

Assignment 3. Deep Generative Models

University of Amsterdam – Deep Learning Course

October 9, 2018

The deadline for this assignment is October 19th at 23:59.

In this assignment you will study and implement Deep Generative Models. Deep generative models come in many flavors, but all represent the probability distribution (of the data) in some way or another. Some generative models allow for explicit evaluation of the log likelihood, whereas other may only support some function that relies on the probability distribution—such as sampling. In this assignment, we focus on two—currently most popular—deep generative models, namely Variational Auto Encoders (VAE) [2] and Generative Adversarial Networks (GAN) [1]. You will implement both in PyTorch as part of this assignment. Note that, although this assignment does contain some explanation on the model, we do not aim to give a complete introduction in this assignment. The best source for understanding the model are the papers that introduced them [1, 2], the hundreds of blogpost that have been written on them ever since, or the lecture.

Note: for this assignment you are *not* allowed to use the `torch.distributions` package. Moreover, try to stay as close as your can to the template files provided as part of the assignment.

1 Variational Auto Encoders

A VAE is a latent variable model that leverages the flexibility of Neural Networks (NN) in order to learn/specify a latent variable model. We will first briefly discuss Latent Variance Models and then dive into VAEs.

1.1 Latent Variable Models

A latent variable model is a statistical model that contains both observed and unobserved (or latent) variables. Assume a dataset $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^N$, where $\mathbf{x}_n \in \{0, 1\}^M$. For example, \mathbf{x}_n can be the pixel values of a binary image. The simplest latent variable model for this data we can think of is shown in Figure 1, which we can also summarize with the following generative story:

$$\mathbf{z}_n \sim \mathcal{N}(0, \mathbf{I}_D) \quad (1)$$

$$\mathbf{x}_n \sim p_X(f_\theta(\mathbf{z}_n)) \quad (2)$$

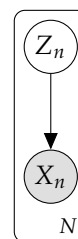


Figure 1. Graphical model of VAE

where f_θ is some function — parameterized by θ — that maps \mathbf{z}_n to the parameters of a distribution over \mathbf{x}_n . For example, if p_X would be a Gaussian distribution we will use $f_\theta : \mathbb{R}^D \rightarrow (\mathbb{R}^M, \mathbb{R}_+^M)$, or if p_X is a product of

Bernoulli distributions, we have $f_\theta : \mathbb{R}^D \rightarrow [0, 1]^M$. Here, D denotes the dimensionality of the latent space. Note that our dataset \mathcal{D} does not contain \mathbf{z}_n , hence \mathbf{z}_n is a latent (or unobserved) variable in our statistical model. In the case of a VAE, a (deep) NN is used for $f_\theta(\cdot)$.

Question 1.1 (10 points)

How does the VAE relate to probabilistic PCA (pPCA) or Factor Analysis (FA)? You are not required to discuss the difference between pPCA and FA. Answer in 2–3 sentences.

1.2 Decoder: The Generative Part of the VAE

In the previous section a general graphical model for VAEs was given. In this section we will define a more specific generative model that we will use throughout this assignment. This will later be referred to as the *decoding* part (or decoder) of a VAE. For this assignment we will assume the pixels of our images \mathbf{x}_n in \mathcal{D} are Bernoulli(p) distributed.

$$p(\mathbf{z}_n) = \mathcal{N}(0, \mathbf{I}_D) \quad (3)$$

$$p(\mathbf{x}_n | \mathbf{z}_n) = \prod_{m=1}^M \text{Bern}(\mathbf{x}_n^{(m)} | f_\theta(\mathbf{z}_n)_m) \quad (4)$$

where $\mathbf{x}_n^{(m)}$ is the m th pixel of the n th image in \mathcal{D} , and $f_\theta : \mathbb{R}^D \rightarrow [0, 1]^M$ is a Neural Network parameterized by θ that outputs the mean of the Bernoulli distributions for each pixel in \mathbf{x}_n .

Question 1.2 (5 points)

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

Question 1.3 (10 points)

This model makes a very simplistic assumption about $p(\mathbf{Z})$. i.e. It assumes our latent variables follow a standard-normal distribution. Note that there is no trainable parameter in $p(\mathbf{Z})$. Describe why, due to the nature of Equation 4, this is not such a restrictive assumption in practice. (Hint: See Figure 1 and the accompanying explanation in [Carl Doersch's tutorial](#))

Now that we have defined the model, we can write out an expression for the log probability of the data \mathcal{D} under this model:

$$\begin{aligned} \log p(\mathcal{D}) &= \sum_{n=1}^N \log p(\mathbf{x}_n) \\ &= \sum_{n=1}^N \log \int p(\mathbf{x}_n | \mathbf{z}_n) p(\mathbf{z}_n) d\mathbf{z}_n \\ &= \sum_{n=1}^N \log \mathbb{E}_{p(\mathbf{z}_n)} [p(\mathbf{x}_n | \mathbf{z}_n)] \end{aligned} \quad (5)$$

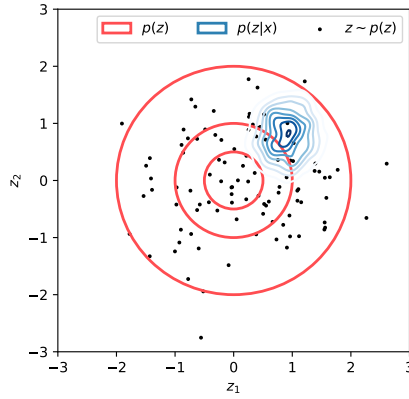


Figure 2. Plot of 2-dimensional latent space and contours of prior and posterior distributions.

Question 1.4 (15 points)

- Evaluating $\log p(\mathbf{x}_n) = \log \mathbb{E}_{p(\mathbf{z}_n)} [p(\mathbf{x}_n|\mathbf{z}_n)]$ involves an intractable integral. However, equation 5 hints at a method for approximating it. Write down an expression for approximating $\log p(\mathbf{x}_n)$ by sampling. (Hint: you can use Monte-Carlo Integration).
- Although this approach can be used to approximate $\log p(\mathbf{x}_n)$, it is not used for training VAE type of models, because it is inefficient. In a few sentences, describe why it is inefficient and how does this efficiency scale with the dimensionality of \mathbf{z} . (Hint: you may use Figure 2 in you explanation.)

1.3 The Encoder: $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$

In the previous section we have developed the intuition that we only want to sample \mathbf{z}_n for which $p(\mathbf{z}_n|\mathbf{x}_n)$ is not close to zero — in order to compute the Monte-Carlo approximation. One approach is to simply sample from $p(\mathbf{z}_n|\mathbf{x}_n)$ instead of sampling from $p(\mathbf{z}_n)$. However, sampling from (or obtaining an analytical form of) $p(\mathbf{z}_n|\mathbf{x}_n)$ is intractable in the case of VAE. Instead, we can solve this by sampling from a *variational distribution* $q(\mathbf{z}_n|\mathbf{x}_n)$, which we use to approximate the (intractable) posterior $p(\mathbf{z}_n|\mathbf{x}_n)$. One way to see if two distributions are close to each other is the Kullback-Leibner divergence (KL-divergence):

$$D_{\text{KL}}(q||p) = -\mathbb{E}_{q(x)} \left[\log \frac{p(X)}{q(X)} \right] = - \int q(x) \left[\log \frac{p(x)}{q(x)} \right] dx, \quad (6)$$

where q and p are probability distributions in the space of some random variable X .

Question 1.5 (10 points)

Assume that q and p in Equation 6, are univariate gaussians: $q = \mathcal{N}(\mu_q, \sigma_q^2)$ and $p = \mathcal{N}(0, 1)$.

- (a) Give two examples of (μ_q, σ_q^2) : one of which results in a very small, and one of which has a very large, KL-divergence: $D_{\text{KL}}(q||p)$
- (b) Find the (closed-form) formula for $D_{\text{KL}}(q||p)$.

(You do not need to derive it, you can just search for it, but feel free to derive it if that brings you joy. Hint: **This will be helpful.**) You will need to use this answer later.

Now, if we write out the expression for the KL-divergence between our proposal $q(\mathbf{z}_n|\mathbf{x}_n)$ and our posterior $p(\mathbf{z}_n|\mathbf{x}_n)$, we can derive an expression for the probability of our data under our model:

$$D_{\text{KL}}(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)) = -\mathbb{E}_{q(z|\mathbf{x}_n)} \left[q(Z|\mathbf{x}_n) \log \frac{p(Z|\mathbf{x}_n)}{q(Z|\mathbf{x}_n)} \right] \quad (7)$$

$$= -\mathbb{E}_{q(z|\mathbf{x}_n)} \left[q(Z|\mathbf{x}_n) \log \frac{p(\mathbf{x}_n|Z)p(Z)}{q(Z|\mathbf{x}_n)p(\mathbf{x}_n)} \right] \quad (8)$$

$$= -\mathbb{E}_{q(z|\mathbf{x}_n)} \left[q(Z|\mathbf{x}_n) \left(\log \frac{p(Z)}{q(Z|\mathbf{x}_n)} + \log p(\mathbf{x}_n|Z) - \log p(\mathbf{x}_n) \right) \right] \quad (9)$$

$$= D_{\text{KL}}(q(Z|\mathbf{x}_n)||p(Z)) - \mathbb{E}_{q(z|\mathbf{x}_n)} [\log p(\mathbf{x}_n|Z)] + \log p(\mathbf{x}_n) \quad (10)$$

Hence,

$$\log p(\mathbf{x}_n) - D_{\text{KL}}(q(Z|\mathbf{x}_n)||p(Z|\mathbf{x}_n)) = \mathbb{E}_{q(z|\mathbf{x}_n)} [\log p(\mathbf{x}_n|Z)] - D_{\text{KL}}(q(Z|\mathbf{x}_n)||p(Z)) \quad (11)$$

We have arranged the equation above so that directly-computable quantities are on the right-hand side. The right side of the equation is referred to as the *lower bound* on the log-probability of the data. This is what will optimize. So, just as we previously defined our loss to be the mean negative log likelihood (or squared error) over samples, we now define our loss as the mean negative lower bound:

$$\mathcal{L}(\theta, \phi) = -\frac{1}{N} \sum_{n=1}^N \mathbb{E}_{q_\phi(z|\mathbf{x}_n)} [\log p_\theta(\mathbf{x}_n|Z)] - D_{\text{KL}}(q_\phi(Z|\mathbf{x}_n)||p_\theta(Z)) \quad (12)$$

Note that we make an explicit distinction between the generative parameters θ and the variational parameters ϕ .

Question 1.6 (10 points)

Why is the right-hand-side of Equation 11 called the *lower bound* on the log-probability?

Question 1.7 (10 points)

Looking at Equation 11, why must we optimize the lower-bound, instead of optimizing the log-probability directly?

Question 1.8 (10 points)

Now, looking at the two terms on left-hand side of **11**: Two things can happen when the lower bound is pushed up. Can you describe what these two things are?

1.4 Specifying the encoder $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$

In VAE, we have some freedom to choose the distribution $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$. In essence we want to choose something that can closely approximate $p(\mathbf{z}_n|\mathbf{x}_n)$, but we are also free to select a distribution that makes our life easier. We will do exactly that in this case and choose $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$ to be a factored multivariate normal distribution, i.e.,

$$q_\phi(\mathbf{z}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{z}_n|\mu_\phi(\mathbf{x}_n), \text{diag}(\Sigma_\phi(\mathbf{x}_n))), \quad (13)$$

where $\mu_\phi : \mathbb{R}^M \rightarrow \mathbb{R}^D$ maps an input image to the mean of the multivariate normal over \mathbf{z}_n and $\Sigma_\phi : \mathbb{R}^D \rightarrow \mathbb{R}_{>0}^M$ maps the input image to the diagonal of the covariance matrix of that same distribution. Moreover, $\text{diag}(\mathbf{v})$ maps a K -dimensional (for any K) input vector \mathbf{v} to a $K \times K$ matrix such that for $i, j \in \{1, \dots, K\}$

$$\text{diag}(\mathbf{v})_{ij} = \begin{cases} v_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}. \quad (14)$$

Question 1.9 (10 points)

The loss in Equation **12**:

$$\mathcal{L}(\theta, \phi) = -\frac{1}{N} \sum_{n=1}^N \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_n)} [\log p_\theta(\mathbf{x}_n|Z)] - D_{\text{KL}}(q_\phi(Z|\mathbf{x}_n) || p_\theta(Z))$$

can be rewritten in terms of per-sample losses:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N (\mathcal{L}_n^{\text{recon}} + \mathcal{L}_n^{\text{reg}}),$$

where

$$\begin{aligned} \mathcal{L}_n^{\text{recon}} &= -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_n)} [\log p_\theta(\mathbf{x}_n|Z)] \\ \mathcal{L}_n^{\text{reg}} &= D_{\text{KL}}(q_\phi(Z|\mathbf{x}_n) || p_\theta(Z)) \end{aligned}$$

can be seen as a reconstruction loss term and an regularization term, respectively. Explain why the names *reconstruction* and *regularization* are appropriate for these two losses.

(Hint: suppose we use just one sample to approximate the expectation $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_n)} [p_\theta(\mathbf{x}_n|Z)]$ — as is common practice in VAEs.)

Question 1.10 (15 points)

Now we have defined an objective (Equation 12) in terms of an abstract model and variational approximation, we can put everything together using our model definition (Equation 1 and 2) and definition of $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$ (Equation 13), we can write down a single objective which we can minimize.

Write down expressions (including steps) for $\mathcal{L}_n^{\text{recon}}$ and $\mathcal{L}_n^{\text{reg}}$ such that we can minimize $\mathcal{L} = \sum_{n=1}^N \mathcal{L}_n^{\text{recon}} + \mathcal{L}_n^{\text{reg}}$ as our final objective. Make any approximation explicit.

(Hint: look at your answer for question 1.5 for $\mathcal{L}_n^{\text{reg}}$.)

1.5 The Reparametrization Trick

Although we have written down (the terms of) an objective in question 1.10, we still cannot simply minimize this by taking gradients to θ and ϕ . This is due to the fact that we sample from $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$ to approximate the $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_n)}[\log p_\theta(\mathbf{x}_n|Z)]$ term. Yet, we need to pass the derivative through these samples if we want to compute the gradient of the encoder parameters, i.e., $\nabla_\phi \mathcal{L}(\theta, \phi)$.

Question 1.11 (10 points)

Read and understand Figure 4 from [the tutorial by Carl Doersch](#). In a few sentences each, explain why:

- (a) we need $\nabla_\phi \mathcal{L}$
- (b) the act of sampling prevents us from computing $\nabla_\phi \mathcal{L}$
- (c) What the *reparametrization trick* is, and how it solves this problem.

1.6 Putting things together: Building a VAE

Given everything we have discussed before, we now have an objective (the *evidence lower bound* or ELBO) and a way to backpropagate to both θ and ϕ (i.e., the reparametrization trick). As such, we can now implement a VAE in PyTorch. You are free to make your own choices with respect to the architecture, but we advice you to follow the choices presented in [2].

Question 1.12 (40 points)

Build a Variational Autoencoder in PyTorch, and train it on the Binary MNIST data. Start with the template in `a3_vae_template.py` in the code directory for this assignment.

Following standard practice — and for simplicity — you may assume that the number of samples used to approximate the expectation in $\mathcal{L}_n^{\text{recon}}$ is 1.

Provide a short (no more than ten lines) description of your implementation.

You will get full points on this question if your answers to the following questions indicate that you successfully trained the VAE and your description is sufficient, however, do not forget to include your code in your final submission.

Question 1.13 (0 points)

Plot the estimated lower-bounds of your training and validation set as training progresses — using a 20-dimensional latent space.

(Hint: Think about what reasonable average elbo values are for a VAE with random initialized weights (i.e., the predicted mean of the output distribution is random) to make sure you compute the correct lower bound over multiple batches)

Question 1.14 (5 points)

Plot samples from your model at three points throughout training (before training, half way through training, and after training). You should observe an improvement in the quality of samples.

Question 1.15 (5 points)

Train a VAE with a 2-dimensional latent space (or encoding). Use this VAE to plot the data *manifold* as is done in Figure 4b of [2]. This is achieved by taking a a two dimensional grid of points in Z -space, and plotting $f_\theta(Z) = \mu|Z$. Use the percent point function (ppf, or the inverse CDF) to cover the part of Z -space that has significant density.

2 Generative Adversarial Networks

Generative Adversarial Networks (GAN) are a type of deep generative models. Similar to VAEs, GANs can generate images that mimic images from the dataset by sampling an encoding from a noise distribution. In contrast to VAEs, in vanilla GANs there is no inference mechanism to determine an encoding or latent vector that corresponds to a given data point (or image). Figure 3 shows a schematic overview of a GAN. One thing to notice is that a GAN consists of two separate networks (i.e., there is no parameter sharing, or the like) called the generator and the discriminator. Training a GAN leverages uses an adversarial training scheme. In short, that means that instead of defining a loss function by hand (e.g., cross entropy or mean squared error), we train a network that acts as a loss function. In the case of a GAN this network is trained to discriminate

between real images and fake (or generated) images, hence the name *discriminator*. The discriminator (together with the training data) then serves as a loss function for our *generator* network that will learn to generate images to be similar to those in the training set. Both the generator and discriminator are trained jointly. In this assignment we will focus on obtaining a generator network that can generate images that are similar to those in the training set.

Question 2.1 (10 points)

We can think of the generator and discriminator as functions. Explain the input and output for both. *You do not have to take batched inputs into account. Keep your answer succinct.*

For example: for a normal image classification network the input would be an image and the output a vector $\boldsymbol{\rho}$ where ρ_i indicates the probability of the i th class for the given image.

2.1 Training objective: A Minimax Game

In order to train a GAN we have to decide on a noise distribution $p(\mathbf{z})$, in this case we will use a standard Normal distribution. Given this noise distribution, the GAN training procedure is a minimax game between the generator and discriminator. This is best seen by inspecting the loss (or optimization objective):

$$\min_G \max_D V(D, G) = \min_G \max_D \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \quad (15)$$

Question 2.2 (10 points)

Explain the two terms in the GAN training objective defined in Equation 15.

Question 2.3 (10 points)

What is the value of $V(D, G)$ after training has converged.

Question 2.4 (15 points)

Early on during training, the $\log(1 - D(G(\mathbf{z})))$ term can be problematic for training the GAN. Explain why this is the case and how it can be solved.

2.2 Building a GAN

Not that the objective is specified and it is clear how the generator and discriminator should behave, we are ready to implement a GAN. In this part of the assignment you will implement a GAN in PyTorch.

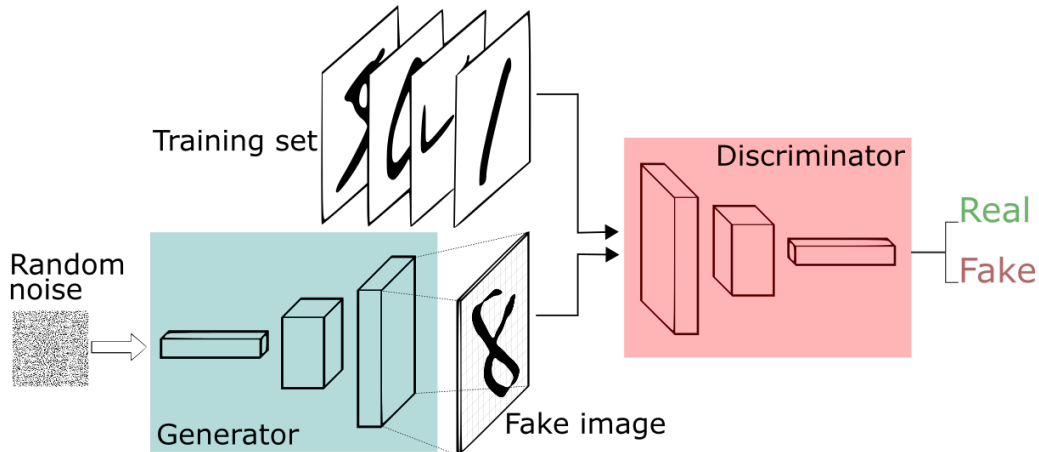


Figure 3. Schematic overview of a GAN. (Taken from <https://skymind.ai/wiki/generative-adversarial-network-gan>)

Question 2.5 (25 points)

Build a GAN in PyTorch, and train it on the MNIST data. Start with the template in `a3_gan_template.py` in the code directory for this assignment.

Provide a short (no more than ten lines) description of your implementation.

You will get full points on this question if your answers to the following questions indicate that you successfully trained the VAE and your description is sufficient, however, do not forget to include your code in your final submission.

Question 2.6 (0 points)

Sample 25 images from your trained GAN and include these in your report (see Figure 2a in [1] for an example). Do this at the start of training, halfway through training and after training has terminated.

Question 2.7 (5 points)

Sample 2 images from your GAN (make sure that they are of different classes). Interpolate between these two digits in latent space and include the results in your report. Use 7 interpolation steps, resulting in 9 images (including start and end point).

3 Conclusion

Question 3.1 (20 points)

Write a short conclusion to your report in which you compare VAEs and GANs. Try to use no more than 10 lines for your conclusion.

Report

We expect each student to write a small report about generative models and make sure to explicitly answering the questions in this assignment. Please clearly mark each answer by a heading indicating the question number. Again, use the NIPS L^AT_EX template as provided here: <https://nips.cc/Conferences/2018/PaperInformation/StyleFiles>.

Deliverables

Create ZIP archive containing your report and all Python code. Please preserve the directory structure as provided in the Github repository for this assignment. Give the ZIP file the following name: `lastname_assignment3.zip` where you insert your lastname. Please submit your deliverable through Canvas. We cannot guarantee a grade for the assignment if the deliverables are not handed in according to these instructions.

The deadline for this assignment is October 19th at 23:59.

References

- [1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 1, 9
- [2] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*, 2014. 1, 6, 7