Bastien LE CHENADEC
bastien.le-chenadec@eleves.enpc.fr



# 1 Question 1

The greedy decoding strategy consists in picking the word of the predictive distribution with the highest probability at each step. This strategy is clearly not optimal with regards to the objective (1) since it restricts the problem to a subset of the possible sequences (the ones that start with the most probable word).

$$\arg\max_{y} \sum_{j=1}^{m} \log p(y_j | y_{<j}, x) \tag{1}$$

Furthermore, sequences of words that are very frequent in the training set will be favored by this strategy, since they will have a higher probability of being picked at each step.

Since exploring the whole tree of possible sequences is not feasible, the beam search strategy suggests approximating the optimal sequence by keeping track of the $K$ most probable sequences at each step. This strategy is not guaranteed to find the optimal sequence, but it is much less likely to get stuck in a local optimum than the greedy strategy. It is at least $K$ times more computationally expensive than the greedy strategy, but seams to be a good trade-off between computational cost and performance.

# 2 Question 2

The translations often contain repeated words (*She is so mean* $\rightarrow$ *Elle est tellement méchant méchant*), especially the final point (. . . . . etc.) which is surprising since the model should take into account the words it already generated thanks to the decoder hidden state. It appears that this hidden state is not enough to keep track of the words that were already generated. A first approach might be to increase the size of the hidden state, but do note that this would be computationally expensive.

An *input-feeding* approach is suggested in [3] to solve this problem. The idea is that each alignment vector is computed independantly, which would explain why the model has a hard time keeping track of which words were already translated. The *input-feeding* approach consists in concatenating the previous alignment vector to the input of the decoder at each step. This way, the decoder has access to the previous alignment vector and can take it into account when computing the current alignment vector.

$$h'_t = [h_t; a_{t-1}] \tag{2}$$

This input feeding approach is the default approach in the RNN with attention model from the *OpenNMT* library [1]. One could say that this approach is still not ideal since it only takes into account the previous alignment vector. Since the attention mechanism is intuitively additive, I would suggest to instead use the sum of the previous alignments, which is not computationally more expensive.

$$h'_t = \left[ h_t; \sum_{i=0}^{t-1} a_i \right] \tag{3}$$

Another variation to this approach might be to add a factor $\gamma \in [0, 1]$ to 3 to "forget" words that were translated long ago.

$$\begin{cases} b_0 = a_0 \\ b_{t+1} = \gamma b_t + a_{t+1} \end{cases} \qquad h'_t = [h_t; b_{t-1}] \tag{4}$$

# 3 Question 3

We proceeded in two times to extract the attentions in the code :

1. Store the computed attention in an attribute of the *Seq2seqAtt* class.
2. Add a forward hook to retrieve this stored attention after each *forward* call to the *Seq2seqAtt* class.

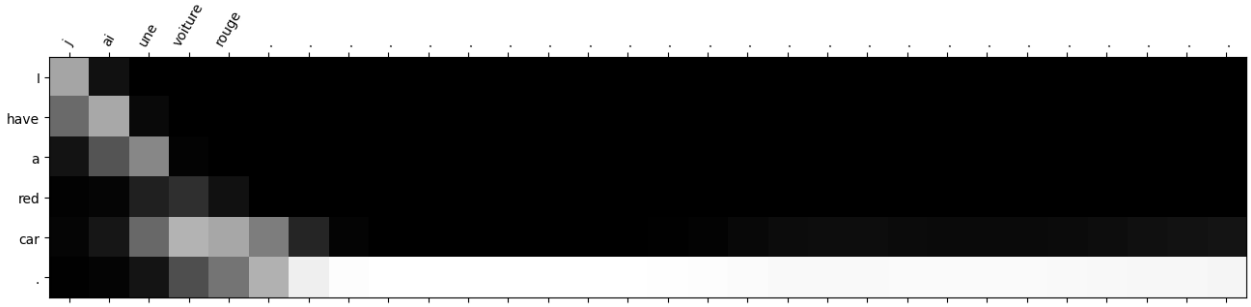


Figure 1: Alignments for the sentence ”I have a red car.”

On the figure 3 we can clearly see the need for input feeding to stop the model from translating the final point of the sentence multiple times.

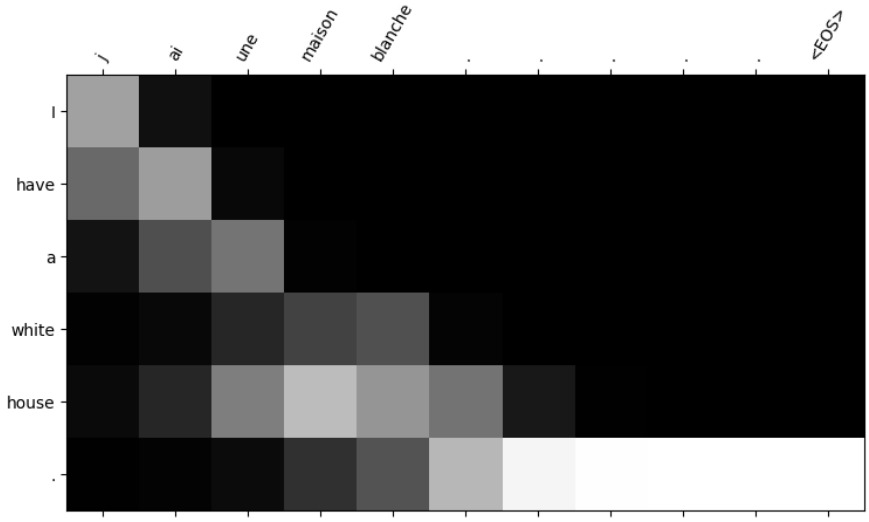


Figure 2: Alignments for the sentence ”I have a white house.”

The alignments on the figure 3 illustrate the adjective-noun inversion : the model first translates the noun *house* and then the adjective *white*. We observe that when translating to ”maison”, the attention is mainly focused on the word ”house” and when translating to ”blanche”, the attention is focused on the both words ”white house”.

# 4 Question 4

We get the following translations :

- ”I did not mean to hurt you” → ”je n ai pas voulu intention de blesser blesser blesser blesser blesser blesser . blesser . blesser . . . . . .”
- ”She is so mean” → ”elle est tellement méchant méchant .”

The model is able to infer different translations for the same word ”mean” in different contexts : ”intention” and ”méchant” – basically it knows how to deal with synonyms. This is nice because these different translations are purely based on the context of the word in the sentence; thus even our simplistic model is doing more than just translating word by word.

However the model is having a harder time to infer the meaning of words using the later part of a sentence. For example I was able to get the following translations :

- ”Right after” → ”Raison après”
- ”Right direction” → ”Raison direction”

In this case the model cannot infer what synonym to use from the context. This is despite all source hidden states being taken into account in the attention mechanism.

To solve these challenges :

- In [2], Delvin et al. suggest using a bidirectionnal transformer to encode the source sentence. In our case, we are using a GRU to encode the source sentence, which is not bidirectionnal. This might explain why the model is having a hard time to infer the meaning of words using the later part of a sentence. We could try to use a bidirectionnal GRU instead.
- In [4], Peters et al. suggest to encode the complex characteristics of words into the embedding layer. In this paper, each embedding is a function of the whole sentence, using bidirectionnal LSTM. This could easily be implemented in our model.

# References


[1] Opennmt website. `https://opennmt.net/OpenNMT/training/models/#default-decoder`. Accessed: 2023-10-13.

[2] Jacob Delvin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Pre-training of deep bidirectional transformers for language understanding. 2018.

[3] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. 2015.

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. 2018.