,firstly inputs are passed through a linear layer then a tanh non-linearity, then a linear layer is used for calculate the mean of Gaussian output, and another linear layer is used for generating log variances.

In the decoder part, MLPs with Bernoulli outputs are used as probabilistic decoder. Firstly inputs are passed through a linear layer then a tanh non-linearity. Then another linear layer is used for calculate the mean of Bernoulli output, and the output is squashed to betwween 0 and 1 by sigmoid function.

A binary cross entropy loss function is used to calculate the reconstruction loss.

Q1.13: VAE model training

As shown in 1, the training loss is decreasing as expected, as also the evaluation loss. It is also noticed that the model is still improving slowly and no overfitting is happening yet, which shows that the regularization term is working as expected.

Q1.14: VAE model results

In Figure 2, I have shown samples generated at different stages from the encoder (used as generator in VAE). As we can see, initially the samples are random noise, then samples are getting more and more similar to its training data as ELBO goes down.

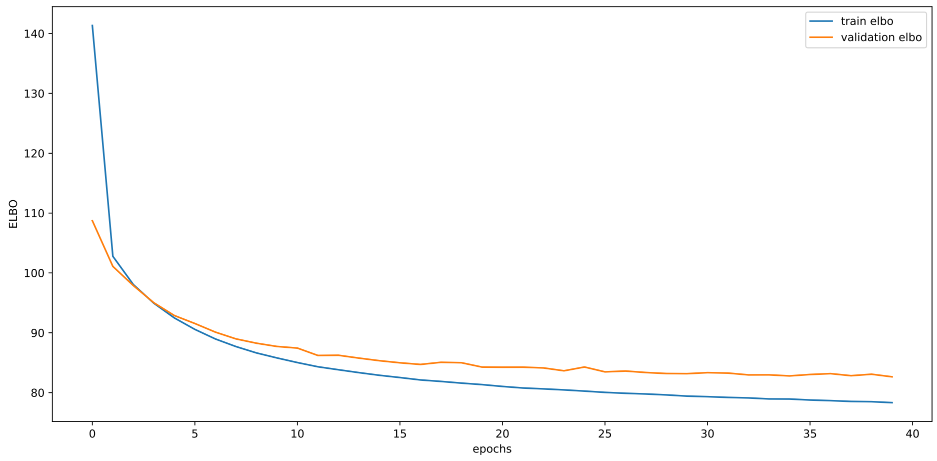
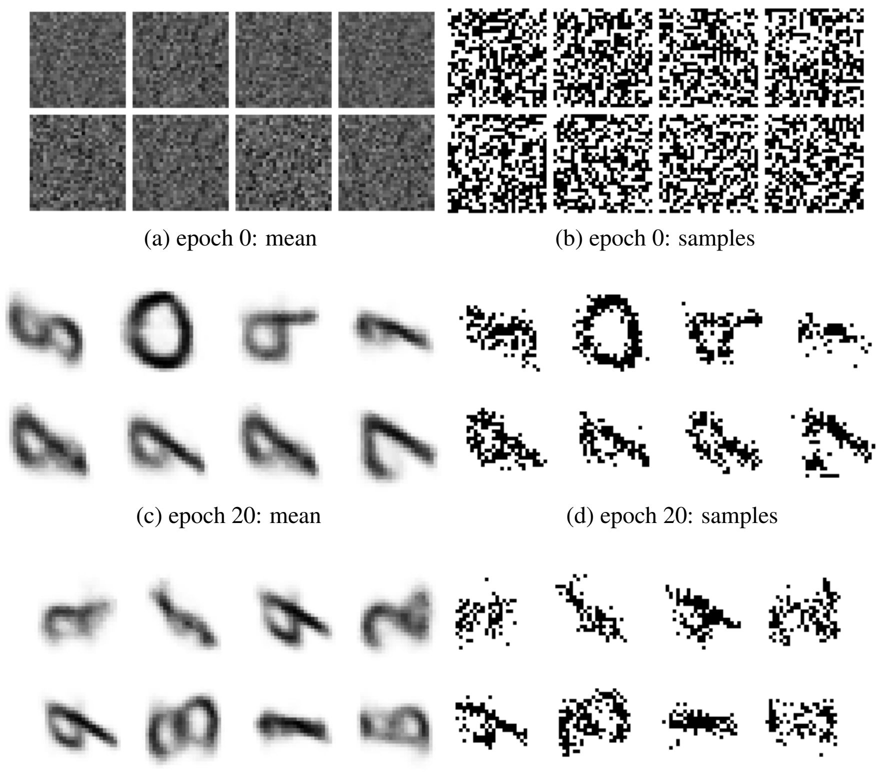


Figure 1: Training VAE


(a) epoch 0: mean (b) epoch 0: samples

(c) epoch 20: mean (d) epoch 20: samples

(e) epoch 40: mean (f) epoch 40: samples

Figure 2: Samples and mean at different epochs

Q1.15: Latent space visualization

By setting the latent space Z's dimension to 2, we are able to visualize what it is actually encoded in the latent space. As demonstrated by Figure 3. It is not surprising to see that the latent space encodes the structure of digits from the training data.

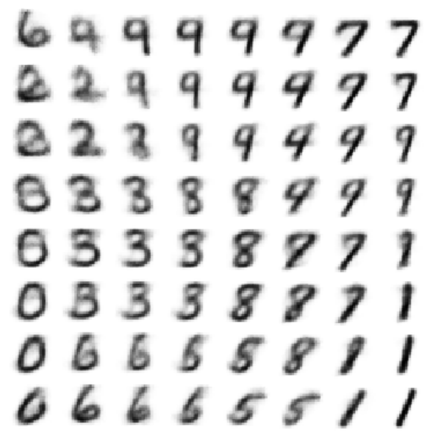


Figure 3: Manifold of 2D latent space

2 Generative Adversarial Networks

Q2.1: Inputs and outputs to generator and discriminator

- The generator is like the decoder part of VAE, the input is a random noise, then it passes through a learned network which produces a output mimicking the training inputs.
- The input to the discriminator is either the training data or the output from the generator, and the output is a probability to classify if the input comes from the training data or synthesized by the generator.

Q2.2: Inputs and outputs to generator and discriminator

By maximizing the first term $\mathbb{E}_{p_{data}(x)}[logD(X)]$, we are optimizing the discriminator to maximize the likelihood of training data, which will push the discriminator to output higher probability for training images.

By minimizing the second term $\mathbb{E}_{p_z(z)}[log(1 - D(G(X)))]$, which represents the log likelihood of generated images. This will push the generator to generator more realistic samples such that $D(G(X))$ is closer to 1.

Q2.3: Optimal value of GAN objective function

In its perfect form, the discriminator is totally "fooled" such that it cannot distinguish a real image or a generated image at all. Then it assigns probability of 0.5 to both terms, so objective function becomes $2log\frac{1}{2}$.

Q2.4: Problems of GAN objective function at early stage

The biggest problem is the original objective is that it is hard to bootstrap the generator. In other words, at the early stage, the generator has not learned anything, thus it is not able to generate good enough images, and the discriminator can distinguish it from real images very easily. In that way, the second term will be very close to 0 at the beginning. This makes it difficult to learn the generator.

We can rewrite the objective function into an equivalent max-max objective:

$$\max_D \max_G \mathbb{E}_{p_{data}(x)}[logD(X)]\mathbb{E}_{p_z(z)}[logD(G(X))] \tag{7}$$

In terms of objective the max-max game is equivalent to the min-max game in its original form. But now, the second term has larger gradients to update the the weights of generator.

Q2.5: Implementing GAN

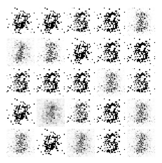
The GAN consists of two parts, one generator network and a discriminator network. The training process, especially the construction of the loss is also very important. The dataset used here in the MNIST.

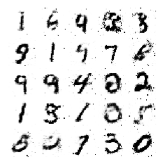
The generator network starts with a "Linear + LeakyReLu" module, then 3 modules of "Linear + BatchNorm1D + LeakyReLu" and another "Linear" layer and a "Tanh" layer in the end. Note that the choice of "Tanh" non-linearity is to make the output range in the training pixel normalized range $[-1, 1]$.

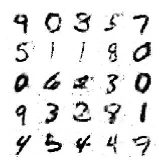
The discriminator network has 2 modules of "Linear + LeakyReLu" at the beginning, then followed by another "Linear" layer and a "Sigmoid" layer. The choice of sigmoid is to make the calculation of cross entropy loss easier.

The generator network and discriminator network are trained jointly with different optimizers and loss functions. The sampling from the generator happens continuously during the training process and its outputs are used as inputs to the losses.

Q2.6: GAN results


(a) epoch 5


(b) epoch 50


(c) epoch 100

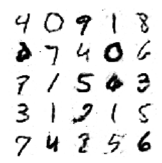

(d) epoch 200

Figure 4: Samples generated at different epoch

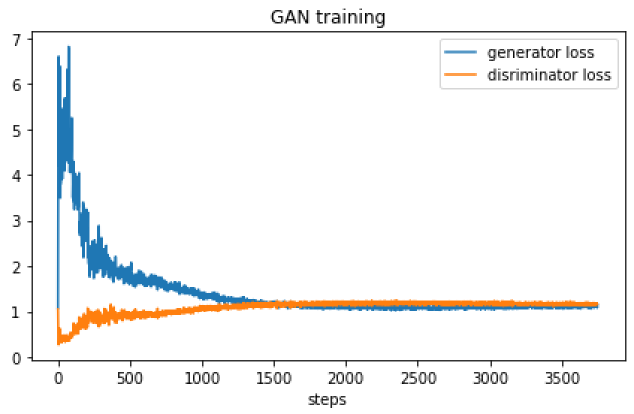


Figure 5: Gan training losses

Q2.7: Interpolation

3 Generative Normalizing Flows

Q3.1: Change of variables for multivariate

For a function f of $\mathbb{R}^m \to \mathbb{R}^m$, according to the change of variable rule:

$$p(\boldsymbol{x}) = p(\boldsymbol{z}) \left| detJ(\frac{\partial f}{\partial \boldsymbol{z}}) \right| \tag{8}$$

where the $\left| detJ(\frac{\partial f}{\partial \boldsymbol{z}}) \right|$ denote absolute value of the determinant of Jacobian, $\boldsymbol{x}$ and $\boldsymbol{z}$ are in $\mathbb{R}^m$, and the Jocobian is a $m \times m$ matrix with the m-th representing $[\frac{\partial f_m}{\partial z_1}, \frac{\partial f_m}{\partial z_2}, \cdots, \frac{\partial f_m}{\partial z_m}]$.

If we take the log on both sides and apply it recursively:

$$\log p(\boldsymbol{x}) = p(\boldsymbol{z_0}) + \sum_{l=1}^{L} \log \left| Jacobian(\frac{\partial f_l}{\partial \boldsymbol{z_{l-1}}}) \right| \tag{9}$$

Q3.2: Theoretical constrains on function mapping

From the above analysis, we see that we need to computer the determinant of a Jacobian in order to complete the invertible mapping. Determinant only exisists for ssquare matrix which requires the Jacobian to be a suqare matrix, hence $\boldsymbol{x}$ and $\boldsymbol{z}$ need to have the same dimensions.

Q3.3: Computation difficulties

As mentioned previsouly, in the flow based equation, we need to tackle a $m \times m$ Jacobian matrix, which is an expensive operation with polynomial order of complexity, and it gets even worse when the dimension of latent space increases.

Q3.4: Problems with discrete data

Many real words data such as images are discrete data quantized from continuous data. When we fit a continuous density model to its discrete domain, the model tends to assign all mass probability to discrete data points. A common solution to this is just to convert the discrete data back to a continuous space by adding random noise.

Q3.6: Implementing normalizing flow model

The NF model contains two parts: a) inference network; b) generative network. The inference network receives training data, and maps observations through the parameters of the flow. The generative model receives posterior samples during training time.

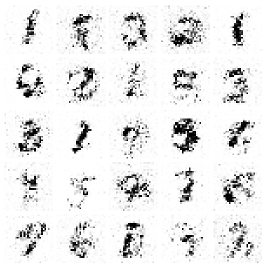
The core component in NF model is the carefully designed flow functions which are both invertible and easy to compute the determinant of Jacobian.

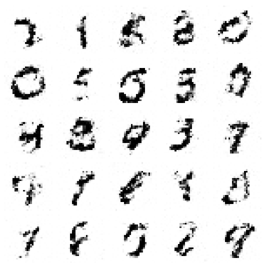
During training, we continuous get batches of observations from training data and apply successively to the log form of flow models. Then we use any SGD algorithm to update the parameters of the network until the model has converged.

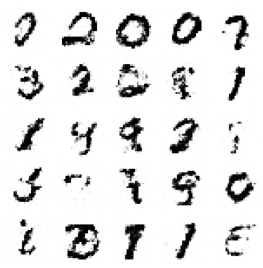
After training, we can sample from a normal distribution, and apply the invertible flow model successively.

Q3.7: NF model results

The implementation is included in the code.


(a) epoch 1


(b) epoch 10


(c) epoch 30

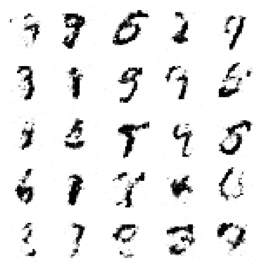

(d) epoch 40

Figure 6: Samples generated at different epoch for VF model

Q3.8: NF training and validation performance

As shown in Figure 7, the *bits per dimension* converges at around 1.85.

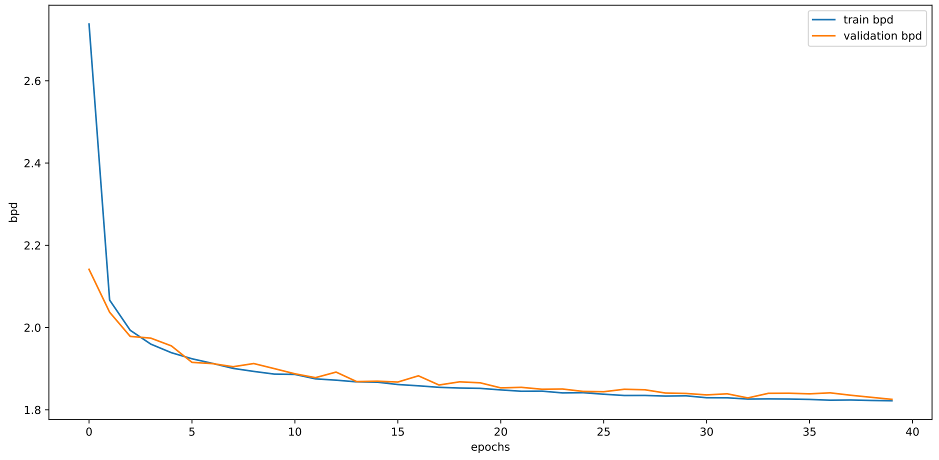


Figure 7: Training NF model

4 Conclusion

Q4.1: Summary

In this practical, neural network based generative models are explored in details, namely VAEs, GANs and Normalizing Flow. We first start with VAE models which use variational bayesian approach to approximate an intractable posterior distribution. It achieves so by optimizing the lower bound of the log-likelihood. VAE also uses a probabilistic encoder-decoder architecture. Therefore, it learn a lower dimensional latent representation, which can enable visualization and more explainability of the data.

Then we look at GANs. GAN has a very intuitive network architecture, but it is not easy to train due to the joint somehow competing joint training of a generator and a disriminator together. Unlike VAEs, GANs does not concern directly about modeling the latent space as it does not has an encoder part in it.

NF models try to address one of the limitations of variational inference that one has to choose some distribution for approximating the intractable posterior distribution. Instead it uses a flow model where we can start with a simple Gaussian prior and pass it through a flow of invertible transformations to approximate any complex distribution.

All these models has their merits and drawbacks. Generative models are still in active research with exciting progresses happening fast.