Deep learning homework

Mahdi Kallel (MVA)

1. *To sample from our model we use Ancestral sampling, this consists of drawing* $h^\star$ *from our prior* $P_r(h)$ *, we pass this through the network to compute the posterior* $P_r(x|h) = f(h^\star, \theta)$ *then we draw samples from this posterior.*

2. *Monte carlo sampling : Montecarlo integration suffers from the curse of dimensionality. To effectively approximate the integral, we should be able to effectively cover the whole space covered by the hypercube of* Z_s*. To cover a unit cube with no more than* 0.01 *distance between points in a* $\mathbb{R}^2$ *space this requires* 10^2 *samples,* 10^3 *in* $\mathbb{R}^3$ *...*

Therefore the number of needed samples of Z *grows exponentionally with the dimension of the embedding space.*

3. *If we take* $\mu_p = 0, \sigma_p = 1,\ p \sim N(0,1)$ *and* $q \sim N(0, \sigma_q^2)$ *and we let* σ_q *tend to zero then*

q *is more of a dirac in* 0 *and* $KL(P||Q) = \lim_{\sigma_q = 0} \dfrac{{\sigma_q}^2 - 1 - log({\sigma_q}^2)}{2} = +\infty$

For $\mu_p = \mu_q = 0, \sigma_p = \sigma_q = 1,$ *we get* $KL(P||Q) = 0$

4. *Kullback liebler divergence being always positive we get that :*

$log(P(x_n)) \geq log(P(x_n)) - KL(...) \geq ELBO$

From question 2*, we saw that computing* $P(x_n)$ *directly can be computationally expensive, therefore optimizing for the same quantity is hard.*

One must note that for most Z*,* $P(X|z)$ *will be zero and will not contribute the our estimate,* $P(x) = \sum P(X|z)\, P(z)$.

Although ELBO as well contains an expectation estimation, unlike sampling from $P(Z)$ *directly, whe are sampling* z *from* $P(z|x)$ *, and thus from spaces that are more likely to generate* X*, thus ,avoiding much of the zeros in the computation. This let's us compute* $\mathbb{E}_{p(z|x)}[P(x|z)]$ *more efficiently, using less samples and in turn this allows us to use SGD.*

5. *If ELBO increases there are two options :*

* $P(x_n)$ *increases which is the quantity we want to maximize.*
* *The second term increases, which can be thought of as a penalty forcing* Q *to produce* z *that can reproduce the image* X*. It's like a reconstruction loss.*

6. *By maximizing* $E_{q_\phi(z|x)}[log\ P_\theta(x|z)]$ *we seek to maximize the probability of finding the image "X" by reconstructing from points "z" samples from the posterior incurred by the image itself. This means that given X we would like to be able to reconstruct the image from the latent space incurred by the Encoder. And therefore the name Reconstruction loss.*

By minimizing $D_{KL}\ (Q(z|xn)\ ||\ P(z))$ *we are forcing the posterior* $Q(z|x)$ *to be close the prior, so it's a reguluzarization term.*

7. *For a batch of size B images , the steps are as follows :*

1. *Feed the images to the encoder which will output* $\mu_\phi(x_n)$ *and the diagonale covariance matrice,* $\sigma_\phi(x_n)$ *for each image.*
2. *Compute* $L_n^{reg} = D_{KL}\ (N(\mu_{\varphi(xn)}, \Sigma_{\varphi(xn)})||N(0, I))$ *, using equation* (9), *where* $N(0, I)$ *is our prior.*
3. *For each image we sample K times from the posterior* $N(\mu_{\varphi(xn)}, \Sigma_{\varphi(xn)})$ *this can be done by sampling a point from* $N(0, I)$ *and the multiplying it by* $\Sigma_{\varphi(xn)}$ *and adding the mean* $\mu_{\varphi(xn)}$ *This scheme allows for back propagation as we used only differentiable operations.*
3. *Pass the latent variables through the Decoder in order to compute* $p_\theta(x_n \mid Z = z)$.
4. *For each image, compute the expectation :* $E_{q_\phi(z|x)}[log\ P_\theta(x_n|z)] = \frac{1}{K}\sum p_\theta(x_n \mid Z = z)$.
5. *Sum the contribution of each image :* $\frac{1}{B}\sum^{B} E_{q_\phi(z|x)}[log\ P_\theta(x_n|z)]$
6. *Compute loss* $= L_n^{reg} + L_n^{rec}$ *and then back propagate.*

8. *To compute the term* $E_{q_\phi(z|x)}[log\ P_\theta(x_n|z)]$ *we need to back propagate the error from a layer that samples z from* $Q(z|x)$*, with this term we can back propagate through the parameters of the decoder* θ *but not the encoder* ϕ *because sampling is not a differentiable operation.*

In order to solve this issue the "reparametrization trick" offers to move the sampling to an input layer. Using the fact that we can sample from $N(\mu, \sigma)$ *by first sampling* $\epsilon \sim N(0, 1)$ *then computing* $z = \mu + \sigma * \epsilon$*. Now sampling from* $E_{q_\phi(z|x)}[log\ P_\theta(x_n|z)]$ *is just a set of differentiable operations with respect to the fixed input* ϵ.

9.

10. *Code is provided in the "Linear VAE " section of the VAE" notebook.*

Our architecture consits of two blocks : An encoder and a decoder.

The encoder is a stack of two MLP, with the second having an output dimension of $z_{dim} * 2 = 20 * 2$. *Representing the stacked mean and diagonal variance of the gaussian distribution.*

The mean and variance are passed to a "Sampling" module, which samples a number $\epsilon \in [0, 1]$ *then returns the latent variable* $z = \mu + \sigma * \epsilon$.

This latent variable is then passed to the decoder which is two stacked MLP's with an output size $= 28 * 28$. *We pass this output through a sigmoid and we get our reconstructed image.*

We use pixel wise binary cross entropy to compute the "Reconstruction loss" between the generated and real image.

We compute the KL divergence between our generated gaussian distribution and $\mathcal{N}(0, 1)$

We back propagate through the sum of these loss terms.

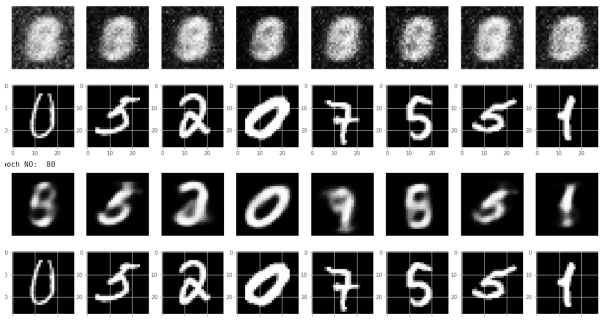

Figure 1: Linear VAE Output (epoch 0 / epoch 30)

11. *Our architecture is the same as UNet with lateral connexions between pairs of convolution blocks of the Encoder and the Decoder.*

Instead of having $Z_{dim} = 20$*, we fix the last ConvBlock of the encoder to have* $2 * 20$ *channels.*

Since information from the original image is passed between each pair of the Encoder / Decoder

The output of each convolution layer in the encoder is passed through the "Sampling module"

This way, the output of each Encoder is treated as the mean and std of a gaussian distribution.

And since each decoder is responsible for larger and larger scales in the reconstructed image, this allows for more variability across all scales. Which we see in figure 2.

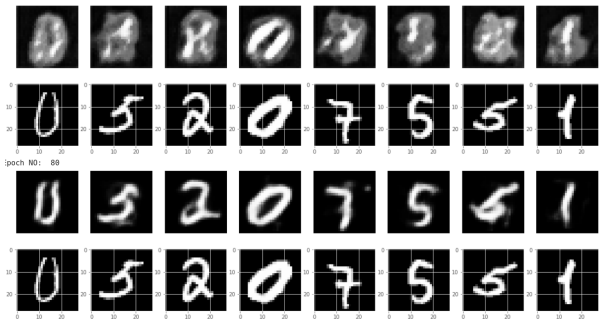

Figure 2: ConvVAE output (Epoch 0/ epoch 30)

12. *For the code please check the "Seq2seq" notebook.*

For the sake of this task, let's take into example translation the sentence "This river is full of fish " into "Cette riviere est pleine de poisson".
When translating the prenoun "Cette", we need to know the subject as to not output "Ce".
For this purpouse we, as humans will pay attention to the subject "Riviere".

We can observe this pattern of watching for the subject when translating the prenoun in figure 3. We can also notice that attention is usually diagonally distributed as in simple sentences we do not to keep into account a large context.

When translating the last word, our model always pays attention to the EOS token.

From figures 4, 5 we see that if we take away this token, the model has a tendency to repeat certain words, usually the last one.

This is because the EOS token means the end of the context, and without it the model does not know if it has finished translating the sentence.

Our interpretation is that, when this token is fed to the decoder it updates it hidden state in a radical manner, in order to keep only untranslated parts or tell it to end translation.

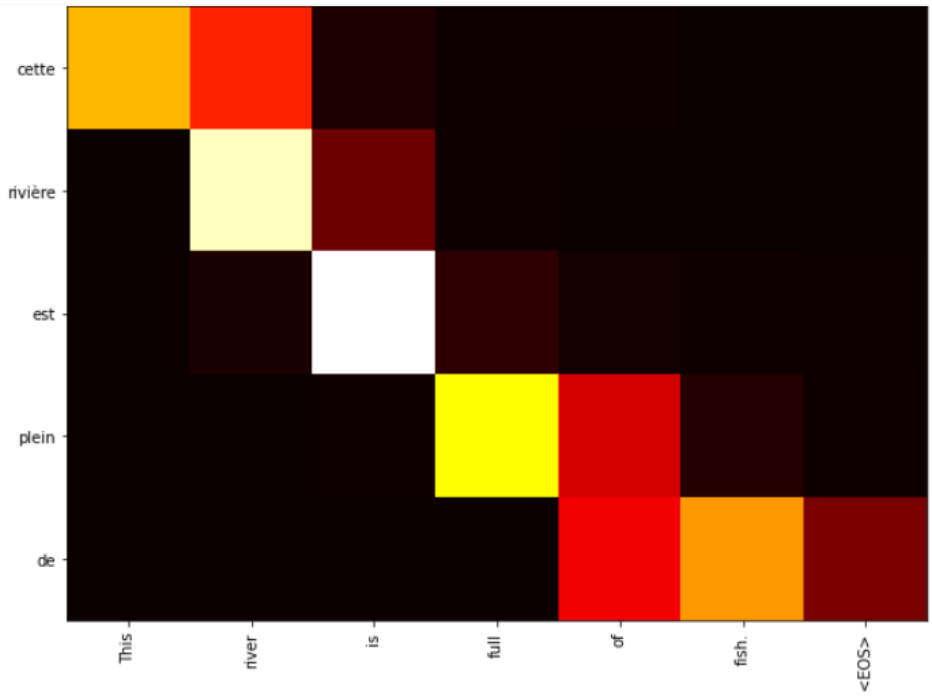

Figure 3: Translating (English to French) :

This river is full of fish ⟹ Cette riviere est plein de.

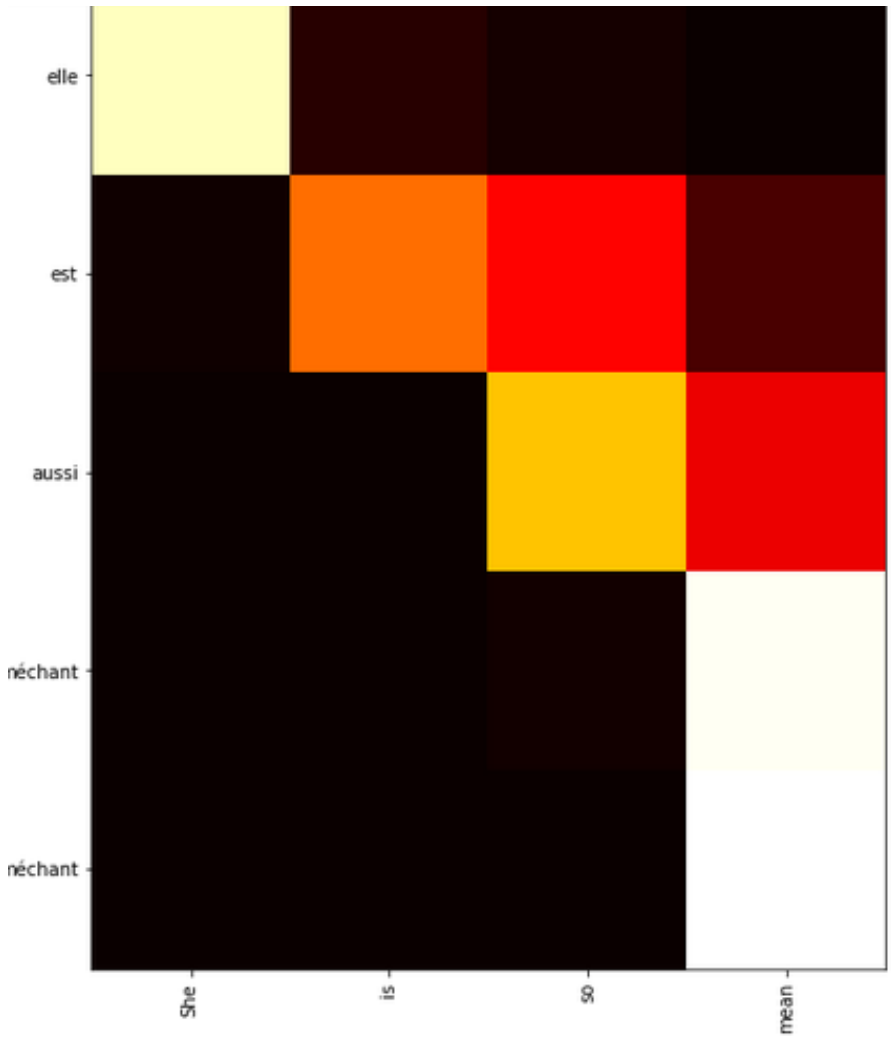

Figure 4: Without EOS there's always repetition.

13. *In a conventional Seq2seq model, the training sentence is fed fully to the Encoder, from which we keep only the last hidden state and discard the previous outputs and hidden states. The decoder then predicts an initial word, and this word is again sent back to the decoder in order to predict the next token.*

If we add an attention mechanism, then we keep track of the history of the hidden states of the encoder. Thus when decoding "La", our model can learn check specifically the hidden state corresponding to "girl" as if we check only the last hidden state, the information for "girl" can be overwritten by the update of "boy". Thus our model can learn noun and prenoun relationships.

If we remove this mechanisms, then the model will struggle in translating complex sentences with multiple subjects. As the more recent subjects in the sentence tend to overwrite the previous information.

From figure 5, we see that all models reach the same loss except for the Encoder Decoder without attention.

From figures 6 and 7, we see the radical improvement in the quality of the translations.

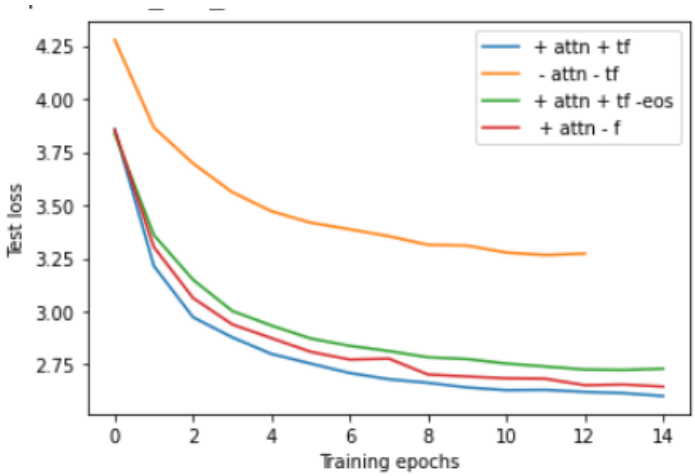

Figure 5: Evolution of training loss (+ is with , - without)

```
= = = = =
 I am a student. -> je suis étudiant étudiant étudiant <EOS>
= = = = =
 I have a red car. -> j ai une une voiture voiture . <EOS>
= = = = =
 I love playing video games. -> j aime aime vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo \
= = = = =
 This river is full of fish. -> cette poisson est une la la la . . <EOS>
= = = = =
 The fridge is full of food. -> le la est la la la la . <EOS>
= = = = =
 The cat fell asleep on the mat. -> le chat est endormi la la la . . . <EOS>
= = = = =
 my brother likes pizza. -> mon frère est mon mon mon . . <EOS>
= = = = =
 I did not mean to hurt you -> je n ai ai pas pas entendu que que que vous vous vous vous pas pas ? . <EOS>
= = = = =
```

Figure 6: Sample translation without attention (15 epochs)

```
= = = = =
 I am a student. -> je suis étudiant étudiant
= = = = =
 I have a red car. -> j ai une rouge rouge
= = = = =
 I love playing video games. -> j adore jouer jeux jeux jeux jeux jeux
= = = = =
 This river is full of fish. -> cette rivière est plein de
= = = = =
 The fridge is full of food. -> le frigo est plein de de
= = = = =
 The cat fell asleep on the mat. -> le chat est endormit endormi sur le
= = = = =
 my brother likes pizza. -> mon frère adore collectionner pizza
= = = = =
 I did not mean to hurt you -> je n ai pas que dire blesser blesser blesser vous vous
= = = = =
```

Figure 7: Sample translation with attention (15 epochs)

14. *When using RNN's for tasks like machine translation, usually the last output (word) of our model is fed to the decoder in order to make the next predictions. In the teacher forcing setup it's not the last output, but rather the real word (that we should have predicted), that gets fed into the decoder.*
Teacher forcing save's our model from accumulating error during translation.
If we do not use teacher forcing the Decoder hidden state will be updated with erronous words. Thus, teacher forcing is helpful especially in the first stages where the predictions are really far off.
Eventually this allows for faster and more stable training. (Which we can slightly see in fig 5). However, during inference, there's no ground truth available, this can lead to a discrepancy between the training task, and test task. As with teacher forcing, the task is approximately to predict the next word of a sentence given the "correct translation" of the previous words. Whereas the test objective is to predict the full translation of a sentence which a more difficult task.

We also notice that without teacher forcing, the model is unable to make sense of the EOS token,

we get similar translation results to removing this token.

```
= = = = =
 I am a student. -> je suis étudiant étudiant
= = = = =
 I have a red car. -> j ai une rouge rouge
= = = = =
 I love playing video games. -> j adore jouer jeux jeux jeux jeux jeux
= = = = =
 This river is full of fish. -> cette rivière est plein de
= = = = =
 The fridge is full of food. -> le frigo est plein de de
= = = = =
 The cat fell asleep on the mat. -> le chat est endormit endormi sur le
= = = = =
 my brother likes pizza. -> mon frère adore collectionner pizza
= = = = =
```

Figure 8: Translation results, with teacher forcing.

```
= = = = =
 I am a student. -> je suis étudiant étudiant étudiant <EOS>
= = = = =
 I have a red car. -> j ai une une voiture voiture . <EOS>
= = = = =
 I love playing video games. -> j aime aime vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo vidéo
= = = = =
 This river is full of fish. -> cette poisson est une la la la . . <EOS>
= = = = =
 The fridge is full of food. -> le la est la la la la . <EOS>
= = = = =
 The cat fell asleep on the mat. -> le chat est endormi la la la . . . <EOS>
= = = = =
 my brother likes pizza. -> mon frère est mon mon mon . . <EOS>
= = = = =
 I did not mean to hurt you -> je n ai ai pas pas entendu que que que vous vous vous vous pas pas ? . <EOS>
= = = = =
```

Figure 9: Translation results, without teacher forcing.

15.

$$\frac{\partial \epsilon}{\partial x_l} = \frac{\partial \epsilon}{\partial x_L} * \frac{\partial x_L}{\partial x_l}\ ,\ \textit{where L is the highest layer in our network.}$$

$$\textit{From the chain rule we know}\ :\ \frac{\partial x_L}{\partial x_l} = \prod_{k=l}^{L-1} \frac{\partial x_{k+1}}{\partial x_k}$$

$$\textit{For}\ x_{k+1} = \textit{LayerNorm}(x_k + A(x_k))$$

$$\frac{\partial x_{k+1}}{\partial x_k} = \frac{\partial LN(y_k)}{\partial y_k} * \left(1 + \frac{\partial A}{\partial x_k}\right)$$

$$\implies \frac{\partial \epsilon}{\partial x_l} = \frac{\partial \epsilon}{\partial x_L} \prod_{k=l}^{L-1} \left\{ \frac{\partial LN(y_k)}{\partial y_k} \left(1 + \frac{\partial A}{\partial x_k}\right) \right\} \text{***(1)}$$

$$\textit{For}\ x_{k+1} = x_k + \textit{LayerNorm}(A(x_k))$$

$$\textit{We can notice that}\ x_L = x_{L-1} + LN(A(x_{L-1})) = x_{L-2} + LN(A(x_{L-2})) + LN(A(x_{L-1}))$$

$$= x_l + \sum_{k=l}^{L-1} LN(A(x_k))$$

$$\implies \frac{\partial x_L}{\partial x_l} = 1 + \sum_{k=l}^{L-1} \frac{\partial LN(A(x_k))}{\partial x_k}$$

$$\implies \frac{\partial \epsilon}{\partial x_l} = \frac{\partial \epsilon}{\partial x_L} * \left(1 + \sum_{k=l}^{L-1} \frac{\partial LN(A(x_k))}{\partial x_k}\right) \text{***(2)}$$

16.

From equations $(1, 2)$ *we see that scheme* #1 *gradient with respect to layer* l *is a product of* $(L - l)$ *computations. Whereas the same gradient from scheme* #2 *consists of a sum of* $(L - l)$ *terms.*

$$If\ we\ consider\ \frac{\partial LN(A(x_k))}{\partial x_k} = X_k \sim \mathcal{N}(\mu, v),\ i.i.d\ variables$$

$$then\ we\ get: E\left[\frac{\partial \epsilon}{\partial x_l}\right]_{\#1} = E\left[\frac{\partial \epsilon}{\partial x_L}\right] * \mu^{L-l}\ which\ is\ eather\ too\ small\ or\ too\ big\ if\ \mu\ is\ far\ from\ 1.$$

$$Whereas\ E\left[\frac{\partial \epsilon}{\partial x_l}\right]_{\#2} = E\left[\frac{\partial \epsilon}{\partial x_L}\right] * n\mu\ which\ a\ relatively\ stable\ update\ for\ all\ values\ of\ \mu.$$

One other interpretation is that following scheme #1, *we are more prone to vanishing or exploding gradients.*

From the computatoin of $E\left[\frac{\partial \epsilon}{\partial x_l}\right]_{\#1}$ *we see that the gradient takes more extreme* (0 *or* ∞) *values the deeper we are in the network. Therefore, as the network gets deeper, training it will be harder with such configuration.*