Natural Language Processing
Final Project

Matthijs Prinsen (s4003365)
Rob Sligter (s5564875)
Marinus van den Ende (s5460484)
Group 34

March 30, 2025

B(I)

To get out vocabulary size, we look inside the *tokenizer.json* file for the French and English tokenizers and read off the amount of tokens that are present at "model → vocab".

English: 3200 tokens

French: 3200 tokens

B(II)

To get the average number of target and source tokens, as well as the proportion of ⟨pad⟩ tokens in our training dataset, we do the following. First, we create a `DataLoader` for our training dataset that applies our `collate_fn` to pad the sequences. The `DataLoader` splits the data into batches to optimize training. This allows us to loop over each batch and count the number of source and target tokens. We then count how many of those tokens are padding tokens in both the source and target inputs.

Finally, we sum up the source and target token counts and divide by the number of batches to get the average number of tokens. We also sum the padding token counts and divide by the total number of tokens to get the proportion of padding. This gives us a batch-size-weighted average of the token counts and padding proportions.[1]

- **Average number of tokens in** `encoder_input_ids`: 433
- **Average number of tokens in** `decoder_input_ids`: 375
- **Average proportion of padding tokens in** `encoder_input_ids`: 33.55%
- **Average proportion of padding tokens in** `decoder_input_ids`: 29.76%

B(III)

The *model.pt* saves all necessary information like model parameters and optimizer states. Saving this data allows us to pause and resume training; Train different models and keep the best; Using the model without having to retrain it; Debugging the model and analysing it.

[1] The code for this analysis is implemented in the notebook below the questions section.

C

Sample 20

Input: je suis fier de vous tous .
Lit: I am proud of you all .
Gold: i m proud of you all .
Pred: i m proud of you .

Sample 30

Input: il est photographe professionnel .
Lit: he is photographer professional .
Gold: he's a professional photographer .
Pred: he's a a a .

Sample 31

Input: vous êtes incroyables .
Lit: you are incredible (plural) .
Gold: you're incredible .
Pred: you're amazing .

Sample 32

Input: je suis désolé .
Lit: I am sorry .
Gold: I'm sorry .
Pred: I'm sorry .

Sample 33

Input: elle lui sourit avec gêne .
Lit: she to-him smiles with embarrassment .
Gold: she smiled at him uneasily .
Pred: she is going to him .

Sample 34

Input: nous sommes frères .
Lit: we are brothers .
Gold: we're brothers .
Pred: we're going .

Sample 35

Input: vous n etes pas fatiguees si ?
Lit: you are not tired, yes?
Gold: you re not tired are you ?
Pred: you re not tired are you ?

BLEU Score: 43.4

The sentences we get from our model, while good, aren't perfect grammatical sentences. Some common mistakes are:

Sample 20: The model misses the word *"all"* at the end of the sentence.

Sample 30: The model incorrectly predicts a longer word.

Sample 31: The model chooses the wrong noun — but one that's still semantically valid. *"amazing"* instead of *"incredible"*

Sample 34 is simply incorrect. The model seems to have over-learned that *"nous sommes"* translates to *"we're going"*, no matter what comes after. This is probably because *"nous sommes"* often appears near movement verbs in the data.

Samples 32 and 35 were translated really well.

Sample 33 went wrong because the first two words in the input — *"elle lui"* — translate literally to *"she to-him"*, which the model read as *"she is going to him"*. It then ignored everything that came after.

A common trend in the model's mistakes is that it usually gets the first few words correct, but then inference performance significantly decreases after that.

D

The current beam search implementation is inefficient because it recomputes the entire decoder hidden states from scratch at every decoding step. Specifically, for each beam and time step, the model reprocesses all previously generated tokens, which leads to unnecessary repeated computation—especially for long sequences or larger beam widths.

This inefficiency comes from the fact that the decoder repeatedly recomputes the key and value projections for all previous tokens in the attention layers, even though they do not change over time. In contrast, efficient implementations, such as those in the Hugging Face Transformers library, like the KV cache strategies(Wolf et al., 2020), use a caching mechanism that stores the key and value pairs from previous steps and reuses them during generation.

To improve our implementation, we should introduce a caching mechanism using a new argument, typically called `past_key_values`, which stores the key and value tensors computed in previous decoding steps. This cache should be passed into the model during each forward pass and updated after each step to include the new values.

To support this, we need to make the following changes:

- `Beam search function`: Add a `past_key_values` argument to carry the cache through each decoding iteration. This cache is updated as new tokens are generated.
- `EncoderDecoderModel`: Modify the decoder's `forward` method to accept `past_key_values` as an input and return updated key/value pairs along with logits.
- `MultiHeadAttention layer`: Update it to check whether `past_key_values` are provided. If so, only compute new projections for the current token and concatenate them with the cached values.

This modification ensures that at each time step, the model only computes attention for the most recent token, reusing all previously computed states. The result is a significantly faster and more memory-efficient generation process.

This approach is particularly important when scaling to larger models or real-time translation tasks, where decoding speed becomes a bottleneck.

E

We implemented the *MarianMTModel* from the HuggingFace Transformers library.
And achieved the following BLUE score:

BLUE Score: 59.46

Below are samples taken from the predictions given by the *MarianMTModel*. The analysis for our own model can be found above in section C.

Sample 20

Input: je suis fier de vous tous .
Lit: I am proud of you all .
Gold: i m proud of you all .
Pred: I'm proud of all of you.

Sample 23

Input: vous etes plus grands que moi .
Lit: you are taller than me .
Gold: you re taller than i am .
Pred: You're bigger than me.

Sample 25

Input: je ne fais pas partie de leur bande .
Lit: I do not belong to their gang .
Gold: i m not one of them .
Pred: I'm not part of their gang.

Sample 27

Input: vous vous approchez .
Lit: you are approaching .
Gold: you re getting closer .
Pred: You're approaching.

Sample 31

Input: vous etes incroyables .
Lit: you are incredible (plural) .
Gold: you re incredible .
Pred: You're amazing.

Sample 33

Input: elle lui sourit avec gene .
Lit: she smiles at him with embarrassment .
Gold: she smiled at him uneasily .
Pred: She smiles at him with gene.

Sample 37

Input: je casse mes nouvelles chaussures .
Lit: I break my new shoes in .
Gold: i m breaking in my new shoes .
Pred: I'm breaking my new shoes.

Sample 39

Input: vous conduisez comme un tare !
Lit: you drive like a crazy person !
Gold: you re driving like a maniac !
Pred: You're driving like a tare!

Sample 20: Here the model paraphrases and says "of all of you" instead of "of you all". Still semantically and grammatically correct.

Sample 23: Incorrect word choice - "bigger" instead of "taller". Seems to mix up similar adjectives.

Sample 25: Translated informal language too literally - "part of their gang" instead of "one of them".

Sample 27: Translated too literally, "vous vous approchez" is an idiomatic expression for saying you're getting closer. Hints that model does not completely imbed idiomatic expressions and instead translates them literally.

Sample 31: Incorrect adjective used - "amazing" instead of "incredible" - the words are synonyms and the sentence is still semantically correct.

Sample 33: Model does not seem to know the translation for "gene". The word, and its translation, is most likely under represented in the examples. The word does in appear in out vocabulary list.

Sample 37: Misses slight nuance between "breaking in [...] shoes", which means wearing them till they snugly fit your feet, and "breaking [...] shoes", which is as it implies, breaking your shoes.

Sample 39: Incorrect translation of "tare". Upon further inspection, the word "tare" does not appear in our vocabulary.

In conclusion, our model implementation is not entirely fluent in translation, while we would be able to understand the majority of translations it provides, there are still cases where the meaning is lost or the translation fails and outputs gibberish. Our model often generates grammatical sentences but they are also often semantically incorrect.

The MarianMTModel on the other hand is much more fluent, we believe it could be passed as a translator from about 5 years ago, before Google Translate was as good as it is now. While the model gets most translations correct, it often replaces words with synonyms and seldom achieves a proper translation of idiomatic expressions from French. This model most often produces grammatically correct sentences.

There is a distinct difference in inference time between our model and the MarianMTModel. Our model generates translations from the test set at 6.24 iterations per second while the MarianMTModel generates translations at a slower rate of 4.48 iterations per second. This discrepancy is due to a couple of factors that work both in favour and against our model. [2]

Firstly, our model only has 2 encoder and decoder attention heads while the MarianMTModel has 8 of them. While this drastically increases the inference accuracy of the MarianMTModel, it decreases its speed. Additionally, we set the max_sequence_length of the MarianMTModel to 50, instead of our model's 32. This also causes a reduction in inference speed of the MarianMTModel since the model can infer longer examples. Finally, the MarianMTModel has another implementation that increases its inference speed, and that is a caching mechanism in the Beam search algorithm that allows it to save previously computed key value attention weights. The pros of this strategy is discussed in section D.

Additional Resources

To complete the coding for this assignment we closely referenced a YouTube series (AI, 2020) that explains in-depth how the Encoder-Decoder transformer architecture works.

References

AI, H. (2020, December). Visual guide to transformer neural networks - (episode 1) position embeddings. https://www.youtube.com/watch?v=dichIcUZfOw&t=124s

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., ... Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[2] These it/s values are from running the models locally, but the comparisons wont differ when run on faster hardware.