Assignment 3. Deep Generative Models

University of Amsterdam – Deep Learning Course

May 9, 2019

The deadline for this assignment is May 19$^{\text{th}}$ 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 three—currently most popular—deep generative models, namely Variational Auto Encoders (VAE) [4], Generative Adversarial Networks (GAN) [2] and Flow-Based Models (e.g. RealNVP) [5, 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 [2, 4, 5, 1], 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 $\mathcal{D} = \{\boldsymbol{x}_n\}_{n=1}^N$, where $\boldsymbol{x}_n \in \{0, 1\}^M$. For example, $\boldsymbol{x}_n$ 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:

$$\boldsymbol{z_n} \sim \mathcal{N}(0, \boldsymbol{I}_D) \tag{1}$$
$$\boldsymbol{x_n} \sim p_X(f_\theta(\boldsymbol{z}_n)) \tag{2}$$

where f_θ is some function — parameterized by θ — that maps $\boldsymbol{z}_n$ to the parameters of a distribution over $\boldsymbol{x}_n$. For example, if p_X would

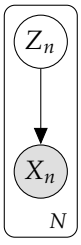

Figure 1. Graphical model of VAE

be a Gaussian distribution we will use $f_\theta : \mathbb{R}^D \to (\mathbb{R}^M, \mathbb{R}_+^M)$, or if p_X is a product of Bernoulli distributions, we have $f_\theta : \mathbb{R}^D \to [0,1]^M$. Here, D denotes the dimensionality of the latent space. Note that our dataset $\mathcal{D}$ does not contain $\boldsymbol{z}_n$, hence $\boldsymbol{z}_n$ 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 a standard autoencoder?

1. Are they different in terms of their main function? How so?
2. A VAE is generative. Can the same be said of a standard autoencoder? Why or why not?
3. Can a VAE be used in place of a standard autoencoder?
4. Generative models need to be able to generalize. What aspect of the VAE missing in the standard autoencoder enables it to be generative.

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 refered to as the *decoding* part (or decoder) of a VAE. For this assignment we will assume the pixels of our images $\boldsymbol{x}_n$ in $\mathcal{D}$ are Bernoulli(p) distributed.

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

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

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

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 $p(Z)$. i.e. It assumes our latent variables follow a standard-normal distribution. Note that there is no trainable parameter in $p(Z)$. 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)

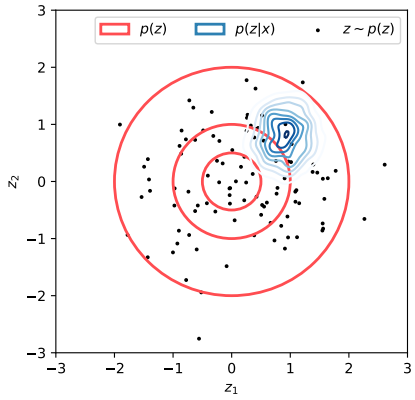

Figure 2. Plot of 2-dimensional latent space and contours of prior and posterior distributions.

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:

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

Question 1.4 (15 points)

(a) Evaluating $\log p(\boldsymbol{x}_n) = \log \mathbb{E}_{p(\boldsymbol{z}_n)} \left[p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right]$ involves an intractable integral. However, equation 5 hints at a method for approximating it. Write down an expression for approximating $\log p(\boldsymbol{x}_n)$ by sampling. (Hint: you can use Monte-Carlo Integration).

(b) Although this approach can be used to approximate $\log p(\boldsymbol{x}_n)$, 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 $\boldsymbol{z}$. (Hint: you may use Figure 2 in you explanation.)

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

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

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

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: $D_{\mathrm{KL}}(q||p)$

(b) Find the (closed-form) formula for $D_{\mathrm{KL}}(q||p)$.

(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 $q(\boldsymbol{z}_n|\boldsymbol{x}_n)$ and our posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$, we can derive an expression for the probability of our data under our model:

$$D_{\mathrm{KL}}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n)) = -\int q(z|\boldsymbol{x}_n) \log \frac{p(z|\boldsymbol{x}_n)}{q(z|\boldsymbol{x}_n)} dz \tag{7}$$

$$= -\int q(z|\boldsymbol{x}_n) \log \frac{p(\boldsymbol{x}_n|z)p(z)}{q(z|\boldsymbol{x}_n)p(\boldsymbol{x}_n)} dz \tag{8}$$

$$= -\int q(z|\boldsymbol{x}_n) \left(\log \frac{p(z)}{q(z|\boldsymbol{x}_n)} + \log p(\boldsymbol{x}_n|z) - \log p(\boldsymbol{x}_n) \right) dz \tag{9}$$

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

Hence,

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

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:

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

Note that we make an explicit distiction 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 $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$

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

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

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

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

Question 1.9 (10 points)

The loss in Equation 12:

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

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

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

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 $\mathbb{E}_{q_\phi(z|\boldsymbol{x}_n)}[p_\theta(\boldsymbol{x}_n|Z)]$ — 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 $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ (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 $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ to approximate the $\mathbb{E}_{q_\phi(z|\boldsymbol{x}_n)}[\log p_\theta(\boldsymbol{x}_n|Z)]$ 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 [4].

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 [4]. 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 mimick images from the dataset by sampling an encoding from a noise distribution. In constract 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 a 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 are 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 $\boldsymbol{\rho}_i$ indicates the probability of the ith 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 $p(\boldsymbol{z})$, in this case we will use a standard Normal distribution. Given this noise distribution, the GAN training

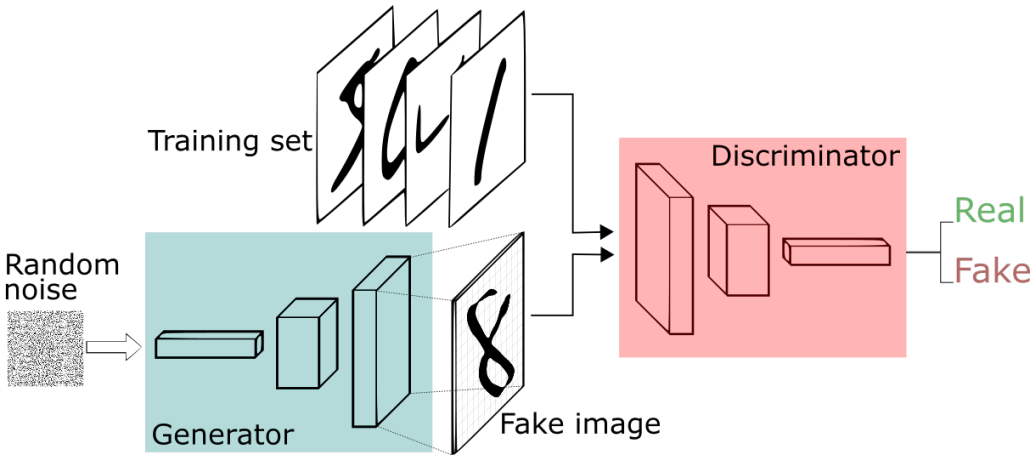

Figure 3. Schematic overview of a GAN. (Taken from https://skymind.ai/wiki/generative-adversarial-network-gan)

procedure is a minimax game between the generator and discriminator. This is best seen by inspecting the loss (or optimization objective):

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

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 $\log(1-D(G(Z)))$ 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.

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 GAN 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 [2] 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 Generative Normalizing Flows

Thus far, two methods for high dimensional density estimation were presented: VAEs and GANs. However, there is another third type of generative models, which is getting more attention in literature recently: Flow-based generative models.

Similar to VAEs, they have a distribution over the latent variables. Unlike VAEs, there is an exact correspondence (not a distribution) between a single datapoint and a latent representation.

Flow-based models utilize normalizing flows. Rezende et al. [5] defines it as follows: "A normalizing flow describes the transformation of a random variable through a sequence of invertible mappings. By repeatedly applying the rule for change of variables, the initial density 'flows' through the sequence of invertible mappings. At the end of this sequence we obtain a valid probability distribution and hence this type of flow is referred to as a normalizing flow."

Idea is very similar to VAEs in that every datapoint can be obtained by sampling a latent variable and then transforming it through sequence of invertible mapping getting actual data distribution in the end. You can see overview of flow-based models in 4. To ensure that the distribution is valid at each timestep, we are using change of variables rule, *which only works if the transformation is* ***invertible***:

$$z = f(x); x = f^{-1}(z); p(x) = p(z)|\frac{df}{dx}| \tag{16}$$

Practically, objectives are optimized in log-likelihood. Let $h_0 = x$ and $z = h_L$, and the other h_l are intermediate representations, then the objective can be written as:

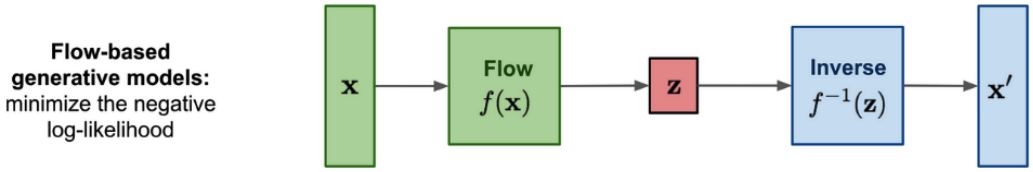

Figure 4. Schematic overview of a Flow-based model. (Taken from https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html)

$$\log p(x) = \log p(z) + \sum_{l=1}^{L} \log \left| \frac{dh_l}{dh_{l-1}} \right| \tag{17}$$

3.1 Change of variables for Neural Networks

In all of the generative models we have looked at so far, we have used Neural Networks to transform random variables, which can be seen as smooth maps between spaces of real numbers. To use them in normalizing flows models, we first need to understand what would equations 16 and 17 look like if $f : \mathbb{R}^m \to \mathbb{R}^m$ is an invertible smooth mapping and $x \in \mathbb{R}^m$ is a multivariate random variable.

Question 3.1 (5 points)

Rewrite equations 16 and 17 for the case when $f : \mathbb{R}^m \to \mathbb{R}^m$ is an invertible smooth mapping and $x \in \mathbb{R}^m$ is a *multivariate* random variable

Question 3.2 (5 points)

What are the constraints that have to be set on the function f to make this equation computable? (Hint: think about the number of dimensions in $\mathbf{z}$ and $\mathbf{x}$)

Question 3.3 (5 points)

Even if we ensure property mentioned in the previous question, what might be a computational issue that arises when you optimize your network using the objective you derived in question 3.1?

Question 3.4 (10 points)

The change-of-variables formula assumes continuous random variables, however, images are often stored as discrete integers. What might be the consequence of that and how would you fix it? (Hint: take a look at [3])

3.2 Building a flow-based model

Now that we have discussed theory behind normalizing flows models, it's time to look at what the practical implementation should look like and, finally, implement a simple flow-based model. Let's start with discussing how to train a model.

Question 3.5 (5 points)

What is the input and output of the flow-based model during training and how is it different during inference time?

Question 3.6 (20 points)

Describe steps you need to take to train a flow-based model during training and inference time.

Now that we have an understanding of how flow-based models work conceptually, it's time to implement one!

Question 3.7 (40 points)

Implement a flow-based generative model. As a reference you may use the template and look at [1].

Question 3.8 (10 points)

Plot your training and validation performance in bits per dimension (negative $\log_2$ likelihood divided by the size of the image). Your model should be able to reach below 1.85 bits per dimension after 40 epochs.

4 Conclusion

Question 4.1 (20 points)

Write a short conclusion to your report in which you compare VAEs, GANs and Flow-based models. Try to use no more than 15 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 LaTeX 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] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *International Conference on Learning Representations, ICLR*, 2017. 1, 11

[2] 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

[3] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *CoRR*, abs/1902.00275, 2019. 10

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*, 2014. 1, 6, 7

[5] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1530–1538. JMLR.org, 2015. 1, 9