Exercise project 5 – Transformer networks

Introduction

The point of this document is to study how transformer networks work and to compare it with other neural networks. For this exercise, I chose the Keras English-to-Spanish translation with a sequence-to-sequence Transformer example. The original dataset contains English and Spanish sentence pairs to teach and test the model.

Parts of transformer networks

Preparation

After importing the necessary libraries and reading the dataset, we prepend [start] and [end] tokens to the Spanish translation part. This explicitly tells the decoder when to start and stop the translation, otherwise the model wouldn't know when a translation is complete and would produce infinitely long sentences.

```
("We'll lose everything.", '[start] Lo perderemos todo. [end]')
('Do you really expect to win?', '[start] ¿Realmente piensas vencer? [end]')
('Now, what do you think?', '[start] ¿Y ahora qué piensas? [end]')
('Her hair is long.', '[start] Ella tiene el pelo largo. [end]')
```

1 Sentences after [start] and [end] tokens.

Then comes the train/test/validation split which is basically the same that we did during the lectures. In the Keras example, they use a 70/15/15 proportion.

Vectorization

Machines understand numbers instead of text, therefore the strings must be converted to sequences of integers. We create a vocabulary of words, and the integers will represent the index of the word in the vocabulary. This is why this process is called vectorization because the text will become a set of integers. Our tokens will be complete words in this case, but they can also be shorter and longer units of text. The **vocabulary size** defines how many distinct tokens the model can represent. If it is too small, there is a potential

loss of information, but too large can be expensive to train. The **batch size** tells how many training examples will be processed in one step. Too small batch size can cause noisy results but is sometimes good for generalization. In most cases, a larger batch_size can be more useful. The **sequence length** determines how far the model can see, it's the number of tokens in a sequence. The best if the model attends the entire sequence, but a large size can increase memory usage a lot.

Encoder and decoder inputs

The goal of the model is to predict the next word of the sentence using the English sentence and the Spanish target sentence that was composed so far. So, encoder inputs will be the English equivalents, and decoder inputs being the beginning of the Spanish sentence, while target will be what the model is trying to predict.

Transformer encoder

Here comes the fun part. The next parts were introduced in the paper "Attention is All You Need" and are designed for sequence-to-sequence tasks. This layer contains an important part: *MultiHeadAttention*. This ensures that the words in the input sentence attend every other word in the sentence. And because it is multi head, the model can capture multiple relationships at the same time. Then we put the results into a small neural network called feed-forward network so that our model will learn even more complex relationships. This step makes the learning non-linear. For stability, we also include *LayerNormalization*.

Positional embedding

The transformer does not contain any information about the sequence of words. Therefore, we must create the *PositionalEmbedding* class. First, it makes a ***token embedding*** where the indices will be transformed into vectors, and then the ***positional embedding*** where the tokens also get a position vector. These two embeddings will be added up and returned, so our model will be able to understand position and sequences.

Transformer decoder

The decoder's role is to generate one token at a time for the output sequence. It uses masked self-attention which can't see the future tokens. Its point is to avoid data leakage during the learning process. Then comes the Encoder-Decoder Cross-Attention. This is where the translation happens. The decoder attends to the source sentence and connects it with the so far created sentence. Then comes the same structure feed-forward neural network as the encoder and a normalization after each previous step. A causal attention mask creates a lower-triangular matrix, that ensures that the model can't see any of the future positions

1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

2 Example of a lower-triangular matrix.
1-s represent the tokens that the model can see, and 0 what it can't.

Assembling the whole model

Now we combine the previously made custom layers. First, we decide on some hyperparameters. The **embedding dimension** defines the size of the vector for each token. If it is too small, the model won't be able to capture complex relationships, but if it's too large, there is a risk of overfitting. The **latent dimension** determines the size of the hidden layer in the feed-forward networks of the encoder and decoder. And the **number of heads** is important for the attention: how many inputs the model can look at the same time. Then we assemble our custom model in Keras and fit the data. After that we can decode the new example sentences by adding [start] and [end] tokens at the beginning and end of the sentences.

Experimenting with the number of epochs

First, after changing the necessary parts in the notebook, I ran it with one epoch. Of course, the result was not optimal, although I was surprised how

far it got with only 1 epoch. I found quite many unknown [UNK] tokens as visible here:

```
------------------
I've only heard good things about him.
[start] solo he oído cosas de cosas [end]
------------------
------------------
Many rivers in Japan are polluted by waste water from factories.
[start] muchos [UNK] son [UNK] son [UNK] de agua [end]
------------------
------------------
She was bitten by a wild animal.
[start] ella estaba [UNK] por un [UNK] [end]
```

Then I tried increasing the vocabulary size to 45000 while keeping the number of epochs just to see if I got fewer unknown tokens. I actually got none, but instead of [UNK] tokens, the model predicted false words (e.g. *manzana* means apple instead of medicine) or missing words. (*con un buen* = with a good).

```
How long should I take this medicine?
[start] cuánto tiempo debería tomar esta manzana [end]
------------------
------------------
I really wanted to go.
[start] realmente quería ir [end]
------------------
------------------
How did you come up with such a good excuse?
[start] cómo te has venido con un buen [end]
```

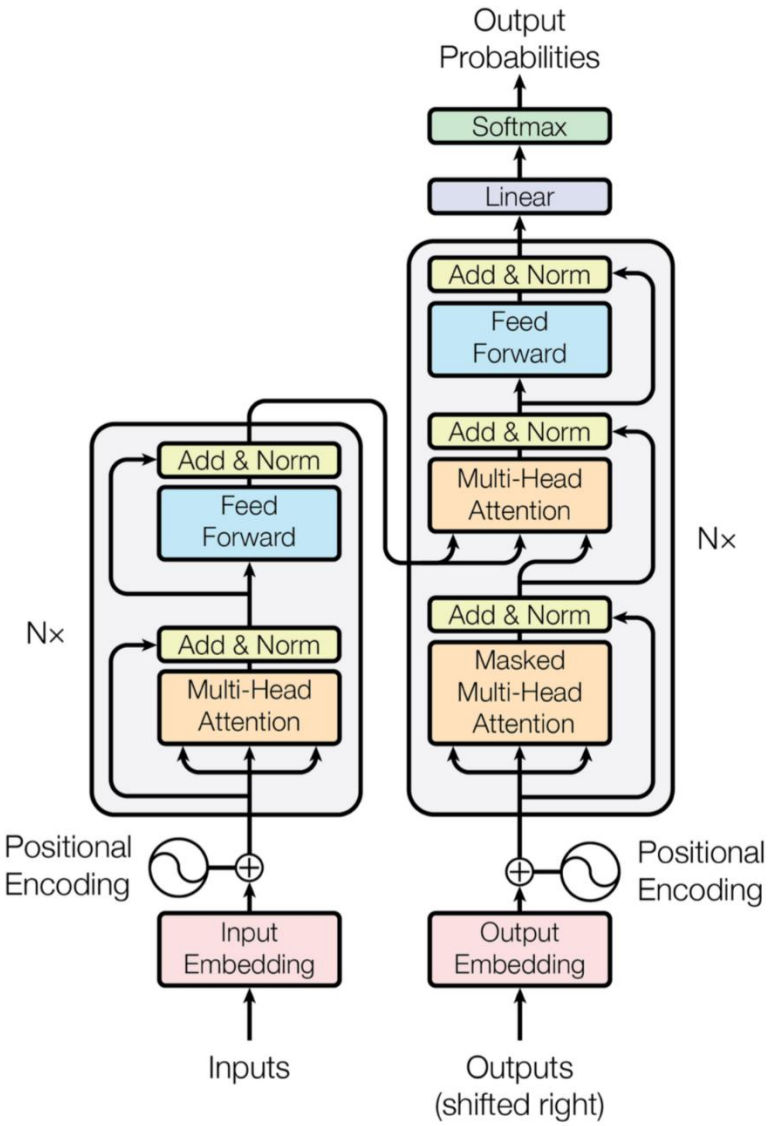
Then I reverted the vocab size to the original 15000 and increased the number of epochs to 20 which gave the best results without [UNK] tokens and hallucinating words.

Comparing with other neural networks

In the end of the day, transformer networks use similar components as other neural networks: Dense layers, nonlinear activation functions like ReLU. They both use backpropagation and update the weight with an optimizer. The main difference is attention: instead of recurrence or convolution, transformers use attention. Attention allows that the input is not only processed locally, but tokens can also focus on other tokens to understand the context of a text. That's how we get sentences that actually make sense.

Understanding the transformer network image

When I saw this picture for the first time, I thought transformer networks are too complex to comprehend, but after going through every step, now it makes more sense, and it gives a good overview of the relationship of the different steps. So, for me the best chronological order of understanding how transformer networks work is to first understand what happens in each step, then to see the summary of the network in the picture.



When creating this document, I used Gemini in Google Colab to understand the role of each part and how they work. Then I used ChatGPT as a validation tool to check whether my explanations and understanding were correct.