NLP(DEEP) - Lab 08 - Theoretical Questions

Eliot LECLAIR, Alex POIRON, Tom THIL, Aurelien VISENTIN

1 What is the purpose of subword tokenization used by transformer models ?

We use subword tokenization to avoid out-of-vocabulary word problems. The word vocabulary cannot handle words that are not in the training data. A subword vocabulary can be used to express these words. By using subword tokens, we can encode words that weren't even in the training data. There are still issues with characters not present in the training data, but for the most part it's acceptable.

1.1 What is the effect on the vocabulary size ?

With this technique, the vocabulary size is considerably reduced and can be more manageable for the model.

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) ?

As said above, subword tokenization can express out-of-vocabulary words due to the fact that in this technique we choose to split the rare words into smaller meaningful subwords and not split the frequently used words.

2 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 ?

Attention mechanism is a solution to the bottleneck state that is the last state of the encoder part. It's a simple manner to let the decoder access all hidden states.

2.2 How does attention solve the problem ?

We use a vector of fix size corresponding to a weighted sum of hidden layers. These weights help the decoder focus on parts of the input text which are important to decode the current token.

3 In a transformer model what is the multihead attention used for ?

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

The encoder uses a self-attention layer to add information about the context of the encoded token, for example, information about surrounding tokens. This allows more complex relationships between tokens to be modeled and also enables faster retropropagation.

These layers can also be used by the decoder to model the relationship between the current token and previously decoded tokens. However, the next token hasn't been decoded yet, so we can't and shouldn't care. This behavior is implemented by masking the self-attention layer.

3.2 Why do we use muliple head instead of one ?

Instead of learning one type of relation across words, we can using multiple heads per self-attention layer, hoping each head will focus on a different type of relation. This is what we are calling multi head attention. Each head can generate its own subspace of representation.

4 In a transformer model, what is the purpose of positional embedding ?

When we use self-attention for example, we lose informations related to order. To deal with this problem, the transformer adds a vector representing the position of each token to the embedding. This is what we are calling "positional embedding". Positional embeddings need to satisfy two constraints :

- They must remain the same for a given position, without any dependence to the token they project or the size of the input.
- Their addition to the embedding must not modify it so much that the embedding information is lost or damaged.

The goal of the positional embedding is to give a hint of position to the later layers, not to alter the semantic of the words.

5 What are the are the purpose of benchmarks ?

Benchmarks are truly useful on various points :

- A point of reference/comparison accepted by the community.
- A way to monitor/measure progress in a domain.
- A proxy for all tasks in NLP.
- A way to know where the domain is.
- What to use.
- What are the risks coming with the state of the art.

5.1 And are they reliable ? Why ?

Benchmarks are not always reliable as potential problems can occur and be discovered. For example, the benchmark task may be restrictive or too general, or there may be potential biases or shortcuts in the data that are amplified by the models that use them, resulting in a lack of true mastery of the task. You get good performance. Benchmark tasks may be too easy or too difficult to truly reflect the performance of the models in your domain. Finally, data annotation can be subjective. A good benchmark should have good agreement between annotators.

6 What are the differences between BERT and GPT ?

Both BERT and GPT are Transformer models pretrained to perform several NLP tasks such as natural language inference, classification, ... Nevertheless, their respective architecture and pre-training methods are different.

6.1 What kind of transformer-based model are they ?

BERT has an architecture called **Encoder-only** while GPT has a **Decoder-only** one.

6.2 How are they pretrained ?

BERT was pre-trained on BookCorpus and Wikipedia. The dataset is unlabeled. Unsupervised training was performed through two tasks : Masked language model and Next Set Prediction. For the first one, It consists of recognizing the surrounding context and predicting each hidden word in the sentence. The second task attempts to predict whether two sets are consecutive.

GPT was pre-trained using BookCorpus and 1B Word Benchmark. Both are also unlabeled datasets composed of sentences. Therefore, unsupervised pre-training of GPT consisted of learning Next Word Prediction. This task use sentences from these datasets to predict the next word in the sentence and know the previous word.

6.3 How are they fine-tuned ?

The fine-tuning of BERT and GPT are quite similar. Their aim is to train a model on a labeled dataset. Depending on the label, additional linear output layers may need to be added in order for the model to train on specific tasks. Certain tasks may require minor modifications. Additionally, the dataset, similar to that used during pre-training, can be used for unsupervised training to improve the model's performance on the task.

7 How are zero-shot and few-shots learning different from fine-tuning ?

The difference is that the zero-shot and few-shot learning methods consist of providing a specific input to the model so that it performs a specific task using the context provided in the input.

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

Fine-tuning consists of training a pre-trained model with new data. This means updating all or part of its weights.

While since the task context is given to the model only by the input, zero-shots and few-shots only use the model and do not train it. This means that model weights are not updated when zero-shot or few-shot learning is used.

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

When we train a bi-encoder model for semantic similarity, we learn an embedding for sentences because we will transform them in tensors with which we calculate their distance with a specific function (cosine similarity for example). The aim is that this distance function relates two embedded sentences proportionally to their similarity.

Triplet loss formula :

$$\mathsf{L}(\mathfrak{a}, \mathfrak{p}, \mathfrak{n}) = \max\{\mathsf{cos}_{\mathsf{sim}}(\mathfrak{a}_i, \mathfrak{p}_i) - \mathsf{cos}_{\mathsf{sim}}(\mathfrak{a}_i, \mathfrak{n}_i) + \alpha, 0\}$$

In this formula, we have 3 points denotes as $\mathfrak{a}$ the reference point, $\mathfrak{p}$ and $\mathfrak{n}$, that correspond respectively to another point that is quite similar to the reference one and vice versa. The aim of this loss function is to minimize the distance with the $\mathfrak{p}$ point and maximise it with the $\mathfrak{n}$ point.

Moreover, we use a $\mathfrak{max}$ function around it to stop it when the value is under a certain threshold. This threshold here is α. Also, we use present the formula with the cosine similarity function but in fact we can use any distance function.

9 What is the purpose of using an Approximate Nearest Neighbour method to speed up search ?

9.1 What does it really reduce ?

In the context of semantic search it reduces the time needed to find embeddings with high similarity by partitioning the data in order to create an index. For example ANNOY creates a binary search tree to speed up the search of the highest similarity embeddings. However, it also reduces the precision, as sometimes embeddings with high similarity can be missed. Generally, the faster an ANN method is, the higher the probability of missing high similarity embeddings is.