3A-MDS-E2-DL: Introduction to Deep Learning
MVA Master

Assignment 1

Due: January 07, 2025

Information about the exercise:

- Please complete the first assignment **individually**.
- Please send your answers on a **PDF** file together with your source code in order to produce the answers whenever it is needed. You can also submit a single **Jupyter** notebook which summarise all your answers.
- For your answers please provide short and clear descriptions with figures where ever this is possible.
- Please do not forget to write your name on the reports and submit your assignment on `deeplearning.mva@gmail.com` by the day of the deadline, using as title "3A-MDS-E2-DL_Assignment".
- The total number of points is 100.
- Good luck!

A. Computer Vision Part

1 Variational autoencoders [35 points]

VAEs leverage the flexibility of neural networks (NN) to learn and specify a latent variable model. Mathematically, they are connected to a distribution ($p(x)$) over x in the following way: $p(x) = \int p(x|z)p(z)dz$. This integral is typically too expensive to evaluate, which VAEs have resolved in a way that you learn in this assignment.

1.1 Latent Variable Models

A latent variable model is a statistical model that contains both observed and unobserved (i.e. latent) variables. Assume a dataset $D = \{x_n\}_{n=1}^N$, where $x_n \in \{0,1\}^M$. For example, x_n can be the pixel values of a binary image.

$$z_n \sim N(0, I_D) \tag{1}$$

$$x_n \sim p_X(f_\theta(z_n)) \tag{2}$$

where f_θ is some function – parameterized by θ – that maps z_n to the parameters of a distribution over x_n. For example, if p_X would be a Gaussian distribution we will use $f_\theta{:}R^D \to (R^M, R_+^M)$ for a mean and covariance matrix, or if p_X is a product of Bernoulli distributions, we have $f_\theta{:}R^D \to [0,1]^M$. Here, D denotes the dimensionality of the latent space. Note that our dataset D does not contain z_n, hence z_n is a latent (or unobserved) variable in our statistical model. In the case of a VAE, a (deep) NN is used for $f_\theta(\cdot)$.

1.2 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 x_n in D are Bernoulli (p) distributed.

$$p(z_n) = N(0, I_D) \tag{3}$$

$$p(x_n|z_n) = \prod_{m=1}^{M} Bern(x_n^{(m)}|f_\theta(z_n)_m) \tag{4}$$

where $x_n^{(m)}$ is the m-th pixel of the n-th image in D, and f_θ:$R^D \rightarrow [0,1]^M$ is a neural network parameterized by θ that outputs the means of the Bernoulli distributions for each pixel in x_n.

Question 1 [3 point] 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 D under this model:

$$\log(p(D)) = \sum_{n=1}^{N} \log p(x_n) = \sum_{n=1}^{N} \log \int p(x_n|z_n)p(z_n)dz_n = \sum_{n=1}^{N} \log E_{p(z_n)}[p(x_n|z_n)] \tag{5}$$

Evaluating $\log p(x_n) = \log E_{p(z_n)}[p(x_n|z_n)]$ involves a very expensive integral. However, Eq. 5 hints at a method for approximating it, namely Monte-Carlo Integration. The log-likelihood can be approximated by drawing samples $z_n^{(l)}$ from $p(z_n)$:

$$\log(p(x_n)) = \log E_{p(z_n)}[p(x_n|z_n)] \tag{6}$$

$$\approx \log \frac{1}{L} \sum_{l=1}^{L} p(x_n|z_n^{(l)}), \quad z_n^{(l)} \sim p(z_n) \tag{7}$$

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 $\log p(x_n)$ with a sufficient large number of samples.

Question 2 [3 points] Although Monte-Carlo Integration with samples from $p(z_n)$ can be used to approximate $\log p(x_n)$, 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 z.

1.3 KL Divergence

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

$$D_{KL}(q||p) = -E_{q(x)}[\log \frac{p(X)}{q(X)}] = -\int q(x)[\log \frac{p(x)}{q(x)}]dx, \tag{8}$$

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

Question 3 [3 points] Assume that q and p in Eq. 8, are univariate gaussians: $q = N(\mu_q, \sigma_q^2)$ and $p = N(\mu_p, \sigma_p^2)$. Give two examples of $(\mu_q, \mu_p, \sigma_q^2, \sigma_p^2)$: one of which results in a very small, and one of which has a very large, KL-divergence: $D_{KL}(q||p)$.

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

$$KL(q,p) = -\int q(x) \log p(x)dx + \int q(x) \log p(x)dx \tag{9}$$

$$= \frac{1}{2}\log(2\pi\sigma_p^2) + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2}(1 + \log 2\pi\sigma_q^2) \tag{10}$$

$$= \log \frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2} \tag{11}$$

$$= \frac{\sigma_q^2 + \mu_q^2 - 1 - \log \sigma_q^2}{2} \tag{12}$$

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.

1.4 The Encoder: $q_\phi(z_n|x_n)$ - Efficiently evaluating the integral

In the previous section 0.2, we have developed the intuition that we only want to sample z_n for which $p(z_n|x_n)$ is not close to zero - in order to compute the Monte-Carlo approximation. Unfortunately, the true posterior $p(z_n|x_n)$ is as difficult to compute as $p(x_n)$ itself. To solve this problem, instead of modeling the true posterior $p(z_n|x_n)$, we can learn an approximate posterior distribution, which we refer to as the variational distribution. This variational distribution $q(z_n|x_n)$ is used to approximate the (very expensive) posterior $p(z_n|x_n)$ and to more efficiently integrate $\int p(x_n|z_n)p(z_n)dz_n$.

Now, we have all the tools to derive an efficient bound on the log-likelihood $\log p(D)$. We start from Eq. 5 where the log-likelihood objective is written, but for simplicity in notation we write the log-likelihood $\log p(x_n)$ only for a single datapoint.

$$\log p(x_n) = \log E_{p(z_n)}[p(x_n|z_n)] = \log E_{p(z_n)}[\frac{q(z_n|x_n)}{q(z_n|x_n)}p(x_n|z_n)] = \log E_{q(z_n|x_n)}[\frac{p(z_n)}{q(z_n|x_n)}p(x_n|z_n)] \tag{13}$$

$$\geq E_{q(z_n|x_n)}\log[\frac{p(z_n)}{q(z_n|x_n)}] = E_{q(z_n|x_n)}[\log p(x_n|z_n)] + E_{q(z_n|x_n)}\log[\frac{p(z_n)}{q(z_n|x_n)}] = E_q(z_n|x_n)[\log p(x_n|z_n)] - KL(q(Z|x_n)||p(Z)) \tag{14}$$

We have derived a bound on $\log p(x_n)$, 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:

$$\log p(x_n) \geq E_{q(z_n|x_n)}[\log p(x_n|z_n) - KL(q(Z|x_n)||p(Z))] \tag{15}$$

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 $\log p(x_n)$? With an alternate derivation[1] we can find the answer. It turns out the gap between $\log p(x_n)$ and the ELBO is exactly $KL(q(Z|x_n)||p(Z|x_n))$ such that:

$$\log p(x_n) - KL(q(Z|x_n)||p(Z|x_n)) = E_{q(z_n|x_n)}[\log p(x_n|z_n)] - KL(q(Z|x_n)||p(Z)) \tag{16}$$

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

$$\mathbf{L}(\theta, \phi) = -\frac{1}{N}\sum_{n=1}^{N} E_{q_\phi(z|x_n)}[\log p_\theta(x_n|Z)] - D_{KL}(q_\phi(Z|x_n)||p_\theta(Z)) \tag{17}$$

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

Question 4 [3 points] Explain how you can see from Eq. 16 that the right hand side had to be a lower bound of the log-probability $\log p(x_n)$. Why must we optimize the lower-bound, instead of optimizing the log-probability $\log p(x_n)$ directly?

Question 5 [3 points] Now, looking at the two terms on left-hand side of Eq. 16: Two things can happen when the lower-bound is pushed up. Can you describe what these two thing are?

1.5 Specifying the Encoder $q_\phi(z_n|x_n)$

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

$$q_\phi(z_n|x_n) = N(z_n|\mu_\phi(x_n), diag(\Sigma_\phi(x_n))), \tag{18}$$

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

$$diag(v)_{ij} = \begin{cases} v_i, & \text{if } i = j. \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

Question 6 [4 points] The loss in Eq. 17 can be rewritten in terms of per-sample losses:

$$\mathbf{L} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{L_n^{recon}} + \mathbf{L_n^{reg}}) \tag{20}$$

where $\mathbf{L_n^{recon}} = -E_{q_\phi(z|x_n)}[\log p_\theta(x_n|Z)]$ and $\mathbf{L_n^{reg}} = D_{KL}(Q_(Z|x_n)||p_\theta(Z))$ 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 $E_{\phi(z|x_n)}[p_\theta(_n|Z)]$ – as is common practice in VAEs.)

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

Question 7 [4 points] Now we have defined an objective (Eq. 17) in terms of an abstract model and variational approximation, we can put everything together using our model definition (Equations 1, 2) and definition of $q_\theta(z_n|x_n)$ (Eq. 18), and we can write down a single objective which we can minimize. Write down expressions (including steps) for $\mathbf{L_n^{recon}}$ and $\mathbf{L_n^{reg}}$ such that we can minimize $\mathbf{L} = \sum_{n=1}^{N}(\mathbf{L_n^{recon}} + \mathbf{L_n^{reg}})$ as our final objective. Make any approximation explicit. (Hint: look at Eq. 9 for $\mathbf{L_n^{reg}}$).

1.6 The Reparametrization Trick

Although we have written down (the terms of) an objective in Question 7, 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 $q_\phi(z_n|x_n)$ to approximate the $E_{q_\phi(z|x_n)}[\log p_\theta(x_n|Z)]$ term. Yet, we need to pass the derivative through these samples if we want to compute the gradient of the encoder parameters, i.e., $\nabla_\phi \mathbf{L}(\theta, \phi)$. Our posterior approximation $q_\phi(z_n|x_n)$ is parameterized by ϕ. If we want to train $q_\phi(z_n|x_n)$ to maximize the lower bound, and therefore approximate the posterior, we need to have the gradient of ϕ with respect to the lower-bound.

Question 8 [4 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 $\nabla_\phi \mathbf{L}$, and how the reparameterization trick solves this problem. (Hint: you can take a look at Figure 4 from the tutorial by Carl Doersch)

1.7 Putting things together: Building a VAE

Given everything we have discussed so far, we now have an objective (the evidence lower boundor 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 MNIST 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.

Question 9 [4 points] Build a Variational Autoencoder, and train it on the binarized MNIST dataset. Both the encoder and decoder should be implemented as an MLP. Following standard practice – and for simplicity – you may assume that the number of samples used to approximate the expectation in $\mathbf{L_n^{recon}}$ is 1. Use a latent space size of $z_{dim} = 20$. In your report, provide a short description (no more than 10 lines) of the used architectures for the encoder and decoder, any hyperparameters and your training steps.

Question 10 [4 points] Plot 64 samples (8×8grid) 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.

2 Diffusion [35 points]

In this part, you will focus on the working of a more recent generative process called Diffusion in order to understand its inner process. A Hierarchical Variational Autoencoder (HVAE) [3, 5] is a generalization of a VAE extended to multiple hierarchies over latent variables. Under this formulation, latent variables themselves are generated from other higher-level, more abstract latents.

In the general HVAE with T hierarchical levels, each latent is allowed to condition on all previous latents. However, in this work, we focus on a special case called a Markovian HVAE (MHVAE). In a MHVAE, the generative process is a Markov chain; that is, each transition down the hierarchy is Markovian, where decoding each latent $\boldsymbol{z}_t$ only conditions on previous latent $\boldsymbol{z}_{t+1}$. Intuitively, and visually, this can be seen as simply stacking VAEs on top of each other; another appropriate term describing this model is a Recursive VAE. Mathematically, we represent the joint distribution and the posterior of a Markovian HVAE as:

$$p(\boldsymbol{x}, \boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) \tag{21}$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_1|\boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) \tag{22}$$

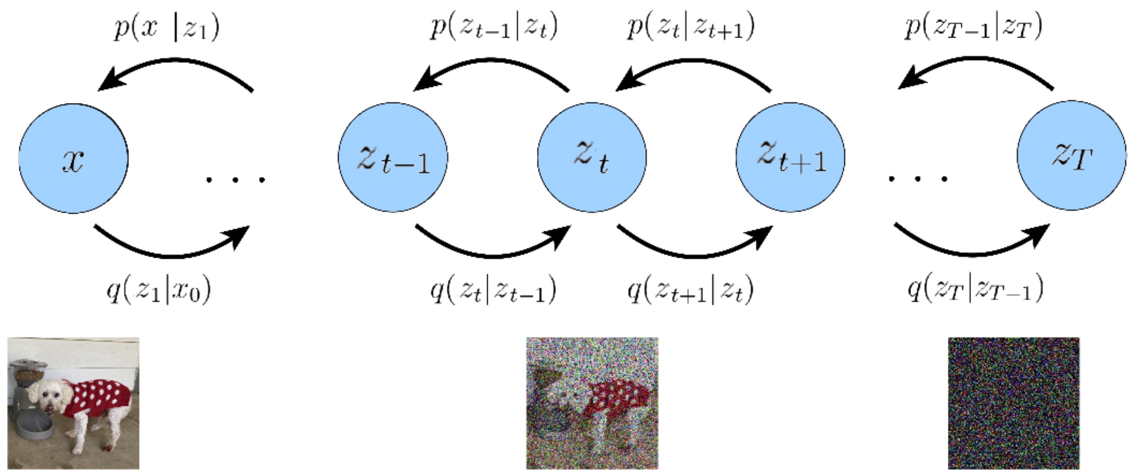


Figure 1: A visual representation of a Variational Diffusion Model; $\boldsymbol{x}$ represents true data observations such as natural images, $\boldsymbol{z}_T$ represents pure Gaussian noise, and $\boldsymbol{z}_t$ is an intermediate noisy version of $\boldsymbol{x}$. Each $q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$ is modeled as a Gaussian distribution that uses the output of the previous state as its mean.

Question 11 [4 points] Using $\log p(\boldsymbol{x}) = \log(\int p(\boldsymbol{x}, \boldsymbol{z}_{1:T})d\boldsymbol{z}_{1:T})$, extend the ELBO formula for Markovian Hierarchical Variational Autoencoder to obtain : $\log p(\boldsymbol{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right]$

The easiest way to think of a Variational Diffusion Model (VDM) [4, 1, 2] is simply as a Markovian Hierarchical Variational Autoencoder with three key restrictions:

- The latent dimension is exactly equal to the data dimension.
- The structure of the latent encoder at each timestep is not learned; it is pre-defined as a linear Gaussian model. In other words, it is a Gaussian distribution centered around the output of the previous timestep.
- The Gaussian parameters of the latent encoders vary over time in such a way that the distribution of the latent at final timestep T is a standard Gaussian.

Furthermore, we explicitly maintain the Markov property between hierarchical transitions from a standard Markovian Hierarchical Variational Autoencoder. Figure 1 gives a representation of the VDM. **For simplicity of notation, we note $\boldsymbol{x} = z_0$ for the following questions.**

Question 12 [4 points] Based on these 3 restrictions, describe in a few lines the process (architecture, transformations, inputs and outputs).

These rules imply mathematically that :

$$q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \mathcal{N}(\boldsymbol{z}_t; \sqrt{\alpha_t}\boldsymbol{z}_{t-1}, (1-\alpha_t)\mathbf{I}) \tag{23}$$

with α_t a potentially learnable coefficient that may vary with the chosen depth of the model.
As well as :

$$p(\boldsymbol{z}_{0:T}) = p(\boldsymbol{z}_T)\prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) \qquad \text{where,} \qquad p(\boldsymbol{z}_T) = \mathcal{N}(\boldsymbol{z}_T; \mathbf{0}, \mathbf{I})$$

Question 13 [3 points] We can update the ELBO formula as

$$\begin{aligned}\log p(\boldsymbol{z}) = &\underbrace{\mathbb{E}_{q(\boldsymbol{z}_1|\boldsymbol{z}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}_0|\boldsymbol{z}_1)\right]}_{\text{reconstruction term}} - \underbrace{\mathbb{E}_{q(\boldsymbol{z}_{T-1}|\boldsymbol{z}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{z}_T|\boldsymbol{z}_{T-1})||p(\boldsymbol{z}_T))\right]}_{\text{prior matching term}} \\ &- \sum_{t=1}^{T-1}\underbrace{\mathbb{E}_{q(\boldsymbol{z}_{t-1},\boldsymbol{z}_{t+1}|\boldsymbol{z}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})||p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t+1}))\right]}_{\text{consistency term}}\end{aligned} \tag{24}$$

Comment on the several terms of the formula in eq. 24. What is the role of each of these terms in the ELBO formula?

Question 14 [4 points] Using the reparametrization trick and eq. 23, we can write that:

$$z_t = \sqrt{\alpha_t} z_{t-1} + \sqrt{1-\alpha_t}\epsilon \quad \text{with } \epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I}) \tag{25}$$

Show that z_t follow a distribution that can be written as eq. 26.

$$z_t \sim \mathcal{N}(z_t; \sqrt{\bar{\alpha}_t} z_0, (1-\bar{\alpha}_t)\,\mathbf{I}) \tag{26}$$

Hint: Use a recursive reparametrization trick and remember that if $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ then $(\alpha\epsilon_1 + \beta\epsilon_2) \sim \mathcal{N}(\mathbf{0}, (\alpha^2 + \beta^2)\mathbf{I})$.

Rewriting the loss. Using Equation 26, we can show that:

$$q(z_{t-1}|z_t, z_0) \propto \mathcal{N}(z_{t-1}; \underbrace{\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})z_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)z_0}{1-\bar{\alpha}_t}}_{\mu_q(z_t, z_0)}, \underbrace{\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}}_{\sigma_q^2(t)}\mathbf{I}) \tag{27}$$

We can compute the KL-divergence and obtain :

$$\arg\min_{\theta} D_{\text{KL}} q(z_{t-1}|z_t, z_0) p_{\theta}(z_{t-1}|z_t) = \arg\min_{\theta} \frac{1}{2\sigma_q^2(t)} \left[\|\mu_{\theta} - \mu_q\|_2^2 \right] \tag{28}$$

where we have written μ_q as shorthand for $\mu_q(z_t, z_0)$, and μ_θ as shorthand for $\mu_\theta(z_t, t)$ for brevity. In other words, we want to optimize a $\mu_\theta(z_t, t)$ that matches $\mu_q(z_t, z_0)$, which from our derived Equation 27, takes the form:

$$\mu_q(z_t, z_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})z_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)z_0}{1-\bar{\alpha}_t} \tag{29}$$

As $\mu_\theta(z_t, t)$ also conditions on z_t, we can match $\mu_q(z_t, z_0)$ closely by setting it to the following form:

$$\mu_\theta(z_t, t) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})z_t + \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{z}_\theta(z_t, t)}{1-\bar{\alpha}_t} \tag{30}$$

where $\hat{z}_\theta(z_t, t)$ is parameterized by a neural network.

Question 15 [4 points] Plug equations 29 and 30 in equation 28 and find an interpretation to the final optimization term.

2.1 Generate Image by learning the denoise

Question 16 [4 points] Recall that $q(z_t|z_0)$ is a Gaussian of form $\mathcal{N}(z_t; \sqrt{\bar{\alpha}_t} z_0, (1-\bar{\alpha}_t)\,\mathbf{I})$. Then, following the definition of the signal-to-noise ratio (SNR) as SNR $= \frac{\mu^2}{\sigma^2}$, deduce the SNR at timestep t.

Question 17 [4 points] Introduce the SNR into the equation that you got at Question 15 (recall that $\sigma_q^2(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$). What does it mean for our diffusion model? Link it to the definition of the SNR.

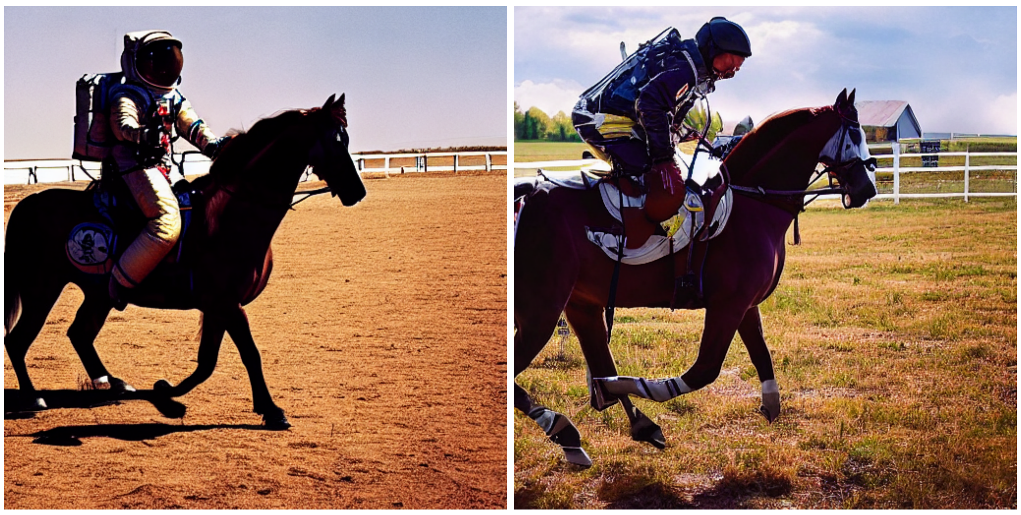
2.2 Diffusion in practice

Now that we have introduced how to train diffusion models, let's see how to use them effectively in practice. Hint: you can use the Diffusers library and modify this simple tutorial for this section: `https://huggingface.co/docs/diffusers/using-diffusers/write_own_pipeline`.

Question 18 [4 points] The reverse diffusion process (ie. generating an image from noise) can be applied with multiple ways of sampling the denoised image at each timestep t. For exemple it is possible to skip some timesteps to fasten the generation process, or to change the amount of noise added to an image to a different schedule to reduce the variations at the end of the diffusion process. The goal is to look at the practical details of diffusion by changing the scheduler when generating an image.

Find a way to generate images with the same seed but with different number of steps and different kind of noise schedulers. Plot a grid of generated images with different number of steps as rows and different schedulers as columns (use at least three schedulers including Euler, Euler Ancestral and DPM). What can you say about the differences between schedulers? What is the impact of the number of steps?

Hint: you can look at the tutorial `https://huggingface.co/docs/diffusers/using-diffusers/schedulers`.



(a) Original Image. No negative prompt **(b)** Modified image. Negative prompt: 'desert'

Figure 2: Two pictures generated with the same seed, noise and positive prompt: 'a photograph of an astronaut riding a horse' but with a different negative prompt.

Guided generation The diffusion model can be guided during training and inference by adding a loss that conditions the generated image z_t to be of a certain class or to match a specific text prompt.

Question 19 [4 points] Use a pre-trained diffusion model that can be guided by a textual prompt and generate an image with a given prompt (eg. 'a photograph of an astronaut riding a horse'). Guidance can also be used in a negative manner to remove something from generated images (eg. hands with 6 fingers, bad weather, undesired objects...).

Using the same initial noise and prompt as before, find a way to remove an object/a color/a concept that is present in your generated image but not in your prompt (e.g. if you generate an image without specifying the background and you get a desert but you want something else, remove the desert by using a negative prompt 'desert' like in Fig. 2).

Plot the two images side by side, along with their prompt/negative prompt, and write a small paragraph analyzing them. What can you say about the position of the object?

Hint: you can use the Diffusers library and modify the line

```
noise_pred = noise_pred_uncond + guidance_scale * (noise_pred_text - noise_pred_uncond)
```

in the section 'Deconstruct the Stable Diffusion pipeline' from the previous url.

B. Natural Language Processing Part

3 Multi-lingual Translation [30 points]

Multi-lingual translation is one of the most exciting directions in natural language processing, combining a task that is challenging from a semantic point of view as well as having a direct impact on millions of users; with recent developments in neural networks and multi-task learning. Instead of keeping many bilingual models to translate across different tasks, a multi-lingual models only has one (large) model that has seen many language directions during training. The goal of this exercise is to train your own multi-lingual translation model, based on the notebook shared together with the assignment.

Tips:

- Create a multilingual dictionary by concatenating the tokenized data in all languages. Or simply re-use the dictionary of the pre-trained model (`multi_dict`).
- Use the same dictionary for the source and target sides, and share the embeddings between your encoder and decoder (do: `decoder.embedding = encoder.embedding`).
- Use `nmt_dataset.MultiBatchIterator(iterator_list)` to concatenate a list of training iterators (one for each language pair) into a single iterator, which is compatible with `train_model`.
- `train_model` can take a list of several validation iterators, which will let you validate your model on several language pairs.
- Improve your model's performance on **de-fr** and **fr-de** by including training data for these languages pairs (`data/train.de` and `data/train.de-fr.fr`).

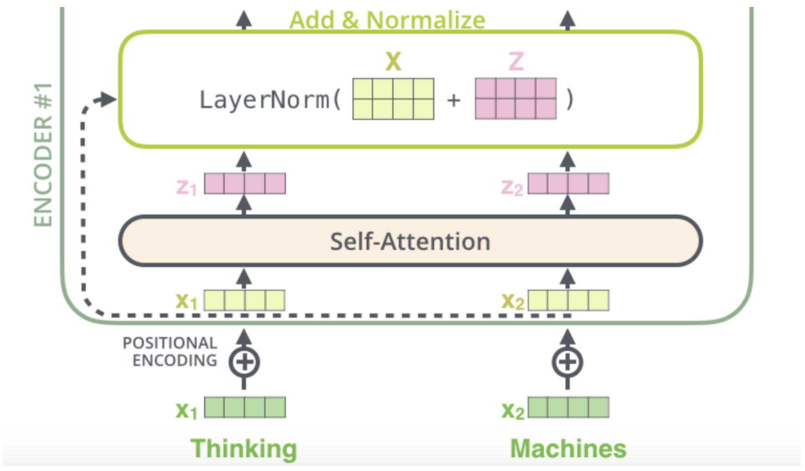


Figure 3: Original sequence of layers.The image is coming from this source

3.1 Pre-normalization

The original Transformer architecture [6] proposed the sequence of layers shown in Fig. 3. Note in particular that the normalization occurs after the self-attention operation, with an intermediate addition of the input embedding (the residual connection). In equation, and denoting by $\mathcal{A}(x)$ the self-attention layer:

$$x_{\ell+1} = LayerNorm(x_\ell + \mathcal{A}(x)) \tag{31}$$

where for simplicity we suppose an architecture without a feed-forward layer. An alternative which is often used is to do the layer normalization before the attention layer:

$$x_{\ell+1} = x_\ell + \mathcal{A}(LayerNorm(x_\ell)) \tag{32}$$

Question 20. [13 points] Derive analytically the impact of that different sequence of layers, when doing back-propagation during training. (Hint: If you denote by e the error produced in one pass, and x_L the top-most layer, then by applying chain rule you obtain:)

$$\frac{\partial e}{\partial x_\ell} = \frac{\partial e}{\partial x_L} \frac{\partial x_L}{\partial x_\ell}$$

Derive then $\frac{\partial x_{\ell+1}}{\partial x_\ell}$ (no need of actually deriving $\frac{\partial LayerNorm(x)}{\partial x}$).

Question 21. [8.5 points] Explain in a few paragraphs what do you think are the consequences of training (very deep models) of those derivations when using Eq 32 instead of Eq. 31?

Question 22. [8.5 points] Show empirically the impact of this. This is, design a measure and plot that measure against training steps for a standard task (eg: language modelling), comparing those plots between a deep and shallow model. Finding what exactly to measure is an important part of the problem.

References

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[2] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

[3] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

[4] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[5] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. *Advances in neural information processing systems*, 29, 2016.

[6] 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.