Deep Learning - Assignment

Oliver Jack
MVA

January 4, 2025

1 Variational Autoencoders

Question 1

The first step consists of sampling the latent variable $z_n \sim \mathcal{N}(0, I_D)$. Next, this sampled variable is then fed into the parametrized NN f_θ, to compute the probabilities of each pixel:

$$\mu = f_\theta(z_n) \tag{1}$$

Here, μ is a vector of length M, where each element μ_m represents the mean of the Bernoulli distribution for the m-th pixel. Finally, we can sample each pixel $x_n^{(m)} \sim Bern(\mu_m)$, which can then be combined to form the image $x_n = \left[x_n^{(1)}, \ldots, x_n^{(M)}\right]$.

Question 2

Sampling $z_n \sim p(z_n)$ directly from the prior, does not take into account the most important regions of the specific data point x_n. Hence, most samples from $p(z_n)$ will contribute very little to $p(x_n|z_n)$, leading to high variance in the estimator. This makes it hard for the estimator to converge efficiently during optimization. Moreover, due to the curse of dimensionality, the efficiency of Monte Carlo Integration scales poorly with the dimensionality of z. For high-dimensional z, a large number of samples L is required cover the space of z in a significant way, as the prior $p(z_n)$ becomes increasingly sparse in high dimensions.

Question 3

As shown in equation (12) of the assignment, the KL divergence has a closed-form solution in the case where consider $p = \mathcal{N}(0, 1)$:

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

By taking $\sigma_q^2 = 1$, this expression becomes $\mu_q^2/2$, which allows us to find a KL-divergence of any given (positive) size (take e.g. $\mu_q = 0.001$ and $\mu_q = 1000$).

Question 4

As seen in class, we know that for two distributions p and q, we have:

$$KL(p(x)||q(x)) \geq 0, \quad \forall x \tag{3}$$

Thus, by equation (16), we get that:

$$\log(p(x_n)) \geq \mathbb{E}_{q(z_n|x_n)}[\log(p(x_n|z_n)] - KL(q_\phi(Z|x_n)||(Z)) \tag{4}$$

As far as optimization is concerned, computing $\log(p(x_n))$ requires the computation of a difficult integral:

$$\log(p(x_n)) = \log \int p(x_n|z_n)p(z_n)dz_n \tag{5}$$

On the other hand, the computation of the ELBO relies on the approximated variational distribution $q_\phi(z_n|x_n)$, which is far more efficient in practice.

Question 5

When the lower bound is pushed up, one of the following things can happen. On the one hand, $KL(q_\phi(Z|x_n)||p(Z|x_n))$ shrinks, which essentially means that the variational distribution $q_\phi(z_n|x_n)$ gets closer to the true posterior $p(z_n|x_n)$. In other words, this tells us that our approximation of the posterior is improving. On the other hand, the ELBO gets closer to the real value of $\log(p(x_n))$, which implies that our model is learning a better fit for the data.

Question 6

$\mathbf{L_n^{recon}}$ is called the reconstruction loss since it claims to evaluate how well the decoder $p(x_n|z_n)$ can reconstruct x_n given a sampled latent vector z_n drawn from the encoder $q_\phi(z_n|x_n)$, penalizing the model if the predicted x_n is far from the true input. $\mathbf{L_n^{reg}}$ is called the regularization loss since it aims to keep the approximated variational distribution $q_\phi(z_n|x_n)$ close to the prior distribution $p_\theta(z_n)$, penalizing any type of overfitting on the training data.

Question 7

Let us start off by considering the reconstruction loss term:

$$\begin{aligned} \mathbf{L_n^{recon}} &= -\mathbb{E}_{q_\phi(z|x_n)}\left[\log(p_\theta(x_n|Z))\right] \\ &= -\int \log(p_\theta(x_n|z_n))q_\phi(z_n|x_n)dz_n \end{aligned} \tag{6}$$

Since this integral is difficult to compute, one can consider the following Monte-Carlo approximation:

$$\mathbf{L_n^{recon}} \approx -\frac{1}{L}\sum_{l=1}^{L} \log(p_\theta(x_n|z_n^{(l)})), \tag{7}$$

where $z_n^{(l)}$ are sampled from $q_\phi(z_n|x_n) = \mathcal{N}(z_n|\mu_\phi(x_n), diag(\Sigma_\phi(x_n)))$.

Next, let us consider the regularization loss term:

$$\begin{aligned}\mathbf{L_n^{reg}} &= D_{KL}(q_\phi(Z|x_n)||p_\theta(Z)) \\ &= -\int q_\phi(z_n|x_n)\log(p_\theta(z_n))dz_n + \int q_\phi(z_n|x_n)\log(q_\phi(z_n|x_n))dz_n.\end{aligned} \tag{8}$$

The first term of 8 is simply the expectation of the log-prior with respect to $q_\phi(z_n|x_n)$. Since we assume that $p_\theta(z_n) = \mathcal{N}(0, \mathbf{I})$, we have that:

$$\log(p_\theta(z_n)) = -\frac{1}{2}\left(z_n^T z_n + D\log(2\pi)\right), \tag{9}$$

where D is the dimension of z_n. Furthermore:

$$\begin{aligned}\mathbb{E}_{q_\theta(z_n|x_n)}[z_n^T z_n] &= \sum_{i=1}^{D}\mathbb{E}[z_{n,i}^2] \\ &= ||\mu_\phi||_2^2 + \mathrm{Tr}(\mathrm{diag}(\Sigma_\phi(x_n)))\end{aligned} \tag{10}$$

Thus:

$$\begin{aligned}\mathbb{E}_{q_\phi(z_n|x_n)}[\log(p_\theta(z_n))] &= -\frac{1}{2}\mathbb{E}_{q_\theta(z_n|x_n)}[z_n^T z_n + D\log(2\pi)] \\ &= -\frac{1}{2}\left(\mathbb{E}_{q_\theta(z_n|x_n)}[z_n^T z_n] + D\log(2\pi)\right) \\ &= -\frac{1}{2}\left(||\mu_\phi||_2^2 + \mathrm{Tr}(\mathrm{diag}(\Sigma_\phi(x_n))) + D\log(2\pi)\right)\end{aligned} \tag{11}$$

The second term of 8 is the negative entropy of $q_\phi(z_n|x_n)$. For a multivariate Gaussian distribution $\mathcal{N}(\mu_\phi(x_n), \mathrm{diag}(\Sigma_\phi(x_n)))$, the entropy is given by:

$$\begin{aligned}H(q_\phi) &= -\int q_\phi(z_n|x_n)\log(q_\phi(z_n|x_n))dz_n \\ &= \frac{1}{2}\log(\det(\mathrm{diag}(\Sigma_\phi))) + \frac{D}{2} + \frac{D}{2}\log(2\pi)\end{aligned} \tag{12}$$

Combining both of these terms gives us:

$$\mathbf{L_n^{reg}} = -\frac{1}{2}\left(||\mu_\phi||_2^2 + \mathrm{Tr}(\mathrm{diag}(\Sigma_\phi(x_n))) + 2D\log(2\pi) + \log(\det(\mathrm{diag}(\Sigma_\phi))) + D\right) \tag{13}$$

Question 8

The main problem that we face is that sampling is not differentiable due to $z_n \sim q_\phi(z_n|x_n)$ being non-differentiable. This is however crucial when performing backpropagation. Therefore, we can use the reparameterization trick which introduces differentiability by rewriting z_n:

$$z_n = \mu_\phi(x_n) + \sigma_\phi(x_n) \odot \epsilon, \tag{14}$$

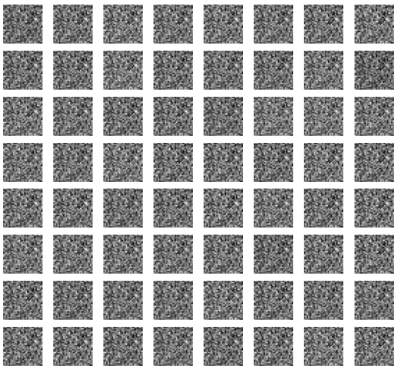
where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. Since the expression is differentiable with respect to $\mu_\phi(x_n)$ and $\sigma_\phi(x_n)$, we are able to perform gradient-based backpropagation and optimization.

Question 9

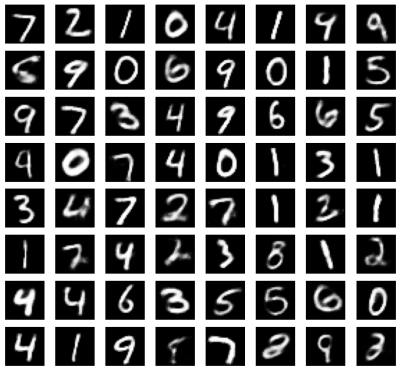
To implement a variational autoencoder, I used the Pytorch library, more specifically the `nn` (Neural Network) module. Since the images are grids of size 28×28, I first preprocessed the data by transforming it to tensors, before "binarizing" it (i.e. setting the values to 1 if they were larger than 0.5, otherwise setting them to 0). The reason for this is that we are working with a Bernoulli likelihood in our VAE. After downloading the MNIST training and test datasets, I implemented the methods `__init__` and `forward` for both the `Encoder` and `Decoder` classes. The input dimension for the encoder and output dimension for the decoder were 784 (28×28). I used 3 fully connected hidden layers of dimension 256 for both the encoder and decoder, while the latent space is of dimension 20. ReLu was used as activation function for each of the different hidden layers, while I used sigmoid for the final output layer of the decoder, ensuring the output lies in the interval [0,1]. Finally, the model was trained on a batch size of 64 for a total of 100 epochs, using Adam as an optimizer.

Question 10

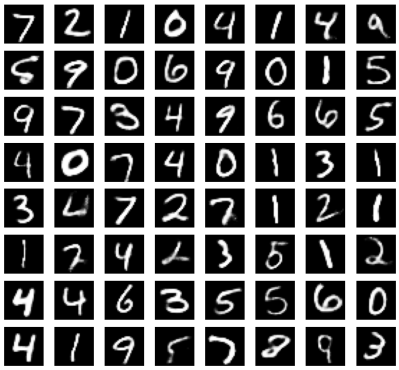
As seen in the figures, one can notice that the quality of the reconstructed samples improves with the number of epochs our model was trained for. While the untrained model (0 epochs) is unable to reconstruct any significant structure or pattern at all, the model performs relatively well after only 10 epochs, despite the images being somewhat blurry and at times ambiguous. Finally, the model which was trained for 80 epochs performs slightly better, enabling a close reconstruction of the initial input image.



After 0 epochs



After 10 epochs



After 80 epochs

2 Diffusion

Question 11

$$\begin{aligned}
\log(p(x)) &= \log\left(\int p(x, z_{1:T}) dz_{1:T}\right) \\
&= \log\left(\mathbb{E}_{p(z_{1:T})}[p(x|z_{1:T})]\right) \\
&= \log\left(\mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\frac{p(z_{1:T})}{q_\phi(z_{1:T}|x)} p(x|z_{1:T})\right]\right) \\
&\geq \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log\left(\frac{p(z_{1:T})}{q_\phi(z_{1:T}|x)} p(x|z_{1:T})\right)\right] \\
&= \mathbb{E}_{q_\phi(z_{1:T}|x)}\left[\log\left(\frac{p(x, z_{1:T})}{q_\phi(z_{1:T}|x)}\right)\right]
\end{aligned} \tag{15}$$

Question 12

The Variational Diffusion Model is a Markovian hierarchical variational autoencoder following three key restrictions. The encoder is predefined as a linear Gaussian model, $q_\phi(z_t|z_{t-1}) = \mathcal{N}(z_t; \sqrt{\alpha_t} z_{t-1}, (1-\alpha_t)I)$, where α_t controls the information retained from z_{t-1}. At the final timestep T, the latent variable z_T follows a standard Gaussian distribution, $p(z_T) = \mathcal{N}(z_T; 0, I)$. The forward process, known as diffusion, gradually transforms the data $x = z_0$ into z_T using the Markovian encoder transitions. The reverse process, known as generation, reconstructs z_0 from z_T through a learnable Gaussian decoder $p_\theta(z_{t-1}|z_t)$. The inputs of the model are the initial data $x = z_0$, the hyperparameters α_t, and the number of timesteps T. The outputs are the diffused latent representation z_T and the reconstructed data $\hat{x}$.

Question 13

The reconstruction term measures how well the reconstructed data z_0 aligns with the true input data, conditioned on the latent variable z_1. The prior matching term regularizes the latent variable z_T at the final timestep T, to make sure it closely follows the prior distribution, commonly taken as $\mathcal{N}(0, \mathbf{I})$. The consistency term ensures a certain level of consistency between the encoder's forward process $q(z_t|z_{t-1})$ and the decoder's reverse process $p(z_t|z_{t+1})$.

Question 14

Let us show this property by recursion. Set $\bar{\alpha}_t = \prod_{k=1}^{t} \alpha_k$. For $t = 1$, we have by equation (25):

$$z_1 = \sqrt{\alpha_1} z_0 + \sqrt{1-\alpha_1}\epsilon, \tag{16}$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. We have that z_1 is Gaussian and that $\mathbb{E}[z_1] = \sqrt{\bar{\alpha}_1} z_0$, $\mathrm{Cov}(z_1) = (1-\bar{\alpha}_1)\mathbf{I}$. Let us now assume that our hypothesis holds true for t and let us show that it remains true for $t+1$:

$$z_{t+1} = \sqrt{\alpha_{t+1}} z_t + \sqrt{1-\alpha_{t+1}}\epsilon \tag{17}$$

Since z_{t+1} is the sum of two Gaussian random vectors, we know that it is also Gaussian. Moreover:

$$\mathbb{E}[z_{t+1}] = \mathbb{E}[\sqrt{\alpha_{t+1}}z_t + \sqrt{1-\alpha_{t+1}}\epsilon] = \sqrt{\alpha_{t+1}}\sqrt{\bar{\alpha}_t}z_0 = \sqrt{\bar{\alpha}_{t+1}}z_0 \tag{18}$$

$$\begin{aligned} \text{Cov}(z_{t+1}) &= \text{Cov}(\sqrt{\alpha_{t+1}}z_t + \sqrt{1-\alpha_{t+1}}\epsilon) \\ &= \alpha_{t+1}\text{Cov}(z_t) + (1-\alpha_{t+1})\text{Cov}(\epsilon) \\ &= \alpha_{t+1}(1-\bar{\alpha}_t)\mathbf{I} + (1-\alpha_{t+1})\mathbf{I} \\ &= (1-\bar{\alpha}_{t+1})\mathbf{I} \end{aligned} \tag{19}$$

Thus, $z_{t+1} \sim \mathcal{N}(z_{t+1}; \sqrt{\bar{\alpha}_{t+1}}z_0, (1-\bar{\alpha}_{t+1})\mathbf{I})$.

Question 15

By equations (29) and (30):

$$\mu_\theta - \mu_q = \frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)\hat{z}_\theta(z_t,t) - \sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)z_0}{1-\bar{\alpha}_t} = \frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)}{1-\bar{\alpha}_t}(\hat{z}_\theta(z_t,t) - z_0) \tag{20}$$

Thus, equation (28) can be written as:

$$\arg\min_\theta D_{KL}(q(z_{t-1}|z_z, z_0)||p_\theta(z_{t-1}|z_t)) = \arg\min_\theta \frac{\bar{\alpha}_{t-1}(1-\alpha_t)^2}{2\sigma_q^2(t)(1-\bar{\alpha}_t)^2}||\hat{z}_\theta(z_t,t) - z_0||_2^2 \tag{21}$$

The final optimization term can be regarded as a weighted mean squared error loss between the prediction $\hat{z}_\theta(z_t,t)$ and the true latent variable $x = z_0$.

Question 16

For multivariate Gaussian vectors following $\mathcal{N}(\mu, \Sigma)$, the SNR is given by:

$$\text{SNR} = \frac{||\mu||_2^2}{\text{Tr}(\Sigma)} \tag{22}$$

Thus, at timestep t, the SNR for $q(z_t|z_0)$ is given by:

$$\text{SNR}_t = \frac{\bar{\alpha}_t||z_0||_2^2}{(1-\bar{\alpha}_t)D} \tag{23}$$

Question 17

By recalling that $\sigma_q^2(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}$, equation 21 can be rewritten as:

$$\begin{aligned} \arg\min_\theta \frac{\bar{\alpha}_{t-1}(1-\alpha_t)^2}{2\frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}(1-\bar{\alpha}_t)^2}||\hat{z}_\theta(z_t,t) - z_0||_2^2 &= \arg\min_\theta \frac{\bar{\alpha}_{t-1}(1-\alpha_t)}{2(1-\bar{\alpha}_{t-1})(1-\bar{\alpha}_t)}||\hat{z}_\theta(z_t,t) - z_0||_2^2 \\ &= \arg\min_\theta \text{SNR}_{t-1}\frac{D(1-\alpha_t)}{2||z_0||_2^2(1-\bar{\alpha}_t)}||\hat{z}_\theta(z_t,t) - z_0||_2^2 \end{aligned} \tag{24}$$

The SNR at timestep t controls how much the noise affects the optimization. When the SNR is high, the focus is on reconstructing z_0 accurately. When the SNR is low, the focus shifts to modeling the noise accurately.

Question 18

To test diffusion in practice, I used the schedulers LMSDiscreteScheduler, EulerDiscreteScheduler, EulerAncestralDiscreteScheduler and DPMSolverMultistepScheduler, and generated the images with different number of inference steps (10, 100, 1000), using the same seed (99) for each image. The prompt that was used is "A photograph of an astronaut riding a horse on Mars, high resolution, high definition.". While three of the schedulers (LMSDiscreteScheduler, EulerDiscreteScheduler, and DPMSolverMultistepScheduler) led to similar results with slight variations in sharpness and detail, the scheduler EulerAncestralDiscreteScheduler produced images that were more visually diverse. This difference arises because EulerAncestralDiscreteScheduler introduces higher variability at each timestep, which can result in more creative outputs but less consistency with the original prompt. Meanwhile, the number of inference steps played a key role in the quality of the generated images. With fewer steps (10), the images appeared blurry and lacked detail, while increasing the number of steps (100, 1000) progressively enhanced the sharpness, of the images.

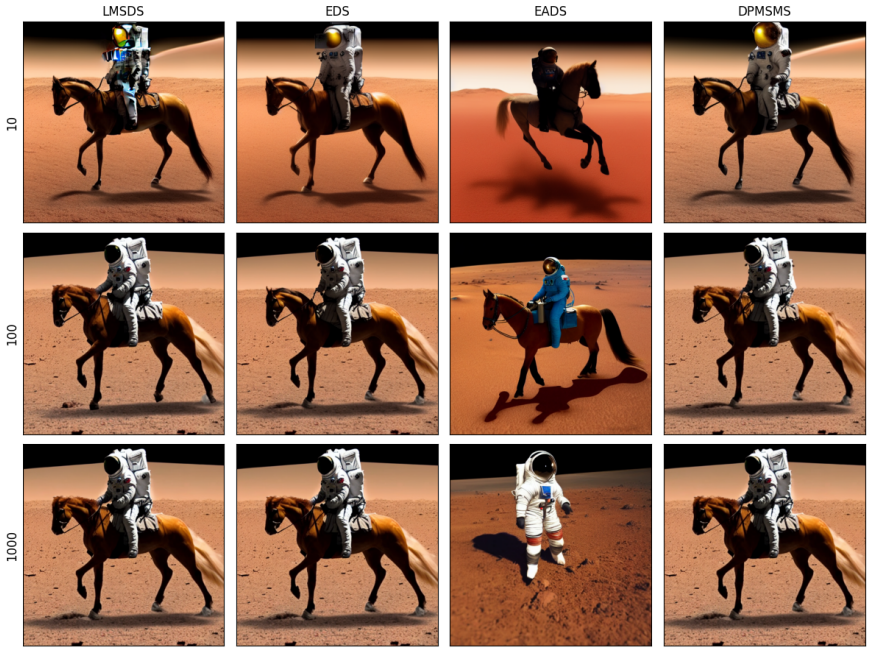


Figure 1: Generated images for different number of steps and different noise schedulers (seed=99)

Question 19

To test the effect of a "negative" prompt on the generated images, I used the (positive) prompt "A photograph of an astronaut riding a horse on Mars, high resolution, high definition.", while the negative prompt was "desert". Using a fixed seed (42), I generated two images, one that doesn't consider the negative prompt and one that does. The position of the astronaut and the horse remained consistent in both images, showing that the negative

prompt primarily influenced the background. This suggests that the diffusion model is effective at making changes to specific elements without significantly modifying the main focus of the image.

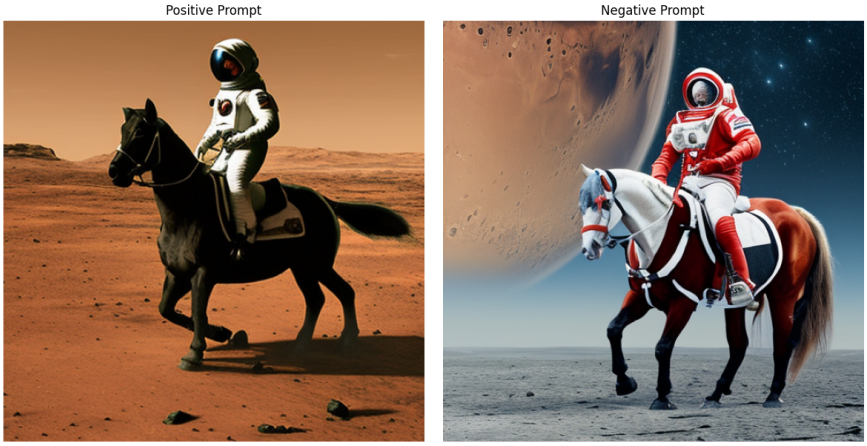


Figure 2: Positive prompt vs. negative prompt (seed=42)

3 Multi-lingual Translation

Question 20

Let us first derive the impact of x_l on x_{l+1} when considering post-normalization:

$$x_{l+1} = LayerNorm(x_l + \mathcal{A}(x_l)) \tag{25}$$

$$\frac{\partial x_{l+1}}{\partial x_l} = \frac{\partial LayerNorm(x_l + \mathcal{A}(x_l))}{\partial (x_l + \mathcal{A}(x_l))} \cdot \left(1 + \frac{\partial \mathcal{A}(x_l)}{\partial x_l}\right) \tag{26}$$

Next, let us derive the impact of x_l on x_{l+1} when considering pre-normalization:

$$x_{l+1} = x_l + \mathcal{A}(LayerNorm(x_l)) \tag{27}$$

$$\frac{\partial x_{l+1}}{\partial x_l} = 1 + \frac{\partial \mathcal{A}(LayerNorm(x_l))}{\partial x_l} = 1 + \frac{\partial \mathcal{A}}{\partial LayerNorm(x_l)} \cdot \frac{\partial LayerNorm(x_l)}{x_l} \tag{28}$$

Then the impact of x_l on e is given by:

$$\begin{aligned}
\frac{\partial e}{\partial x_l} &= \frac{\partial e}{\partial x_L}\frac{\partial x_L}{\partial x_l} \\
&= \frac{\partial e}{\partial x_L}\frac{\partial x_L}{\partial x_{L-1}}\frac{\partial x_{L-1}}{\partial x_{L-2}} \cdots \frac{\partial x_{l+1}}{\partial x_l} \\
&= \frac{\partial e}{\partial x_L}\prod_{k=l}^{L-1}\frac{\partial x_{k+1}}{\partial x_k} \\
&= \begin{cases} \frac{\partial e}{\partial x_L}\frac{\partial LayerNorm(x_l+\mathcal{A}(x_l))}{\partial (x_l+\mathcal{A}(x_l))} \cdot \left(1 + \frac{\partial \mathcal{A}(x_l)}{\partial x_l}\right) & \text{for post-normalization} \\ \frac{\partial e}{\partial x_L}\left(1 + \frac{\partial \mathcal{A}}{\partial LayerNorm(x_l)} \cdot \frac{\partial LayerNorm(x_l)}{x_l}\right) & \text{for pre-normalization} \end{cases}
\end{aligned} \tag{29}$$

Question 21

When applying normalization before applying the attention mechanism $\mathcal{A}$ (i.e. pre-normalization), we ensure that the inputs to $\mathcal{A}$ are centered and have a controlled variance, which reduces the risk of exploding or vanishing gradients upon backpropagation. Thus, gradients flowing backward through $\mathcal{A}$ are less sensitive to extreme variations in x_l, which leads to more stable and consistent gradient updates.

On the other hand, the residual connection and attention transformation $\mathcal{A}$ are computed before normalization in the case of post-normalization, leading to potential gradient explosions or vanishes, especially in very deep models. Overall, this can lead to slow or unstable training, requiring greater care in fine-tuning the learning rate.

Therefore, modern deep architectures prefer to use pre-normalization over post-normalization.

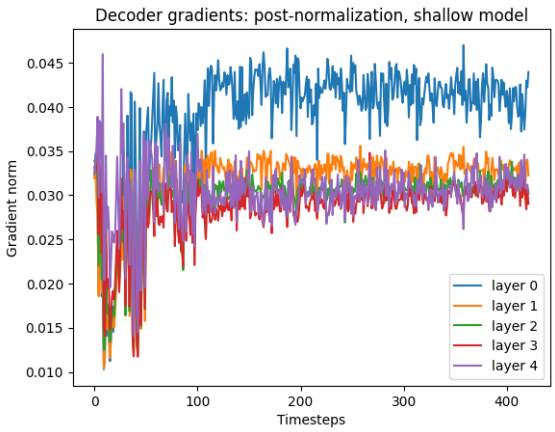
Question 22

In order to compare the impact of both normalization strategies, I decided to work with the norm of the gradient after each transformer layer. This allowed me to get a better understanding of the gradient's variations in deep and shallow models.

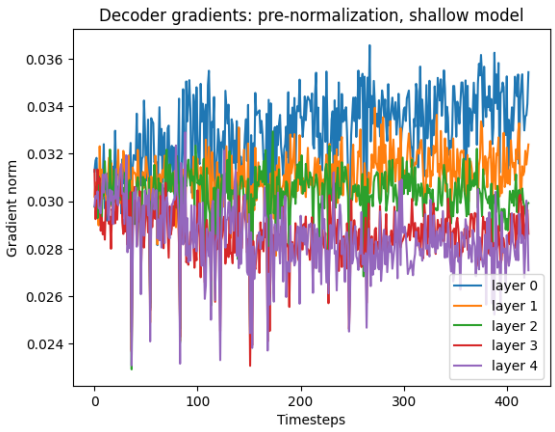
For shallow models, I observed that post-normalization quickly leads to gradients with a stable norm, meaning that after a certain number of timesteps, the gradient norm stabilizes at a consistent value (see figure 3a). On the other hand, for pre-normalization in shallow models, the gradient norm remains relatively stable throughout all timesteps right from the beginning (see figure 3b).

For deep models, the difference between the two normalization strategies becomes more pronounced. When considering post-normalization, the gradients show more variability during the early stages of training, and it takes longer for the gradient norm to stabilize (see figure 4a). Meanwhile, pre-normalization shows a significant advantage in deeper models, as the gradient norm remains relatively stable throughout all training steps (see figure 4b).

As a result, this empirical comparison highlights how pre-normalization may be better suited for deeper models, while both strategies perform comparably well for shallow models.

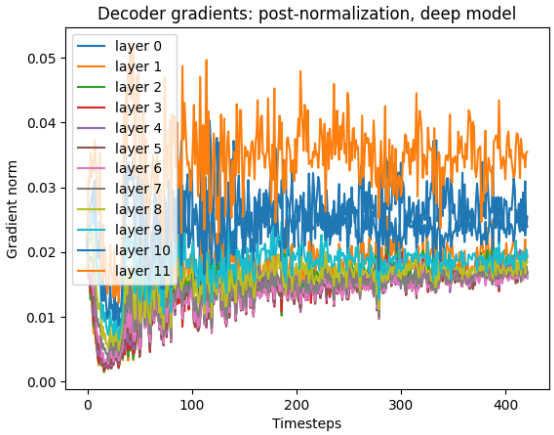


(a) Post-normalization

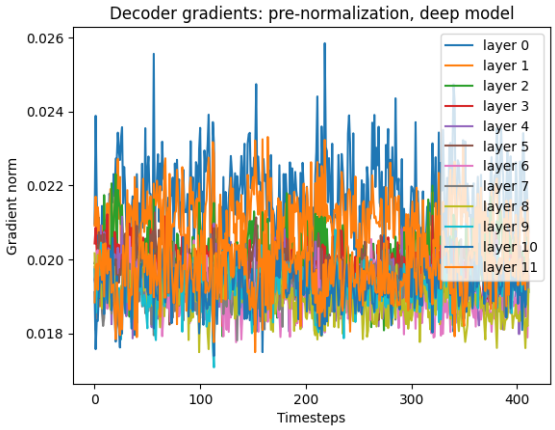


(b) Pre-normalization

Figure 3: Shallow model (5 layers)



(a) Post-normalization



(b) Pre-normalization

Figure 4: Deep model (12 layers)