# NLP2

## - SCIA 2023 -

Jules DORBEAU - Noé JENN-TREYER - Yacine ANANE - Adrien HOUPERT

# Theoretical questions (16 points)

# 1 (2 points) What is the purpose of subword tokenization used by transformer models ?

Hint: Part of the answer is in the first part of the course (lesson 2).

The purpose of the subword tokenization used by the transformer models is to ensure that all the words, even the infrequent words, can be easily used and understood by the transformer. To do so, it will interpolate between word-based and character-based tokenization. This means that the familiar words will get a slot in the vocabulary but have a chance to be split into pieces and individual characters to allow the transformer to adapt when encountering unknown words.

With the subword tokenization algorithm, the frequently used words should not be split into smaller subwords (as this will mostly be words such as "the", and "this" for example). However, rare words should be decomposed into subwords (for example, the word "Transformers" might be decomposed into "Transform" and "ers"). So after this, following our precedent example, the word "Transform" and the word "ers" will be considered unique words and should appear more often and the meaning of the word "Transformers" will be conserved as their composition.

## 1.1 What is the effect on the vocabulary size ?

As subword tokenization is based on dividing rare words into subwords and adding them to the vocabulary, we can deduce that the size of the vocabulary will decrease by using this method because, instead of adding all the declensions of each word, all the suffixes are added once and only the root is added for each new word.

## 1.2 How does it impact out-of-vocabulary words (words which are not in the training data, but appear in the test data, or production environment) ?

For the out-of-vocabulary words, as we will have new subwords, this may help to process them as the combination of some of the created subwords might result in one of the out-of-vocabulary words encountered. So this method will help to process easily some of them by adding new words to the vocabulary and letting the transformer combine them when processing the out-of-vocabulary words.

# 2 (2 points) When building an encoder-decoder model using an RNN, what is the purpose of adding attention ?

## 2.1 What problem are we trying to solve?

The problem that we are trying to solve by adding attention to an encoder-decoder model using an RNN is to "extend" his limitations. Indeed, an issue that can be encountered with this approach is that the model needs to compress all the information from the source sentence into a fixed-length vector and might have difficulties doing that. This will be even more difficult for our model if he tries to use longer sentences than those in the training corpus. Attention will also help the model to understand the correlation between words in sentences.

## 2.2 How does attention solve the problem ?

By adding attention to our model, it will be able to process longer sentences. It will now be able to align the sentences which mean identifying which parts of the input sentence are relevant to the word of the output. This will be used by the translation process which will make use of the relevant information parsed to select the appropriate output words.

By selecting the relevant words, the model will not have to encode the input sentence to a fixed context vector. Instead, it will encode it into a dynamic context vector that was filtered especially depending on the output time step.

# 3 (2 points) In a transformer model what is the multihead attention used for ?

## 3.1 What are we trying to achieve with self-attention ?

To remind us what self-attention is, in an Encoder and in a Decoder, the input and target sequences will respectively only pay attention to themselves, but in the Decoder part of an Encoder-Decoder model with attention, the target sequence will pay attention to the input sequence. The attention module takes in parameters a query, a key, and a value that is processed using a head (only one in a self-attention module) and will compute their scores to finally merge them and return the result.

By doing so, when passing the data in the self-attention module, this will add its own attention scores to each word's representation. As this is done in both the Encoder and Decoder, this will help the model by having its own score for the input and the target data. So the Encoder-Decoder attention is getting the input sentence from the Encoder stack as well as the representation of the scores that we mentioned before.

Finally, a self-attention module, by comparing every word in a sentence to every other word (including itself as we just said), modifying the word embeddings to include contextual relevance and taking n words without any contextual information, aims to avoid producing similarity in the n outputs that it will be returning.

### 3.2 Why do we use multiple heads instead of one ?

By using multiple heads instead of one in a Transformer, we will be able to repeat the attention module (that will be called attention heads) computation multiple times in parallel. All of those attention computations will be then combined all together to produce a final score. This is why this is called multi-head attention. This will allow us to give our Transformer more power to encode multiple relationships for each word and nuance.

## 4 (2 points) In a transformer model, what is the purpose of positional embedding ?

In a transformer model, positional embedding is used to determine dynamic, meaningful relationships between tokens, and gives the transformer a way of knowing the order / position of the inputs. It consists of mapping the positions of the sequence to a trainable vector consisting of multiple dimensions. Each dimension will generally correspond to a "cosin" type of function with a higher frequency as we advance into the dimensions when the data we have is text, which will allow the transformer to "pinpoint" more accurrately where it is on the word sequence thanks to the characteristics of cosin function value variations.

A good explaination of this process can be learned in the very clear YouTube video "Positional embeddings in transformers EXPLAINED : Demystifying positional encodings". Nowadays, positional embedding can train and learn, and there are different implementations. In absolute positional embedding, each input token we have will be linked to a trainable embedding vector. In relative positional embedding, we generate the distance between the tokens (the distance corresponds to a certain number of tokens).

### 4.1 What would be the problem if we didn't use it ?

Without positional embedding, context-driven translation would prove more difficult are we would hardly know the position we currently are in the sequence as well as a less detailed relashionships between tokens (words) that the positional embedding allow to "shift" along the dimensions based on their position in said sequence, to make more context-accurate.

## 5 (2 points) What are the are the purpose of benchmarks ?

Benchmarks are used to choose the best transformer models among many by having a testing and ranking summary of the possible model we want to use. In this way and ideally, we can start building our solution on a right foot by not wasting time on models not adapted to our problematic.

### 5.1 And are they reliable ? Why ?

Benchmarks are heavily dependant on the criteria and datasets used to benchmark different models. These criteria are arbitrarly chosen by the people who did these benchmarking. Usually they should be explained in great details but not always, and not knowing why exactly a model obtained a better score than an other could mislead in the choice of the right model for our context and dataset.

## 6 (4 points) What are the differences between BERT and GPT?

The big difference between the BERT and GPT models is in their uses. Indeed, BERT is an encoder-only model, which means that it involves only transformer encoder blocks and GPT, unlike BERT, is a decoder-only model and so involves only decoder blocks.

### 6.1 What kind of transformer-based model are they ?

Both of these models are transformer-based models using self-supervised learning. Supervised learning is usually conducted by humans as this needs labeled data to be fed to the model. But in this context, the models are labeling themself the data.

### 6.2 How are they pretrained ?

As in this case, we are talking about self-supervised learning, the pretraining process is a little bit unusual. Indeed, it might sound strange but the pretraining process is unsupervised. The labels will be automatically generated based on the data attributes and definitions of the pretraining tasks.

In this context, doing the pretraining with a very large amount of unlabeled data helps. Indeed, the model will learn the general syntax while seeing the data and the semantic patterns. Then it will be able to generate better labels and will be more efficient after the pretraining.

Those models can also be used with downstream adaptation by adding just one or two layers to them. This will help to avoid the downstream model having to retrain from scratch but also to have better results on small datasets and avoid overfitting.

### 6.3 How are they fine-tuned ?

The fine-tuning of those models is simply done by adding some specific task in the labeled data. This will allow all the parameters of the model to be finetuned while it is pretraining as it will have to learn them to label other unsupervised data.

## 7 (2 points) How are zero-shot and few-shots learning different from fine-tuning?

Fine-tuning a model consist on training a model that is already trained on a new dataset (usually smaller than the original one) to adapt it more cases.

Zero-shot learning and few-shot learning are made to learn classification with very few examples. It is generally implemented at the beginning of the machine learning process and followed by transfer learning and fine-tuning to adapt the model to a greater variety of data.

### 7.1 How do fine-tuning, zero-shot, and few-shot learning affect the model's weights ?

Fine-tuning only slightly changing the weights, to adapt the model to this new batch of inputs.

Zero-shot learning and few-shot learning determine the weights at the beginning of the model by greatly affecting the weights, also because there is few data and each example is very important to learn how to classify them.

# 8 (2 points) In a few paragraphs, explain how the triplet loss is used to train a bi-encoder model for semantic similarity ?

A bi-encoder is a model that will transform queries and corpus in a vectories form so we can, in the context of semantic similarity, find queries that are really similar whith the part of the corpus.

The triplet loss goal is to give for one entry two output and to said to the model that the input has to be like the first output and has to be really different than the second.

So in the case of a bi-encoder model for semantic similarity, it is easy to see that the triplet loss function will allow us to train the model for recognizing similar sentences while learning how to differenciate other sentence.

# 9 (2 points) What is the purpose of using an Approximate Nearest Neighbour method to speed up search ?

ANN method are an efficient way of improving the model speed by a factor of 2 times or more by reducing it's complexity and size (compressing it), usually with a graph-based approach, while aiming to preserve accuracy. The ANN method will group corpus together so it can be faster to retrieve them. The downside of it means that sometimes text that doesn't match the queries will be retrieve with ohter text.

## 9.1 What does it really reduce ?

ANN reduces the model features vectors by projecting them from high-dimensional space to low-dimensional space without sacrifacing too much accuracy.