Practical 3. Transformers and Deep Generative Models

University of Amsterdam – Deep Learning Course

December 3, 2021

The deadline for this assignment is December 19$^{\text{th}}$ at 23:59.

This assignment contains two parts. First, you will take a closer look at the Transformer architecture. They have been already introduced in Lecture 9 and Tutorial 6, but here, we will discuss more about the theoretical and practical aspects of the architecture.

The second part, which is the main part of the assignment, will be about Deep Generative Models. Modelling distributions in high dimensional spaces is difficult. Simple distributions such as multivariate Gaussians or mixture models are not powerful enough to model complicated high-dimensional distributions. The question is: How can we design complicated distributions over high-dimensional data, such as images or audio? In concise notation, how can we model a distribution $p(\boldsymbol{x}) = p(x_1, x_2, \dots, x_M)$, where M is the number of dimensions of the input data $\boldsymbol{x}$? The solution: Deep Generative Models.

Deep generative models come in many flavors, but all share a common goal: to model the probability distribution of the data. Examples of well-known generative models are Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs). In this assignment, we will focus on VAEs [Kingma and Welling, 2014]. The assignment guides you through the theory of a VAE with questions along the way, and finally you will implement a VAE yourself 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. The best source for understanding the models are the lectures, these papers [Goodfellow et al., 2014, Kingma and Welling, 2014, Rezende and Mohamed, 2015], and the hundreds of blog-posts that have been written on them ever since.

Throughout this assignment, you will see a new type of boxes between questions, namely **Food for thought** boxes. Those contain questions that are helpful for understanding the material, but are not essential and **not required to submit in the report** (no points are assigned to those question). Still, try to think of a solution for those boxes to gain a deeper understanding of the models.

This assignment contains 50 points: 10 on Transformers, and 40 on VAEs. Only the VAE part contains an implementation task at the end.

Note: for this assignment you are not allowed to use the `torch.distributions` package. You are, however, allowed to use standard, stochastic PyTorch functions like `torch.randn` and `torch.multinomial`, and all other PyTorch functionalities (especially from `torch.nn`). Moreover, try to stay as close as your can to the template files provided as part of the assignment.

1 Attention and Transformers **(Total: 10 points)**

In this part, we will discuss theoretical questions with respect to the Transformer architecture. Make sure to check the lecture on Transformers. It is recommended to also check the UvA Deep Learning Tutorial 6 on Transformers and Multi-Head Attention before continuing with this part.

1.1 Attention for Sequence-to-Sequence models

Conventional sequence-to-sequence models for neural machine translation have a difficulty to handle long-term dependencies of words in sentences. In such models, the neural network is compressing all the necessary information of a source sentence into a fixed-length vector. Attention, as introduced by Bahdanau et al. [2015], emerged as a potential solution to this problem. Intuitively, it allows the model to focus on relevant parts of the input, while decoding the output, instead of compressing everything into one fixed-length context vector. The attention mechanism is shown in Figure 1. The complete model comprises an encoder, a decoder, and an attention layer.

Unlike conventional sequence-to-sequence, here the conditional probability of generating the next word in the sequence is conditioned on a distinct context vector c_i for each target word y_i. In particular, the conditional probability of the decoder is defined as:

$$p(y_i|y_1, \ldots, y_{i-1}, x_1, \ldots, x_T) = p(y_i|y_{i-1}, s_i), \tag{1}$$

where s_i is the RNN decoder hidden state for time i computed as $s_i = f(s_{i-1}, y_{i-1}, c_i)$.

The context vector c_i depends on a sequence of annotations $(h_1, \cdots, h_{T_x})$ to which an encoder maps the input sentence; it is computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{2}$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{3}$$

where $e_{ij} = a(s_{i-1}, h_j)$ is an alignment model which scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} (just before emitting y_i, Eq. (1)) and the j-th annotation h_j of the input sentence. The alignment model a is parametrized as a feedforward neural network which is jointly trained with all the other components of the proposed system. Use the Figure 1 to understand the introduced notations and the derivation of the equations above.

1.2 Transformer

Transformer is the first encoder-decoder model based solely on (self-)attention mechanisms, without using recurrence and convolutions. The key concepts for understanding Transformers are: queries, keys and values, scaled dot-product attention, multi-head and self-attention.

Queries, Keys, Values The Transformer paper redefined the attention mechanism by providing a generic definition based on queries, keys, values. The encoded representation of the input is viewed as a set of key-value pairs, of input sequence length. The previously generated output in the decoder is denoted as a query.

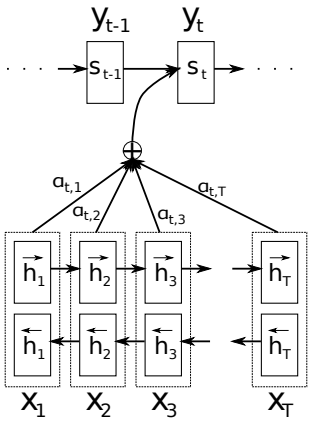

Figure 1. The graphical illustration of the attention mechanism proposed in [Bahdanau et al., 2015]

Scaled dot-product attention In [Vaswani et al., 2017], the authors propose *scaled dot-product attention*. The input consists of queries and keys of dimension d_k, and values of dimension d_v. The attention is computed as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V. \tag{4}$$

From this equation, it can be noticed that the dot product of the query with all keys represents a matching score between the two representations. This score is scaled by $\frac{1}{\sqrt{d_k}}$, because dot products can grow large in magnitude due to long sequence inputs. After applying a softmax we obtain the weights of the values.

Multi-head attention Instead of performing a single attention function, it is beneficial to linearly project the Q, K and V, h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. The outputs are concatenated and once again projected by an output layer:

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O, \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).\end{aligned}$$

Self-attention Self-attention (also called intra-attention) is a variant of the attention mechanism relating different positions of a single sequence in order to compute a revised representation of the sequence.

Question 1.1 (3 points)

Consider the encoder-decoder attention introduced in Bahdanau et al. [2015] and the self-attention used in encoder and decoder of the Transformers architecture [Vaswani et al., 2017]. Compare the two mechanisms by denoting what the queries, keys and values represent. (Words limit: 100)

Question 1.2 (4 points)

Discuss the challenge of long input sequence lengths for the Transformer model by:

- Explaining what is the underlying cause of this challenge.
- Describing a way on how to overcome this challenge.

(Words limit: 150)

1.3 Transformer-based Models

Transformers became the de-facto standard not only for NLP tasks but also for vision and multimodal tasks. The family of Transformer-based models grows every day, thus it is important to understand the fundamentals of different models and which tasks they can solve.

Question 1.3 (3 points)

Describe how would you solve the task of *Spam classification* by using Transformers, if no meta-data is available (only the email text itself). Specifically:

- Explain how the input and output representations should look like.
- Design the training and inference stage.

(Words limit: 200)

2 Variational Auto Encoders **(Total: 40 points)**

VAEs leverage the flexibility of neural networks (NN) to learn and specify a latent variable model. We will first briefly discuss Latent Variable Models and then dive into VAEs. Mathematically, they are connected to a distribution $p(\boldsymbol{x})$ over $\boldsymbol{x}$ in the following way: $p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$. This integral is typically too expensive to evaluate. However, in this assignment, you will learn a solution via VAEs.

2.1 Latent Variable Models

A latent variable model is a statistical model that contains both observed and unobserved (i.e. 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. A simple latent variable model for this data is shown in Figure 2, which we can also summarize with the following generative story:

$$\boldsymbol{z_n} \sim \mathcal{N}(0, \boldsymbol{I}_D) \tag{5}$$
$$\boldsymbol{x_n} \sim p_X(f_\theta(\boldsymbol{z}_n)) \tag{6}$$

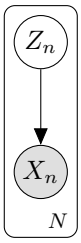

Figure 2. Graphical model of VAE. N denotes the dataset size.

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 be a Gaussian distribution we will use $f_\theta : \mathbb{R}^D \to (\mathbb{R}^M, \mathbb{R}_+^M)$ for a mean and covariance matrix, 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. Likewise, if pixels can take on k discrete values, p_X could be a product of Categorical distributions, so that $f_\theta : \mathbb{R}^D \to (p_1, ..., p_k)^M$. Where $p_1, ..., p_k$ are event probabilities of the pixel belonging to value k, where $p_i \geq 0$ and $\sum_{i=1}^k p_i = 1$. 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 $f_\theta(\cdot)$.

Food for thought

How does the VAE relate to a standard autoencoder (see e.g. Tutorial 9)?

1. Are they different in terms of their main purpose? 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 for its purpose you mentioned above?

2.2 Decoder: The Generative Part of the VAE

In the previous section, we described a general graphical model which also applies to VAEs. 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 the dataset $\mathcal{D}$ are Categorical(p) distributed.

$$p(\boldsymbol{z}_n) = \mathcal{N}(0, \boldsymbol{I}_D) \tag{7}$$
$$p(\boldsymbol{x}_n|\boldsymbol{z}_n) = \prod_{m=1}^{M} \text{Cat}(\boldsymbol{x}_n^{(m)}|f_\theta(\boldsymbol{z}_n)_m) \tag{8}$$

where $\boldsymbol{x}_n^{(m)}$ is the m-th pixel of the n-th image in $\mathcal{D}$, $f_\theta : \mathbb{R}^D \to (p_1, ..., p_k)^M$ is a neural network parameterized by θ that outputs the probabilities of the Categorical distributions for each pixel in $\boldsymbol{x}_n$. In other words, $\boldsymbol{p} = (p_1, ..., p_k)$ are event probabilities of the pixel belonging to value k, where $p_i \geq 0$ and $\sum_{i=1}^k p_i = 1$.

Question 2.1 (3 points)

Describe the steps needed to sample from such a model. (Hint: ancestral sampling)

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:

$$\begin{aligned}\log p(\mathcal{D}) &= \sum_{n=1}^{N} \log p(\boldsymbol{x}_n) \\ &= \sum_{n=1}^{N} \log \int p(\boldsymbol{x}_n|\boldsymbol{z}_n) p(\boldsymbol{z}_n) d\boldsymbol{z}_n \\ &= \sum_{n=1}^{N} \log \mathbb{E}_{p(\boldsymbol{z}_n)} \left[p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right]\end{aligned} \tag{9}$$

Evaluating $\log p(\boldsymbol{x}_n) = \log \mathbb{E}_{p(\boldsymbol{z}_n)} \left[p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right]$ involves a very expensive integral. However, Equation 9 hints at a method for approximating it, namely Monte-Carlo Integration. The log-likelihood can be approximated by drawing samples $\boldsymbol{z}_n^{(l)}$ from $p(\boldsymbol{z}_n)$:

$$\log p(\boldsymbol{x}_n) = \log \mathbb{E}_{p(\boldsymbol{z}_n)} \left[p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] \tag{10}$$

$$\approx \log \frac{1}{L} \sum_{l=1}^{L} p(\boldsymbol{x}_n|\boldsymbol{z}_n^{(l)}), \quad \boldsymbol{z}_n^{(l)} \sim p(\boldsymbol{z}_n) \tag{11}$$

If we increase the number of samples L to infinity, the approximation would be equals to the actual expectation. Hence, the estimator is unbiased and can be used to approximate $\log p(\boldsymbol{x}_n)$ with a sufficient large number of samples.

Question 2.2 (3 points)

Although Monte-Carlo Integration with samples from $p(\boldsymbol{z}_n)$ 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 this efficiency scales with the dimensionality of $\boldsymbol{z}$. (Hint: you may use Figure 3 in you explanation.)

2.3 KL Divergence

Before continuing our discussion about VAEs, we will need to learn about another concept that will help us later: the Kullback-Leibler divergence (KL divergence). It measures how different one probability distribution is from another:

$$D_{\mathrm{KL}}(q||p) = -\mathbb{E}_{q(x)} \left[\log \frac{p(X)}{q(X)} \right] = -\int q(x) \left[\log \frac{p(x)}{q(x)} \right] dx, \tag{12}$$

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

Question 2.3 (2 points)

Assume that q and p in Equation 12, are univariate gaussians: $q = \mathcal{N}(\mu_q, \sigma_q^2)$ and $p = \mathcal{N}(\mu_p, \sigma_p^2)$. Give two examples of ($\mu_q$, μ_p, σ_q^2, σ_p^2): one of which results in a very small, and one of which has a very large, KL-divergence: $D_{\mathrm{KL}}(q||p)$.

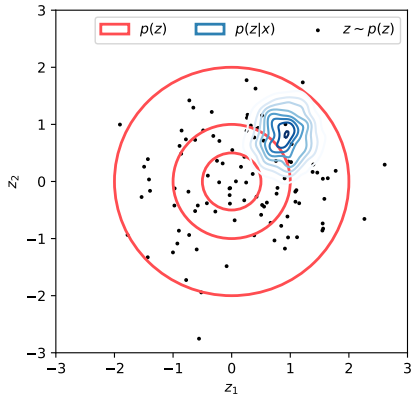

Figure 3. Plot of 2-dimensional latent space and contours of prior and posterior distributions. The red contour shows the prior $p(z)$ which is a Gaussian distribution with zero mean and standard deviation of one. The black points represent samples from the prior $p(z)$. The blue contour shows the posterior distribution $p(z|x)$ for an arbitrary x, which is a complex distribution and here, for example, peaked around $(1, 1)$.

In VAEs, we usually set the prior to be a normal distribution with a zero mean and unit variance: $p = \mathcal{N}(0, 1)$. For this case, we can actually find a closed-form solution of the KL divergence:

$$KL(q, p) = -\int q(x) \log p(x) dx + \int q(x) \log q(x) dx \tag{13}$$

$$= \frac{1}{2} \log(2\pi\sigma_p^2) + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2}(1 + \log 2\pi\sigma_q^2) \tag{14}$$

$$= \log \frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2} \tag{15}$$

$$= \frac{\sigma_q^2 + \mu_q^2 - 1 - \log \sigma_q^2}{2} \tag{16}$$

For simplicity, we skipped a few steps in the derivation. You can find the details here if you are interested (it is not essential for understanding the VAE). We will need this result for our implementation of the VAE later.

2.4 The Encoder: $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ - Efficiently evaluating the integral

In the previous section 2.2, we have developed the intuition why we need the posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$. Unfortunately, the true posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ is as difficult to compute as $p(\boldsymbol{x}_n)$ itself. To solve this problem, instead of modeling the true posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$, we can learn an approximate posterior distribution, which we refer to as the variational distribution. This variational distribution $q(\boldsymbol{z}_n|\boldsymbol{x}_n)$ is used to approximate the (very expensive) posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$.

Now we have all the tools to derive an efficient bound on the log-likelihood $\log p(\mathcal{D})$. We start from Equation 9 where the log-likelihood objective is written, but for simplicity in

notation we write the log-likelihood $\log p(\boldsymbol{x}_n)$ only for a single datapoint.

$$
\begin{aligned}
\log p(\boldsymbol{x}_n) &= \log \mathbb{E}_{p(\boldsymbol{z}_n)} \left[p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] \\
&= \log \mathbb{E}_{p(\boldsymbol{z}_n)} \left[\frac{q(\boldsymbol{z}_n|\boldsymbol{x}_n)}{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] \quad \text{(multiply by } q(\boldsymbol{z}_n|\boldsymbol{x}_n)/q(\boldsymbol{z}_n|\boldsymbol{x}_n)\text{)} \\
&= \log \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\frac{p(\boldsymbol{z}_n)}{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] \quad \text{(switch expectation distribution)} \\
&\geq \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \log \left[\frac{p(\boldsymbol{z}_n)}{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] \quad \text{(Jensen's inequality)} \\
&= \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] + \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \log \left[\frac{p(\boldsymbol{z}_n)}{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \right] \quad \text{(re-arranging)} \\
&= \underbrace{\mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] - KL\left(q(Z|\boldsymbol{x}_n) || p(Z) \right)}_{\text{Evidence Lower Bound (ELBO)}} \quad \text{(writing 2nd term as KL)}
\end{aligned}
\tag{17}
$$

This is awesome! We have derived a bound on $\log p(\boldsymbol{x}_n)$, exactly the thing we want to optimize, where all terms on the right hand side are computable. Let's put together what we have derived again in a single line:

$$
\log p(\boldsymbol{x}_n) \geq \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] - KL\left(q(Z|\boldsymbol{x}_n) || p(Z) \right).
$$

The right side of the equation is referred to as the *evidence lowerbound* (ELBO) on the log-probability of the data.

This leaves us with the question: How close is the ELBO to $\log p(\boldsymbol{x}_n)$? With an alternate derivation[1], we can find the answer. It turns out the gap between $\log p(\boldsymbol{x}_n)$ and the ELBO is exactly $KL(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$ such that:

$$
\log p(\boldsymbol{x}_n) - KL(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n)) = \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)} \left[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n) \right] - KL\left(q(Z|\boldsymbol{x}_n) || p(Z) \right) \tag{18}
$$

Now, let's optimize the ELBO. For this, we define our loss as the mean negative lower bound over samples:

$$
\mathcal{L}(\theta, \phi) = -\frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{q_\phi(z|\boldsymbol{x}_n)} [\log p_\theta(\boldsymbol{x}_n|Z)] - D_{\mathrm{KL}}(q_\phi(Z|\boldsymbol{x}_n)||p_\theta(Z)) \tag{19}
$$

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

Question 2.4 (3 points)

Explain how you can see from Equation 18 that the right hand side has to be a *lower bound* on the log-probability $\log p(\boldsymbol{x}_n)$? Why must we optimize the lower-bound, instead of optimizing the log-probability $\log p(\boldsymbol{x}_n)$ directly?

Question 2.5 (3 points)

Now, looking at the two terms on left-hand side of 18: Two things can happen when the lower bound is pushed up. Can you describe what these two things are?

[1] This derivation is not done here, but can be found in for instance Bishop sec 9.4.

2.5 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 to 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.,

$$q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{z}_n|\mu_\phi(\boldsymbol{x}_n), \text{diag}(\Sigma_\phi(\boldsymbol{x}_n))), \tag{20}$$

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 $\Sigma_\phi : \mathbb{R}^M \to \mathbb{R}^D_+$ 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 $i, j \in \{1, \ldots, K\}$

$$\text{diag}(\boldsymbol{v})_{ij} = \begin{cases} v_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}. \tag{21}$$

Question 2.6 (3 points)

The loss in Equation 19:

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

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.)

Now we have defined an objective (Equation 19) in terms of an abstract model and variational approximation, we can put everything together using our model definition (Equation 5 and 6) and definition of $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ (Equation 20), and we can write down a single objective which we can minimize.

First, we write down the reconstruction term:

$$\begin{aligned}\mathcal{L}_n^{\text{recon}} &= -\mathbb{E}_{q_\phi(z|\boldsymbol{x}_n)}[\log p_\theta(\boldsymbol{x}_n|Z)] \\ &= -\frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n^{(l)}), \quad \boldsymbol{z}_n^{(l)} \sim q(\boldsymbol{z}_n|\boldsymbol{x}_n)\end{aligned}$$

here we used Monte-Carlo integration to approximate the expectation

$$= -\frac{1}{L}\sum_{l=1}^{L}\sum_{m=1}^{M}\log \mathrm{Cat}(\boldsymbol{x}_n^{(m)}|f_\theta(\boldsymbol{z}_n^{(l)}))$$

Remember that $f_\theta(\cdot)$ denotes our decoder. Now let $\boldsymbol{p}_{nl}^{(m)} = f_\theta(\boldsymbol{z}_n^{(l)})_m$, then

$$= -\frac{1}{L}\sum_{l=1}^{L}\sum_{m=1}^{M}\sum_{k=1}^{K}\boldsymbol{x}_{nk}^{(m)}\log p_{nlk}^{(m)}.$$

where $\boldsymbol{x}_{nk}^{(m)} = 1$ if the m-th pixel has the value k, and zero otherwise. In other words, the equation above represents the common cross-entropy loss term. When setting $L = 1$ (i.e. only one sample for $\boldsymbol{z}_n$), we obtain:

$$= -\frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{K}\boldsymbol{x}_{nk}^{(m)}\log p_{nk}^{(m)}$$

where $\boldsymbol{p}_n^{(m)} = f_\theta(\boldsymbol{z}_n)_m$ and $\boldsymbol{z}_n \sim q(\boldsymbol{z}_n|\boldsymbol{x}_n)$. Thus, we can use the cross-entropy loss with respect to the original input $\boldsymbol{x}_n$ to optimize $\mathcal{L}_n^{\text{recon}}$

Next, we write down the regularization term:

$$\begin{aligned}\mathcal{L}_n^{\text{reg}} &= D_{\mathrm{KL}}(q_\phi(Z|\boldsymbol{x}_n)||p_\theta(Z)) \\ &= D_{\mathrm{KL}}(\mathcal{N}(Z|\mu_\phi(\boldsymbol{x}_n), \mathrm{diag}(\Sigma_\phi(\boldsymbol{x}_n)))\,||\,\mathcal{N}(Z|\boldsymbol{0}, \boldsymbol{I}_D))\end{aligned}$$

Using the fact that both probability distributions factorize and that the KL-divergence of two factorizable distributions is a sum of KL terms, we can rewrite this to

$$= \sum_{d=1}^{D} D_{\mathrm{KL}}(\mathcal{N}(Z^{(d)}|\mu_\phi(\boldsymbol{x}_n)_d, \Sigma_\phi(\boldsymbol{x}_n)_d\,||\,\mathcal{N}(Z^{(d)}|0,1))$$

Let $\mu_{nd} = \mu_\phi(\boldsymbol{x}_n)_d$ and $\sigma_{nd} = \Sigma_\phi(\boldsymbol{x}_n)_d$, then using the solution we found for question 1.5 we have

$$= \frac{1}{2}\sum_{d=1}^{D}\sigma_{nd}^2 + \mu_{nd}^2 - 1 - \log\sigma_{nd}^2.$$

Hence, we can find the regularization term via the simple equation above.

2.6 The Reparametrization Trick

Although we have written down (the terms of) an objective above, we still cannot simply minimize this by taking gradients with regard 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., $\nabla_\phi\mathcal{L}(\theta,\phi)$. Our posterior approximation $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ is parameterized by ϕ. If we want to train $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ to maximize the lower bound, and therefore approximate the posterior, we need to have the gradient of the lower-bound with respect to ϕ.

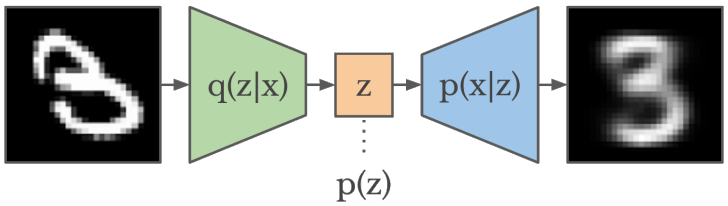

Figure 4. A VAE architecture on MNIST. The encoder distribution $q(z|x)$ maps the input image into latent space. This latent space should follow a unit Gaussian prior $p(z)$. A sample from $q(z|x)$ is used as input to the decoder $p(x|z)$ to reconstruct the image. Figure taken from this blog. Note that we are using FashionMNIST and not MNIST to train our VAE in this assignment.

Question 2.7 (3 points)

Passing the derivative through samples can be done using the *reparameterization trick*. In a few sentences, explain why the act of sampling usually prevents us from computing $\nabla_\phi \mathcal{L}$, and how the reparameterization trick solves this problem.

2.7 Putting things together: Building a VAE

Given everything we have discussed so far, 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). Thus, we can now implement a VAE in PyTorch to train on FashionMNIST images. We will model the encoder $q(z|x)$ and decoder $p(x|z)$ by a deep neural network each, and train them to maximize the data likelihood. See Figure 4 for an overview of the components we need to consider in a VAE.

In the code directory `part1`, you can find the templates to use for implementing the VAE. We provide two versions for the training loop: a template in PyTorch Lightning (`train_pl.py`), and a template in plain PyTorch (`train_torch.py`). You can choose which you prefer to implement. **You only need to implement one of the two training loop templates**. If you followed the tutorial notebooks, you might want to give PyTorch Lightning a try as it is less work, more structured and has an automatic saving and logging mechanism. You do not need to be familiar with PyTorch Lightning to the lowest level, but a high-level understanding as from the introduction in Tutorial 5 is sufficient for implementing the template.
You also need to implement additional functions in `utils.py`, and the encoder and decoder in the files `cnn_encoder_decoder.py`. We specified a recommended architecture to start with, but you are allowed to experiment with your own ideas for the models. For the sake of the assignment, it is sufficient to use the recommended architecture to achieve full points. Use the provided unit tests to ensure the correctness of your implementation. Details on the files can be found in the README of part 2.

As a loss objective and test metric, we will use the bits per dimension score (bpd). Bpd is motivated from an information theory perspective and describes how many bits we would need to encode a particular example in our modeled distribution. You can see it as how many bits we would need to store this image on our computer or send it over a network, if we have given our model. The less bits we need, the more likely the example is in our distribution. Hence, we can use bpd as loss metric to minimize. When we test for the bits per dimension on our test dataset, we can judge whether our model generalizes to new samples of the dataset and didn't in fact memorize the training dataset. In order to calculate the bits per dimension score, we can rely on the negative log-likelihood we got from the ELBO, and change the log base (as bits are binary while NLL is usually

exponential):

$$\text{bpd} = \text{nll} \cdot \log_2(e) \cdot \left(\prod_i d_i \right)^{-1}$$

where $d_1, ..., d_K$ are the dimensions of the input excluding any batch dimension. For images, this would be the height, width and channel number. We average over those dimensions in order to have a metric that is comparable across different image resolutions. The nll represents the negative log-likelihood loss $\mathcal{L}$ from Equation 19 for a single data point. You should implement this function in `utils.py`.

Question 2.8 (12 points)

Build a Variational Autoencoder in the provided templates, and train it on the FashionMNIST dataset. Both the encoder and decoder should be implemented as a CNN. For the architecture, you can use the same as used in Tutorial 9 about Autoencoders. Note that you have to adjust the output shape of the decoder to $1 \times 28 \times 28$ for FashionMNIST. You can do this by adjusting the output padding of the first transposed convolution in the decoder. Use a latent space size of `z_dim=20`. Read the provided README to become familiar with the code template.

In your submission, provide a short description (no more than 8 lines) of the used architectures for the encoder and decoder, any hyperparameters and your training steps. Additionally, plot the estimated bit per dimension score of the lower bound on the training and validation set as training progresses, and the final test score. You are allowed to take screenshots of a TensorBoard plot if the axes values are clear.

Note: using the default hyperparameters is sufficient to obtain full points. As a reference, the training loss should start at around 4 bpd, reach below 2.0 after 2 epochs, and end between 1.20-1.25 after 80 epochs.

Question 2.9 (4 points)

Plot 64 samples (8×8 grid) from your model at three points throughout training (before training, after training 10 epochs, and after training 80 epochs). You should observe an improvement in the quality of samples. Describe shortly the quality and/or issues of the generated images.

Question 2.10 (4 points)

Train a VAE with a 2-dimensional latent space (`z_dim=2` in the code). Use this VAE to plot the data manifold as is done in Figure 4b of [Kingma and Welling, 2014]. This is achieved by taking 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. Implement it in the function `visualize_manifold` in `utils.py`, and use a grid size of 20. Are you recognizing any patterns of the positions of the different items of clothing?

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL http://arxiv.org/abs/1409.0473. 2, 3

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

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. International Conference on Learning Representations (ICLR), 2014. 1, 12

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. URL http://dl.acm.org/citation.cfm?id=3045118.3045281. 1

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017. 3