3MD3020: Deep Learning
MVA Master

Assignment 1

Thiziri NAIT SAADA

Due: January 25, 2020

A. Computer Vision Part

0.1 Latent Variable Models

0.2 Decoder: The Generative Part of the VAE

Question 1 [5 points] The steps needed to sample from such a model is called ancestral sampling and consists in sampling from the parent node first. It is defined as follows :

- Sample z_n from a D-dimensional Gaussian noise $p(z_n) \sim \mathcal{N}(0, \mathcal{I}_D)$
- Feed the generator function f_θ with z_n : its output is a M - dimensional vector $f_\theta(z_n) = ([f_\theta(z_n)]_m)_{1 \leq m \leq M}$
- Each of the M coordinates of the generated sample $x_n = (x_n^m)_{1 \leq m \leq M}$ is given by sampling from a Bernoulli distribution: $\forall m \in [\![1, M]\!], x_n^m \sim Ber([f_\theta(z_n)]_m)$ such that $x_n \in \{0,1\}^M$

Question 2 [5 points] Monte Carlo integration is not used for training VAE because :

- We would need to compute several times $p(x_n|z_n)$
- For large dimensions of the latent space, $\lim_{M \longrightarrow \infty} p(x_n|z_n) = 0$.
  Let us take an example of $\forall m \in [\![1, M]\!], x_n^m \sim Ber(\frac{1}{2})$.
  Then, $p(x_n|z_n) = \frac{1}{2}^M \xrightarrow[M \to \infty]{} 0$.
  So, to avoid sampling over the latent space vectors that will finally have $p(x_n|z_n)$ near to 0, the idea of the VAE is to sample from the latent space only the hidden variables that are likely to have produced x_n. Thus, only hidden vectors such that $p(x_n|z_n)$ is not 0 are sampled.

0.3 KL Divergence

Question 3 [5 points] Let $q = \mathcal{N}(\mu_q, \sigma_q^2)$ and $p = \mathcal{N}(\mu_p, \sigma_p^2)$.

- For $(\mu_q, \sigma_q^2) = (0.2, 1)$ and $(\mu_p, \sigma_p^2) = (0, 1)$, we have $KL(p||q) = 0.02$, a small divergence
- For $(\mu_q, \sigma_q^2) = (150, 1)$ and $(\mu_p, \sigma_p^2) = (0, 1)$, we have $KL(p||q) = 11250$, a large divergence

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

Question 4 [5 points] We have, for any pair of distributions a, b, $KL(a||b) \geq 0$ such that :

$$\log p(x_n) \geq \log p(x_n) - KL(q(Z|x_n)||p(Z|x_n))$$

Thus, $\mathbb{E}_{q(z_n|x_n)}[\log p(x_n|z_n) - KL(q(Z|x_n)||p(Z))] = \log p(x_n) - KL(q(Z|x_n)||p(Z|x_n))$ is a lower bound of the marginal log likelihood $\log p(x_n)$.

To show that $KL(a||b) \geq 0$, let us apply Jensen inequality to the concave function $\mathbb{E}(\phi(X)) \geq \phi(\mathbb{E}(X))$ with $\phi = \log$:

$$KL(a||b) = -\,\mathbb{E}_{a(x)}(\log \frac{b(x)}{a(x)}) \geq -\log(\mathbb{E}_{a(x)}(\frac{b(x)}{a(x)})) = -\log \int a(x)\frac{b(x)}{a(x)}dx = -\log(1) = 0$$

This lower bound is tractable and computationally reasonable whilst the marginal log likelihood $\log p(x_n)$ is intractable.

Question 5 [5 points] If the lower bound is pushed up :

- $KL(q(Z|x_n)||p(Z|x_n)) = 0$, meaning that almost surely $q(Z|x_n) = p(Z|x_n)$. So, the approximate variational distribution equals the true latent posterior distribution, which was our goal.
- The marginal likelihood is maximized $\log p(x_n) = \mathbb{E}_{p(z|x_n)} \log p(x_n) = \frac{p(x_n,z)}{p(z|x_n)}$ We have an unbiased estimate of the marginal likelihood :
$$\hat{p}_\theta(x) = \frac{p_\theta(x,z)}{q_\phi(z|x)}$$

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

Question 6 [5 points]

- **Regularization loss** $= KL(q_\phi(Z|x_n)||p_\theta(Z))$. It is the part of the loss that tells us how far from the prior latent distribution the approximate posterior is. It enforces the encoder to capture meaningful representations of the data and to avoid overfitting it.
- **Reconstruction loss** = $-\mathbb{E}_{q_\phi(Z|x_n)}(\log p_\theta(x_n|Z)) = -\int \log p_\theta(x_n|z) \underbrace{q_\phi(z|x_n)}_{\text{compensation}} dz$

  In this integral, there is an informative term, called compensation, on how likely is the latent variable z to have produced x_n. If it is close to 0, this $p_\theta(x_n|z)$ won't be taken into account in the Monte Carlo integration approximation. So, again, the integration is only performed over latent variables that are responsible of the observations. $\log p_\theta(x_n|z)$ refers to a standard decoding penalty term interpreted in terms of reconstruction error.

Question 7 [5 points] Let us put everything together :

- Considering an image input data x_n, we feed the encoder with x_n and it returns two moments $\mu_\phi(x_n), \Sigma_\phi(x_n)$ of the gaussian distribution of $q_\phi(z|x_n) \sim \mathcal{N}(z; \mu_\phi(x_n), \Sigma_\phi(x_n))$
- Computing the regularization loss (recall $p_\theta(Z) = \mathcal{N}(z; 0, \mathcal{I}_D)$)
$$\begin{aligned} L_n^{reg} =& KL(q_\phi(Z|x_n)||p_\theta(Z)) \\ =& \frac{1}{2}[\mu_\phi(x_n)^T \mu_\phi(x_n) + tr(\Sigma_\phi(x_n)) - D - \log|\Sigma_\phi(x_n)|] \end{aligned}$$
- Sample $z_n \sim q_\phi(z|x_n) \sim \mathcal{N}(z; \mu_\phi(x_n), \Sigma_\phi(x_n))$
- Feeding the generator function f_θ with z_n : its output is a M - dimensional vector $f_\theta(z_n) = ([f_\theta(z_n)]_m)_{1 \le m \le M}$
- Each of the M coordinates of the generated sample $x_n = (x_n^m)_{1 \le m \le M}$ is given by sampling from a Bernoulli distribution: $\forall m \in [\![1, M]\!], x_n^m \sim Ber([f_\theta(z_n)]_m)$ such that $x_n \in \{0,1\}^M$
- Computing the regularization loss, which is the cross-entropy between the original observation and its reconstruction by the decoder.
$$\begin{aligned} L_n^{recon} &= -\mathbb{E}_{q_\phi(Z|x_n)}(\log p_\theta(x_n|Z)) \\ &= -\sum_{m=1}^{M} x_n^m \log([f_\theta(z_n)]_m) + (1 - x_n^m)\log([1 - f_\theta(z_n)]_m) \end{aligned}$$
- Computing the total loss :
$$L_n = L_n^{recon} + L_n^{reg}$$
- We proceed exactly the same way when considering instead of a single observation x_n a batch of N observations $(x_1, ..., x_N)$ and we take the average loss $L_N = \frac{1}{N}\sum_{n=1}^{N} L_n$

0.6 The Reparametrization Trick

Question 8 [5 points] Sampling operations are non differentiable. So, when sampling over the latent variables, the back-propagation cannot flow through the random node $z \sim \mathcal{N}(z; \mu_\phi(x_n), \Sigma_\phi(x_n))$. Thus, we cannot go backward and compute the derivatives with respect to parameter ϕ to derive backpropagation.

Therefore, the reparametrization trick allows to put all the randomness on a vector ε that does not depend on the parameters that are to be optimized (on which we would like to perform the backpropagation). In other words, we have removed the randomness of the parameter by introducing another random node, independent from the parameter. So backpropagation is now possible as it flows only through deterministic nodes now.

Additionally, we didn't lose any information, because $\begin{cases} z \sim \mathcal{N}(z; \mu_\phi(x_n), \Sigma_\phi(x_n)) \\ z = \mu_\phi(x_n) + \Sigma_\phi(x_n)^{\frac{1}{2}}\varepsilon, \text{ with } \varepsilon \sim \mathcal{N}(0, \mathcal{I}_D) \end{cases}$ are strictly equivalent sampling operations.

However, we can now compute :

$$L_n = -\mathbb{E}_{\varepsilon \sim \mathcal{N}(0,\mathcal{I}_D)}(\log(p_\theta(x_n|Z) + KL(q_\phi(Z|x_n)||p_\theta(Z))$$

and its derivative :

$$\nabla_\phi L_n = -\mathbb{E}_{\varepsilon \sim \mathcal{N}(0,\mathcal{I}_D)}[\frac{1}{p_\theta(x_n|Z)}\nabla_\phi p_\theta(x_n|Z)] + \nabla_\phi KL(q_\phi(Z|x_n)||p_\theta(Z))$$

0.7 Putting things together: Building a VAE

Question 9 [5 points] Please find the implementation in the file Deep-Assignment.ipynb for further details.

I used to take as encoder and decoder simple MLP with 512 hidden units and trained the model on the MNIST dataset on 100 epochs. The latent space dimension is set to 20. The optimizer is chosen to be Adam Optimizer with a learning rate $\lambda_r = 1e^{-}4$, a batch size of 64 samples. The training is performed using Google Colab GPUs.

Question 10 [5 points] Here are random generated samples at three checkpoints of the training :

- **Before training** : it is random noise 1

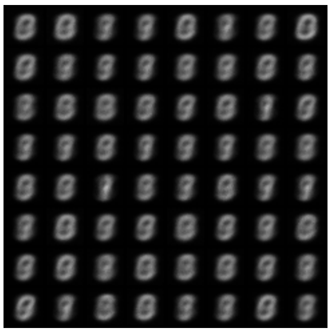

Figure 1: Random Generated samples from VAE epoch 0

- **After training 10 epochs** : the model starts to learn some representation features of the data (white digits centered on a dark background). It needs more training to get satisfying results, especially to lighter the blurry effect on images 2.

- **After training 80 epochs** We have less blurry effect. The results look promising but still not perfect. To get better results, one would need to rethink its model, but for now, the training can be assumed to be finished, it is very unlikely to have better results than we have with such a setting (to convince yourself, looking at the loss behaviours through the training process is a good idea) 3.

Note that these samples are generated by :

- sample z from a random multivariate noise
- decode the sample z to get a generated sample through the decoder function, with weights optimized until the epoch number that is indicated.

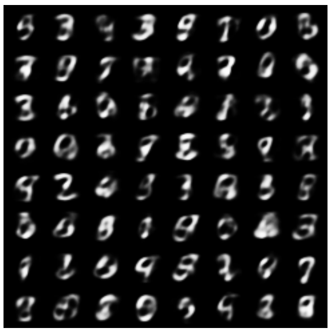

Figure 2: Random Generated samples from VAE epoch 10

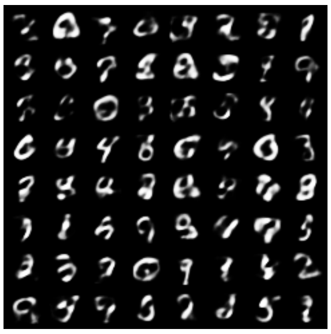

Figure 3: Random Generated samples from VAE epoch 80

(Optional) Question 11 [10 points] I implemented a Convolutional VAE, with a structure detailed in the tables 1 2. By observing the loss behaviour, one can notice that the loss decreases faster than when performing the standard VAE. This explains why the quality of the generated samples looks more satisfying than with the Standard GAN (please see 7) as they seem less blurry. They give promising results, which can explain why VAE are still of interest in research nowadays.

CNN Encoder
3x3 Convolution Layer ReLU (channels=8, padding=1)
3x3 Convolution Layer ReLU (channels=16, padding=1)
3x3 Convolution Layer ReLU (channels=32, padding=1)
3x3 Convolution Layer ReLU (channels=64, padding=1)
Flatten
mu = Linear Layer (latent dimension)
variance = Linear Layer (latent dimension)

Table 1: CNN Encoder

B. Natural Language Processing Part

0.8 RNNs

Question 12. [10 points] The seq2seq I implemented, inspired by the pytorch tutorial that the instructors recommended seem to behave a little bit strangely... Attention is globally distributed along the upper diagonal, which makes me think that maybe there is a shift in the model or in the visualisation... If it was distributed along the diagonal, I could have argued that this means the model is mainly focused on the last token of the sentence, and have not really reached its final goal which was to take into account parts of the sentence that were meaningful. Please see my code to have all the details of the implementation.

Note that through the training process, the loss decreases, which indicates that the likelihood is being maximized, or at least, it increases 8.

I plotted the attention matrices for several examples :

We can say that the model performs heterogeneously (see the examples), it can sometimes tackle some translation issues such as the way to give your name in English is totally different from the manner to say it in French. However, the model overcame this challenge, so we could infer that there was, in the training data, a sentence that was very similar to it... On

CNN Decoder
3x3 Convolution Layer ReLU (channels=32, padding=1)
3x3 Convolution Layer ReLU (channels=16, padding=1)
3x3 Convolution Layer ReLU (channels=8, padding=1)
3x3 Convolution Layer ReLU (channels=1, padding=1)
Sigmoid

Table 2: CNN Decoder

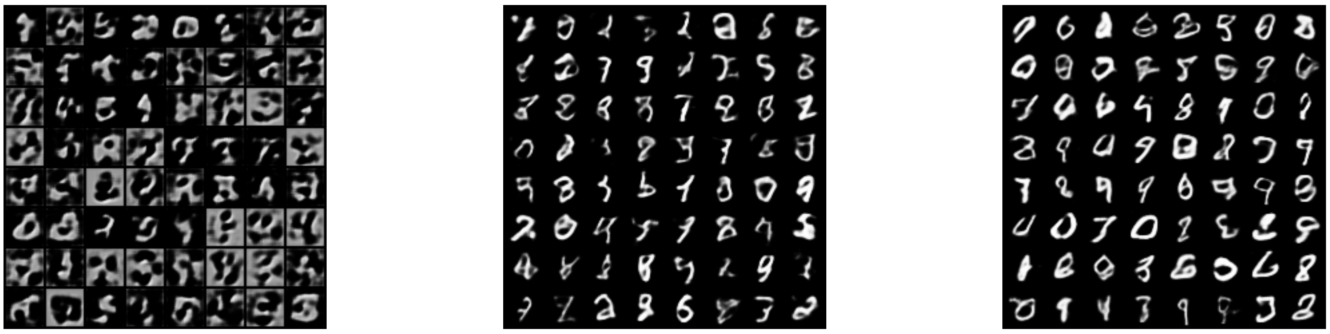
(a) Epoch 0 **(b)** Epoch 10 **(c)** Epoch 80

Figure 4: Random Generated samples from CNN VAE

the other hand, on unknown data, the translations given by the model are not really satisfying even though the results are still promising...

Concerning the EOS and SOS tokens :

- The SOS token is the starter point that makes aware the decoder the sentence has started.
- The EOS token gives the end of the sentence. If we remove it, we observed that a natural limit to the sentence is set up. This limit corresponds to the maximal length of the training data sentences (set to 10 here) because the model has never seen larger sentences. When removed, the EOS token is replaced by a serie of '.', the token that is the most observed in the training data, until reaching the maximal length sentence.

Question 13. [5 points] Without attention, the context of the whole sentence would never have been taken into account by the decoder. Here, some parts of the sentence can be more important to predict a given state, and conversely, some parts are less informative. That is why we put a different weight on each of the token that is put under scrutiny.

Question 14. [5 points] Teaching forcing consists in giving the target output as the hidden input state along the sequence instead of the predicted state by the previous unit. This way, the training converges faster (but might be unstable when dealing with new unknown sequences).

0.9 Transformers

Question 15. [15 points]

- Let us consider we do the normalization layer before self-attention, that is :

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

Then, we can use the chain rule to derive the backpropagation. If we denote L the length of the sequence, e the loss, then, $\forall l \in [\![1, L]\!]$:

$$\begin{aligned}\frac{\partial e}{\partial x_l} =& \frac{\partial e}{\partial x_L}\frac{\partial x_L}{\partial x_l}\\ =& \frac{\partial e}{\partial x_L}\frac{\partial x_L}{\partial x_{L-1}}\frac{\partial x_{L-2}}{\partial x_{L-1}}...\frac{\partial x_{l+1}}{\partial x_l}\\ =& \frac{\partial e}{\partial x_L}(1+\frac{\partial \mathcal{A}(x_{L-1})}{\partial x_{L-1}})(1+\frac{\partial \mathcal{A}(x_{L-2})}{\partial x_{L-2}})...(1+\frac{\partial \mathcal{A}(x_l)}{\partial x_l})\\ =& \frac{\partial e}{\partial x_L}\prod_{i=l}^{L-1}(1+\frac{\partial \mathcal{A}(x_i)}{\partial x_i})\end{aligned} \tag{1}$$

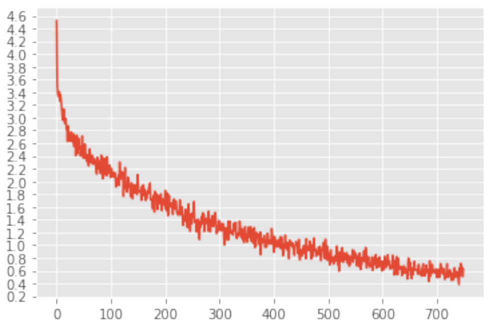

Figure 5: Loss decreasing during the training

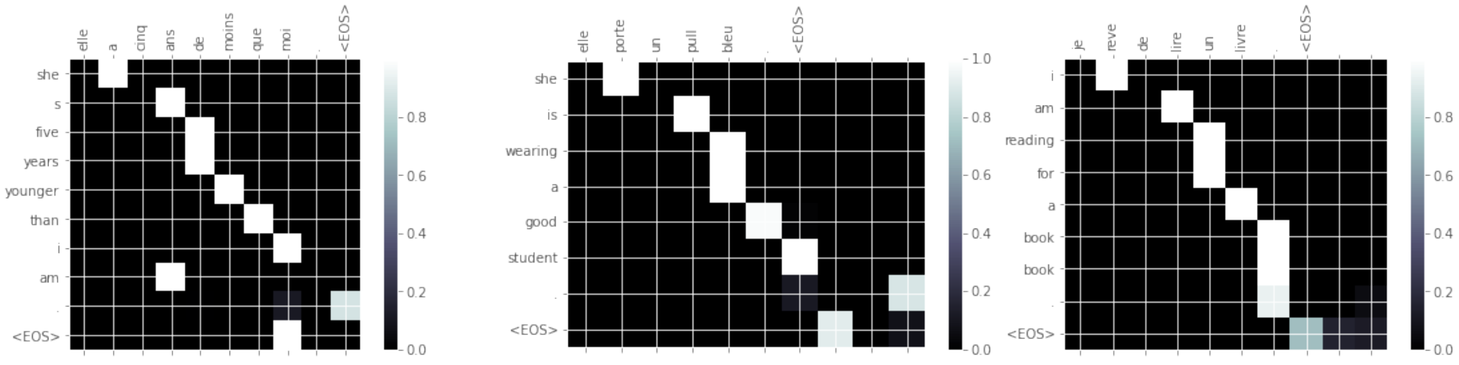

Figure 6: Example of Attention matrices given by Seq2seq model

where we used : $\forall l \in [\![1, L]\!], \quad \frac{\partial x_{i+1}}{\partial x_i} = 1 + \frac{\partial \mathcal{A}(x_i)}{\partial x_i}$

- If we do the layer normalization after self-attention :

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

Then, same computation gives $\forall l \in [\![1, L]\!]$:

$$\begin{aligned} \frac{\partial e}{\partial x_l} &= \frac{\partial e}{\partial x_L} \frac{\partial x_L}{\partial x_l} \\ &= \frac{\partial e}{\partial x_L} \frac{\partial x_L}{\partial x_{L-1}} \frac{\partial x_{L-2}}{\partial x_{L-1}} \dots \frac{\partial x_{l+1}}{\partial x_l} \\ &= \frac{\partial e}{\partial x_L} \prod_{i=l}^{L-1} \frac{\partial LayerNorm(u)}{\partial u} LayerNorm(1 + \frac{\partial \mathcal{A}(x_i)}{\partial x_i}) \end{aligned} \tag{2}$$

Question 16. [15 points] We can notice from equation 1 that doing the normalization layer before self attention leads us to consider a product of a quantity that is larger than 1 (Attention function considered to be non negative). It is an issue since we know that the limit of a geometric sequence with reason larger than 1 diverges. So, if the length of the sequence grows to infinity, the gradient of the loss function is very likely to explode, leading to computational issues in the updates. On the contrary, in 2 the quantity that is likely to be greater than 1 is normalized and then we take the product, so there is no evidence that it will lead to computational issues such as vanishing or exploding gradients.

Therefore, the second option makes the backpropagation easier than the first one.

(Optional) Question 17. [20 points]

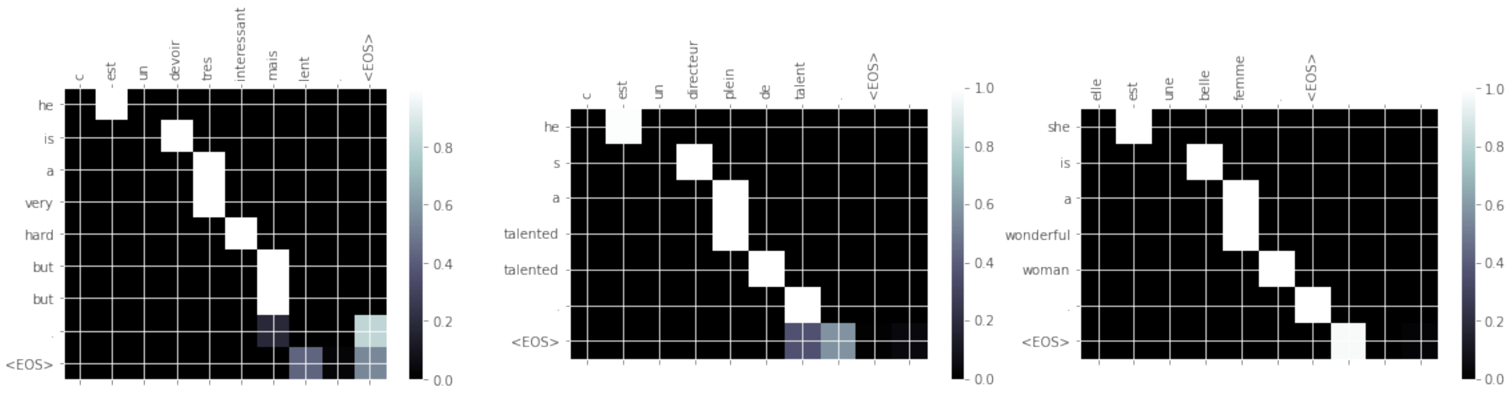

Figure 7: Attention matrices given by Seq2seq model

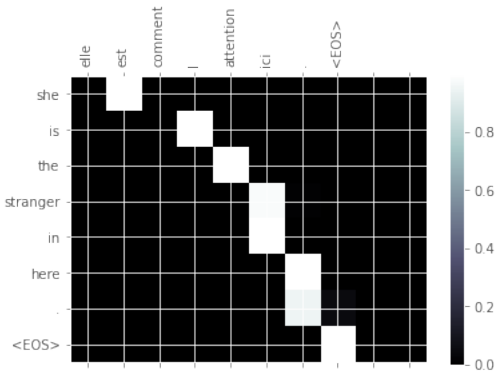

Figure 8: Example of Attention matrix given by Seq2seq model