Master MVA

Deep Learning 3MD3020

Assignment Report

January 2020

Author:
Clément BONNET

Supervised by Professors:
Vincent LEPETIT
Maria VAKALOPOULOU

Contents

Computer Vision Part

1 Decoder: The Generative Part of the VAE

1.1 Question 1

The goal is to build a dataset of random images. The difficulty is that the distribution of the images X is unknown. However, we know the distribution of Z and $X|Z$, Z being a latent variable. We can therefore use the ancestral sampling method to solve our problem. The key steps to generate a random image are the following:

- Drawing a latent sample z_n from the latent variable distribution Z, which is known.
- Estimating $f_\theta(x_n)$ to have an estimation of the parameters of the distribution of X
- By taking the m^{th} component of the vector of parameters estimated above, we can simulate each pixel, with a Bernoulli distribution, in our case.

This technique enables us to generate random images with black and white pixels.

1.2 Question 2

Monte-Carlo methods rely on repeating random sampling a large amount of time to obtain a good numerical approximation of a deterministic quantity. In our case, we wish to estimate the probability of an image $p(x_n)$.

However, to have an accurate result, one condition is to have drawn enough samples. The required amount of sample grows exponentially with the size of the space in which the variables we simulate live. In our case, we simulate first a random variable z in a D-dimensional space, and then a random image from the random z quantity that lives in an N-dimensional space. Therefore, the space to cover, of dimension $N \times D$ is huge.

This drawback makes Monte-Carlo methods inefficient to estimate $p(x_n)$ because we would need to sample too many times to properly cover the space and obtain a meaningful approximation. Such inefficiency increases when the dimension D increases.

2 KL Divergence

2.1 Question 3

Using the log inequality: $\log(1+x) \leq x$, we can show that the KL-divergence is always positive, which is consistent with the fact that we consider it as being a distance between two statistical distributions.

Therefore, a small value of the KL-divergence between two distributions p and q is reached if they are close to 0. The quantity $-q(x)\log\left(\frac{p(x)}{q(x)}\right)$ being always positive, the integration of such quantity will be close to 0 if and only if the function itself is close to 0. Since $log(1) = 0$, having $p = q$ leads to $D_{KL}(p||q) = 0$, which is the smallest value we can obtain.

If we have $p \sim \mathcal{N}(0, 1)$, and $q \sim \mathcal{N}(\mu, \sigma)$, then we can use the analytical expression of the KL-divergence given below:

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

From this expression, we can easily make it very large, by either increasing the absolute value of the mean or the variance. Thus, having $|\mu_q| \gg 1$ or $|\mu_q| \gg 1$ gives us a large value of the KL-divergence between p and q.

3 The Encoder: - Efficiently evaluating the integral

3.1 Question 4

As mentioned above, the KL-divergence is a positive quantity, therefore, we can deduce from eq.(16) that :

$$\log(p(x_n)) \geq E_{q(z_n|x_n)}\left(log(p(x_n|z_n))\right) - D_{KL}(q(Z|x_n)||p(Z))$$

As a consequence, the right-hand side of the equation 16 is indeed a lower-bound of the log-probability $log(p(x_n))$.

We choose to optimize the lower bound, instead of the log-probability directly because it can be computed, whereas the log-probability cannot.

3.2 Question 5

Since we have $\log(p(x_n)) - D_{KL}(q(Z|x_n)||p(Z|x_n)) = E_{q_{(z|x_n)}}\left(log(p(x_n|z_n))\right) - D_{KL}(q(Z|x_n)||p(Z))$, two things can happen if the value of the lower-bound is pushed up:

- The log-probability $\log(p(x_n))$ increases, which is our initial goal: make the data more likely to be observed by our model.
- Or the KL-divergence $D_{KL}(q(Z|x_n)||p(Z|x_n))$ decreases, which means that $q(Z|x_n)$ becomes closer to $p(Z|x_n))$. It is also interesting because it allows us to efficiently estimate $\log(p(x_n))$

4 Specifying the Encoder

4.1 Question 6

The loss used to optimize the evidence lower bound can be written in the following way:

$$\mathbf{L}(\theta, \phi) = -\frac{1}{N}\sum_{n=1}^{N} E_{q_\phi(z|x_n)}\left(\log(p_\theta(x_n|z))\right) - D_{KL}(q_\phi(Z|x_n)||p_\theta(Z)) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{L}_n^{\text{recon}} + \mathbf{L}_n^{\text{reg}}$$

The first term is $-E_{q_\phi(z|x_n)}\left(log(p_\theta(x_n|z))\right)$. Let us suppose we use only one sample to estimate it. We would first draw a z using the variational distribution $q_\phi(z|x_n)$. We would then compute $\log(p_\theta(x_n|z)$.

If we wanted to reconstruct x_n using the encoder-decoder network, we would follow the same steps. So, $p_\theta(x_n|z)$ would reflect the probability of getting the same image we used at the beginning, and it would be close to 1 if the reconstruction's quality is good, and close to 0 if it is not. Thus, after taking the log, the quantity $-E_{q_\phi(z|x_n)}\left(\log(p_\theta(x_n|z))\right)$ will be close to 0 if the reconstruction has a good quality and higher otherwise.

Hence, we can see this quantity as the loss of the image reconstruction x_n. We define $\mathbf{L}_n^{\text{recon}} = -E_{q_\phi(z|x_n)}\left(\log(p_\theta(x_n|z))\right)$.

Now, if one focuses on the second term: $D_{KL}(q_\phi(Z|x_n)||p_\theta(Z))$, it measures how close the distributions $p_\theta(Z)$ and $q_\phi(Z|x_n)$ are. However, the first distribution is independent of x_n, thus, minimizing this second term means decreasing the dependence of $q_\phi(Z|x_n)$ with respect to x_n.

Such independence reflects less overfitting of the distribution $q_\phi(Z|x_n)$, and thus a more regular solution. So, minimizing $D_{KL}(q_\phi(Z|x_n)||p_\theta(Z))$ increases the regularity of the variational distribution. We thus define $\mathbf{L}_n^{\text{reg}} = D_{KL}(q_\phi(Z|x_n)||p_\theta(Z))$.

4.2 Question 7

For the first term: $\mathbf{L}_n^{\text{recon}}$, we are going to:

- Express the expected value as an integral,
- Replace $p_\theta(x_n|z)$ by the known Bernoulli distribution,
- Replace $q_\phi(z|x_n)$ by the factored multivariate normal distribution specified in Eq. 18.

We get:

$$
\begin{aligned}
\mathbf{L}_n^{\text{recon}} &= -E_{q_\phi(z|x_n)}\left(\log(p_\theta(x_n|z))\right) \\
&= -\int_{R^D} q_\phi(z|x_n)\log\left(p_\theta(x_n|z)\right)dz \\
&= -\int_{R^D} q_\phi(z|x_n)\sum_{m=1}^{M}\log\left(Bern\left(x_n^m|f_\theta(z)_m\right)\right)dz \\
&= -\int_{R^D} \frac{\exp\left(-\frac{1}{2}(z-\mu_\phi(x_n))^T\Sigma_\phi(x_n)^{-1}(z-\mu_\phi(x_n))\right)}{(2\pi)^{D/2}|\Sigma_\phi(x_n)|^{1/2}}\sum_{m=1}^{M}\log\left(\text{Bern}\left(x_n^m|f_\theta(z)_m\right)\right)dz
\end{aligned}
$$

Finally, we have the expression of the first term:

$$
\mathbf{L}_n^{\text{recon}} = -\sum_{m=1}^{M}\int_{R^D} \frac{\exp\left(-\frac{1}{2}(z-\mu_\phi(x_n))^T\Sigma_\phi(x_n)^{-1}(z-\mu_\phi(x_n))\right)}{(2\pi)^{D/2}|\Sigma_\phi(x_n)|^{1/2}}\log\left(\text{Bern}\left(x_n^m|f_\theta(z)_m\right)\right)dz
$$

Let's now focus on the second term: $\mathbf{L}_n^{\text{reg}}$, we are going to:

- Express the KL-divergence value as an integral,
- Use the fact that these distributions are multivariate normal distributions to compute their KL-divergence.

We also know that:

- $p(z) \sim \mathcal{N}(0, I_D)$
- $q_\phi(z_n|x_n) \sim \mathcal{N}(\mu_\phi(x_n), \text{diag}(\Sigma_\phi(x_n)))$

We are going to use the formula giving us the value of the KL-divergence between two multivariate normal distributions:

$$
D_{KL}\left(\mathcal{N}_0, \mathcal{N}_1\right) = \frac{1}{2}\left(Tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1-\mu_0)^T\Sigma_1^{-1}(\mu_1-\mu_0) - n + \log\left(\frac{|\Sigma_1|}{|\Sigma_0|}\right)\right)
$$

We can finally derive the analytical expression of $\mathbf{L}_n^{\text{reg}}$ using $\mathcal{N}_0 = q_\phi(z_n|x_n)$ and $\mathcal{N}_1 = p(z)$. We obtain the expression of the second term:

$$
\mathbf{L}_n^{\text{reg}} = \frac{1}{2}\left(-\log\left(|\Sigma_\phi(x_n)|\right) - n + \text{Tr}\left(\Sigma_\phi(x_n)\right) + \mu_\phi(x_n)^T\mu_\phi(x_n)\right)
$$

5 The Reparameterization Trick

5.1 Question 8

Sampling is not a differentiable operation, thus we can't perform backpropagation to estimate $\nabla_\phi L(\theta, \phi)$. So, we need to use the reparameterization trick to solve this problem. The idea

is to use the same random seed for the estimation of the expected value when performing backpropagation and to use a linear transformation to adapt the samples to the distribution. By doing so, we eliminate the sampling influence on the final value, and thus it no longer has to be differentiated.

In other words, instead of using, for each iteration of the minimization, N Gaussian samples from a distribution whose parameters change across the iterations, we draw N samples from an $\mathcal{N}(0, 1)$ distribution and we use a linear transform to get N samples needed for each step. This linear transform depends on the value of the parameters of the current Gaussian distribution.

6 Putting things together: Building a VAE

6.1 Question 9

We implement a variational autoencoder in Python to reconstruct binary MNIST images. The encoder and decoder are built using a multi-layer perceptron. In practice, we use Pytorch to implement two linear neural networks for both the encoder and the decoder. The decoder's output is a vector of size 20 since the latent space has a dimension of 20. When we train the model, we use the following hyperparameters:

- $\lambda = 1 \times 10^{-4}$
- Batch size = 64
- Adam optimizer ($\beta_1 = 0.9,\ \beta_2 = 0.999$)

To train the model, we iterate over the training set, and for each image, we pass it through the encoder to get the latent variable z. Then, we compute the mean and variance to perform the reparameterization trick. Finally, we can compute the reconstructed image, the loss related to the image, and optimize the weights of our linear neural networks.

6.2 Question 10

The 3 pictures in figure 1 show the evolution of the reconstruction of the binary MNIST images at epoch 0, epoch 10 and epoch 80.

6.3 Question 11

We now choose to implement a U-net architecture built with convolutional neural networks to improve the performances of the model. Again, we use Pytorch to implement 4 2d-convolutional networks for both the encoder and the decoder. Since the images are gray, we start with one channel as the input of the encoder's first layer for the first one, and 8 as output. We then double the number of channels for each layer. For the decoder, we use the same architecture, but with transposed convolutional layers (U-net architecture). We keep the same hyperparameters as in the linear model. The results are displayed in figure 2

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

Figure 1: Reconstruction of the binary MNIST images during training, for a dense network.

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

Figure 2: Reconstruction of the binary MNIST images during training, for the convolution architecture.

Natural Language Processing Part

7 RNNs

7.1 Question 12

7.1.1 Attention distribution

We train a seq2seq model using the attention mechanism. It enables the model to translate more accurately each word. Indeed, it focuses on the words of the sentence that have the highest impact on the translation. The results are displayed in figure 3.

We can observe that the attention matrix has an upper diagonal structure. Attention is not fairly distributed, for each word translated, one french word gathers almost all the attention. One explanation of such a result is that since the sentences are very simple, one only needs to

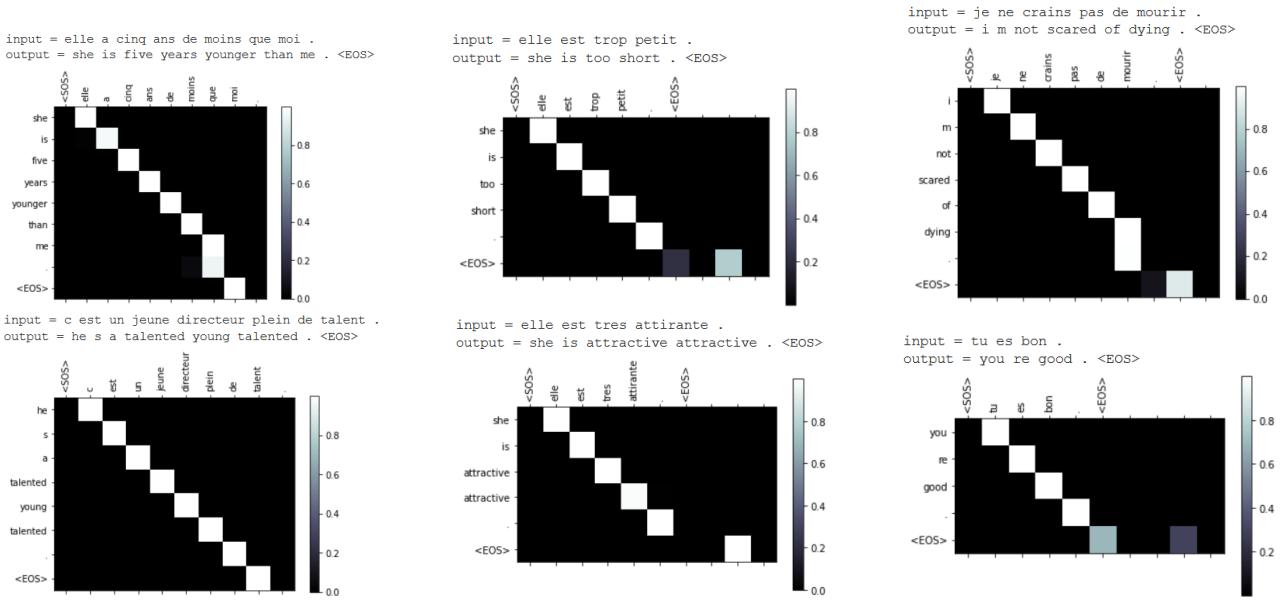

Figure 3: Seq2seq model with attention mechanism.

focus on the context of one word to translate the whole sentence.

Regarding the EOS token, it always seems to use the last token of the French sentence to predict it, even if it is an empty one. We can conclude that it understands that, when the French sentence is finished, it must end the translation with the EOS token. Also, we observe that the encoder-decoder never pays attention to this token in the French sentence (excepted in the case where it is in the last position, but it is not because it is an EOS token). However, it often uses the "." token to perform the translation. We can conclude that the period brings sufficient information about the ending of the French sentence, and it would be redundant to pay attention to the EOS token.

Finally, we observe that removing the EOS token in the French sentence leads to translations of similar quality. However, it outputs many periods at the end of each sentence. This is because, in the train function, we specified a rule that breaks the algorithm when it meets an EOS token. Now such a condition is never met, so the encoder-decoder always outputs a 10-character sentence and uses periods to fill the empty slots. The results without the EOS token are displayed in figure 4.

7.2 Question 13

When using the attention mechanism, we give the decoder all the hidden states used by the encoder, instead of simply the last one. This allows the decoder to have access not only to the general context of the sentence but also to the specific context of each word.

For such a task as translation, it is necessary to have access to such information to perform well. Therefore, the attention mechanism is very important in this model and for this task.

We show below some translations performed without the attention mechanism. As we can see,

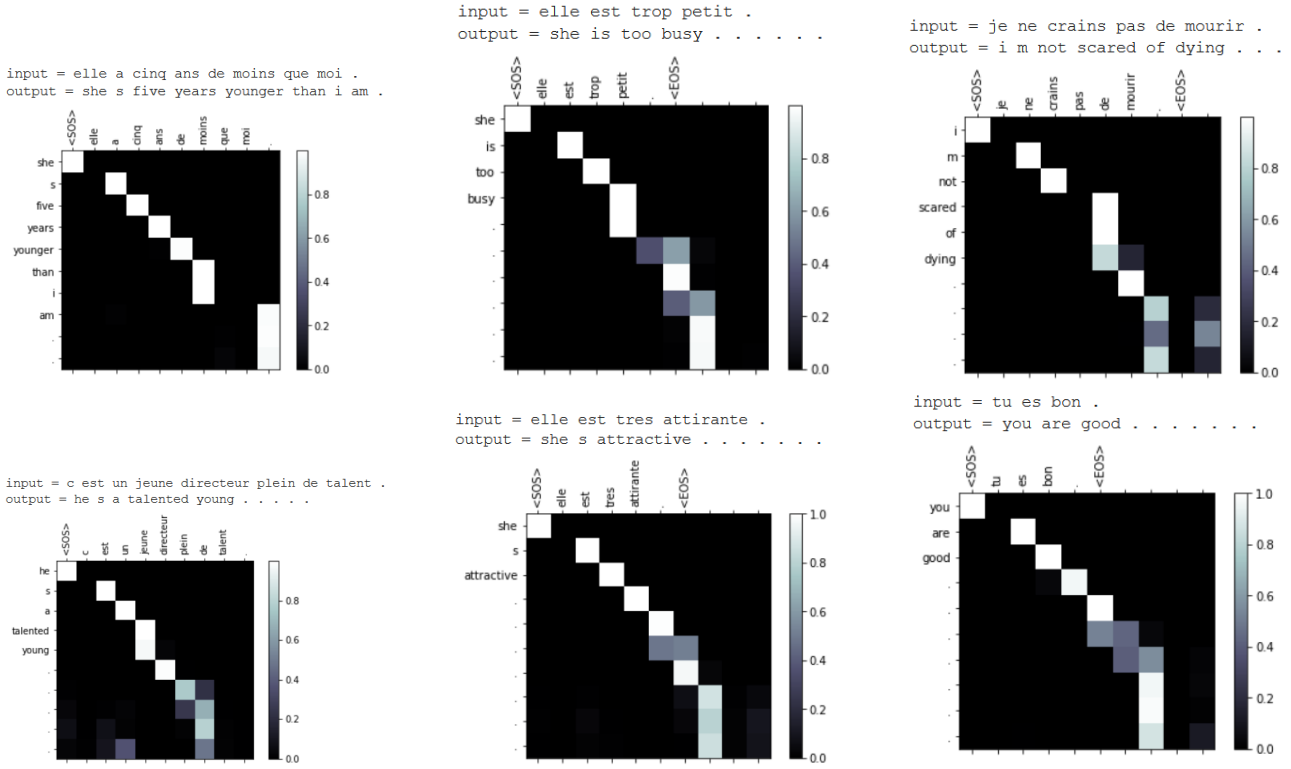

Figure 4: Seq2seq model with the attention mechanism, trained with sentences from which we removed the EOS tokens.

the translation is rather poor, as one hidden state is not enough to reconstruct a clear English sentence:

French: Elle a cinq ans de moins que moi.
English: people he he be rain alone for hair my you

French: Elle a cinq ans de moins que moi.
English: I horse very very large you family success in all

7.3 Question 14

When the decoder performs a translation, it uses the previous word to translate the next one and makes a sentence that has real meaning. However, at the beginning of the training, the model is poor, and it often mistranslates some words. This indeed leads to error propagation to the next word to be translated, and therefore, this confuses the training of the decoder. Therefore, to achieve good performance, we need a very long training time. To be able to train on translating the second word, it has to be almost perfect on translating the first one. Also, while learning to translate the first one, it doesn't learn anything relevant on how to translate the rest of the sentence.

The idea of teacher forcing is to give the real translated word to the algorithm to carry on with translating, however accurately the first word was translated. In this case, the error does not

propagate. However, there is one drawback to such a technique, which is that it can lead to an unstable model: it will learn to rely on a previous word perfectly translated whereas, in the real tests, this is not the case. One can therefore expect a model which may be less robust.

8 Transformers

8.1 Question 15

A transformer that uses the self-attention mechanism can have two different organizations of its sequence of layers. The normalization occurs before or after the self-attention operation. We are going to derive the impact of that different sequence of layers by back-propagating the error using the chain rule.

For the original structure, we have:

$$\begin{aligned}\frac{\partial e}{\partial x_l} &= \frac{\partial e}{\partial x_L} \times \frac{\partial x_L}{\partial x_l} \\ &= \frac{\partial e}{\partial x_L} \times \prod_{i=l}^{L-1} \frac{\partial x_{i+1}}{\partial x_i} \\ &= \frac{\partial e}{\partial x_L} \times \prod_{i=l}^{L-1} \left(1 + \frac{\partial \mathcal{A}}{\partial x_i}(x_i)\right) \frac{\partial \text{ LayerNorm}}{\partial x_i}(x_i + \mathcal{A}(x_i))\end{aligned}$$

For the alternative structure, we have:

$$\begin{aligned}\frac{\partial e}{\partial x_l} &= \frac{\partial e}{\partial x_L} \times \frac{\partial x_L}{\partial x_l} \\ &= \frac{\partial e}{\partial x_L} \times \prod_{i=l}^{L-1} \frac{\partial x_{i+1}}{\partial x_i} \\ &= \frac{\partial e}{\partial x_l} = \frac{\partial e}{\partial x_L} \times \prod_{i=l}^{L-1} \left(1 + \frac{\partial \text{ LayerNorm}}{\partial x_i}(x_i) \frac{\partial \mathcal{A}}{\partial x_i}(\text{LayerNorm}(x_i))\right)\end{aligned}$$

8.2 Question 16

Transformers with numerous layers suffer from vanishing gradient which happens when the gradient of a deep layer gets contracted to 0 when back-propagating. In that case, it prevents the weights from being updated, leading to slow training.

With the chain rule, we showed that for a deep model, using an original architecture, the resulting gradient will be the product of a lot of terms. Thus, if they are many close to 0, it will lead to an almost null gradient. However, if we use the alternating structure, each term of the product has $a + 1$, ensuring that the value remains close to 1, so that the gradient does not vanish.

We can conclude that the consequence of using Eq. 22 instead of Eq. 21 is that it helps to prevent the vanishing gradient problem from occurring during training.