GALAGAIN Calvin
calvin.galagain@gmail.com

Lab session # 2
ALTEGRAD 2022

11/14/22

1 Question 1

Role of the square mask :

The mask is characterized here by the variable *src_mask* which is the additive mask for the src sequence. Having a mask solves several problems in the Transformers and thus plays a very important role in it. In the encoder and decoder, having a mask allows the attention outputs to be set to zero at the padding positions in the input sentences. Moreover, in the decoder, the mask makes it possible to hide the rest of the sentence to be translated in the previous layers of self-attention.

The principle of the mask is thus very intuitive, we mask the rest of the sentence little by little so that the sentence only appears word by word. We can thus see it with the mask of *generate_square_subsequent_mask(n)*.

```
model = nn.Transformer()
original_mask = model.generate_square_subsequent_mask(3)
print(original_mask)
```

```
[OUTPUT]
tensor([[0., -inf, -inf],
        [0., 0., -inf],
        [0., 0., 0.]])
```

Role of the positional encoding :

In addition to the mask, there is another very important process. Indeed, when we speak a language, the order of the words in a sentence makes it possible to give its meaning. The position of these is therefore crucial. Transformers do not use recursion or convolution and each word is treated independently. The positional encoding mechanism [2] will solve this problem by reinjecting the knowledge of the order.

In Transformers, positional encoding takes into account P (a function of position) k (the position of the word in the sentence), d (the dimension of the output embedding space) and n (any value).

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right) \tag{1}$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right) \tag{2}$$

These equations can be coded very easily in Python for an embedding of 30 for example.

```
# explanation of positional encoding
import matplotlib.pyplot as plt
import numpy as np

def positional_encoding(sentence_len, dimension_embedding, n):
    P = np.zeros((sentence_len, dimension_embedding))
    for k in range(sentence_len):
        for i in np.arange(int(dimension_embedding/2)):
            denominator = np.power(n, 2*i/dimension_embedding)
            P[k, 2*i] = np.sin(k/denominator)
            P[k, 2*i+1] = np.cos(k/denominator)
    return P

# for this sentence, we can calcultate the positional encoding
sentence = "Altegrad is the best NLP course."

# and we can plot the result
fig, ax = plt.subplots(figsize=(20, 10))
P = positional_encoding(sentence_len=len(sentence.split(" ")), dimension_embedding=30, n=10)
ax.matshow(P)
for (i, j), z in np.ndenumerate(P):
    ax.text(j, i, '{:0.1f}'.format(z), ha='center', va='center')
plt.show()
```

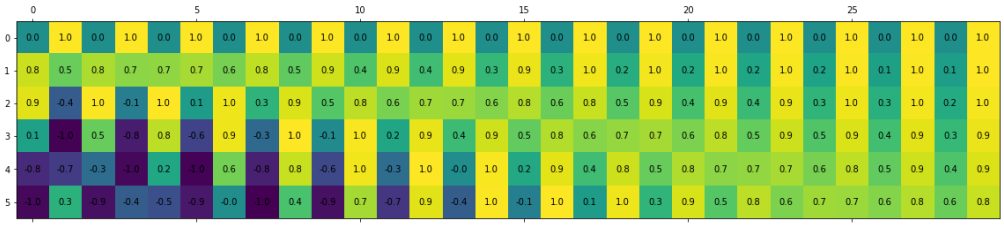

Figure 1: Positional encoding for each position.

2 Question 2

The principle of Transfer Learning is to recover the weights of a model that has already been trained. The model has learned a specific task but can be very good at it. For example, to classify ant species, you must first extract parts of the image. And, to make a classification of butterfly species, we don't want to retrain everything. The goal of Transfer Learning will be to keep the part of the model that has learned to recognize an insect, to recognize its legs, its antennae and what it deems interesting. To transfer knowledge, we will keep the first layers and change the last layers which are specialization layers. This is why we are changing the classification head because the old model was trained to classify another problem, but we want to keep the first stages of treatment.

Moreover, in Natural Language Processing, there are several areas that have different applications. For example, the main objective of language modeling will be to generate the continuation of a given sequence. This implies a fine understanding of linguistic rules, whether it is the representation of words or semantic representations. The purpose of a classifier will be to return a probability of belonging to a class. The goal is therefore not to generate text but to associate a class with an unstructured text.

3 Question 3

Before counting the total number of parameters, we will have to understand how a Transformer [2] is made, which can be represented as shown in Fig. 2.

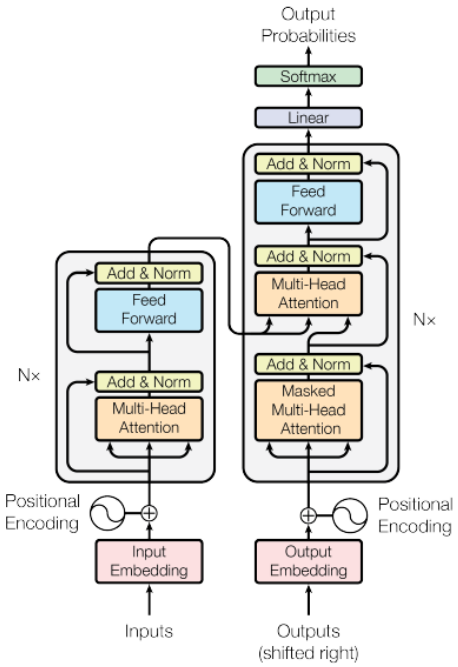

Figure 2: Positional encoding for each position.

We'll divise our problem into blocks considering that $d = n_{hid} = d_{model} = dim_{feedforward}$:

- An "MHA" block needs to do 4 linear calculations. 3 are simple projections and 1 is a concatenation. Considering the biases, we have here $4 \times (d^2 + d)$ parameters.

- An "Add Norm" block is a one-layer problem of vector norm calculations. Considering the biases, we have here $d + d$ parameters.
- A "Feed Forward" block is a network that has two linear layers matrix transformations. Considering the biases, we have here $2 \times (d^2 + d)$ parameters.
- An Embedding block has $d \times n_{tokens}$ parameters.

Our model thus has 3 "Multi-Head Attention" blocks, 5 "Add Norm" blocks and 2 "Feed Forward" blocks, which gives $3 \times 4 \times (d^2 + d) + 5 \times (d + d) + 2 \times 2 \times (d^2 + d) = 16 \times d^2 + 26 \times d$ parameters.

We will then have a difference depending on whether we are doing classification or language prediction. Indeed, the output layer will change and considering the biases, we have here $d \times size_{output} + size_{output}$ parameters.

In a first case, a binary classification is made where the output number is equal to 2. In the second case, the matrix of the linear output layer will depend on the size of the vocabulary. Note that classification is very close to language prediction. Here, our output size is only 2, but a classification with more classes is possible, 3, 4, 5,... or even going up to the size of the vocabulary. We thus find our other case.
By adding the block of Embeddings and the output parameters and, by considering N layers of Transformers, we obtain a number of parameters equal to:

- For a language modeling task : $d \times n_{tokens} + N \times \left(16 \times d^2 + 26 \times d\right) + (d + 1) \times n_{tokens}$
- For a classification task : $d \times n_{tokens} + N \times \left(16 \times d^2 + 26 \times d\right) + (d + 1) \times 2$

4 Question 4

The Figure that we obtain confirms many hypotheses. The principle of Transfer Learning is to recover a model trained on another database. When we seek to do Transfer Learning, it is because we seek to recover knowledge from a model that would take us a long time to calculate or that has better data than us. The first thing we observe in the Fig. 3 is that the result of the already trained model performs better. This implies that he already has the steps to understand the semantic links in the language. This model has potentially been trained for a very long time with a lot of data where our new model has to be content with the data it has and a few epochs to understand everything. Where the Transfer Learning model seeks to specialize, the from scratch model simply seeks to understand the problem.

Indeed, what our pre-trained model was able to learn is certainly far greater than what our model can and ever will be able to. With more data and time, he was able to understand the connections between words, semantic meanings and much more information. This observation is confirmed from the first epoch where our model from scratch is much worse than the trained model.

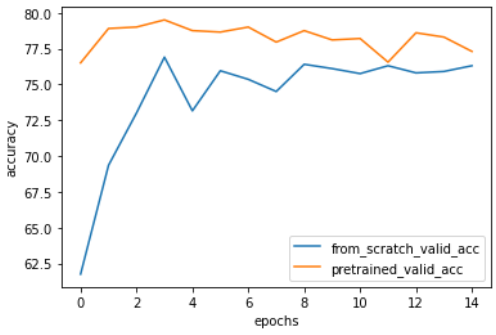

Figure 3: Evolution of accuracy for a training from sratch model and a pretained model.

Finally, the observation that we can raise and that our model of zero overfits very quickly and will not be able to improve. Even with all the time in the world, it won't catch up to the pretrained model. This confirms once again that the training data that the latter had was much more diversified because he was able to make better understandings of the languages.

5 Question 5

Before understanding the difference between our Transformer and *BERT* from the article [1] it is necessary to understand that our model is unidirectional. This means that our model is trained to predict each word based on the previous words in the sentence. When the context of the sentence comes after the words to be predicted, the results are logically bad.

BERT changes this mechanism because it is bidirectional. This means it trains by looking at the whole sentence, it predicts each word with the previous and next words in the sentence. It is even possible to predict based on the next sentence.

The problem that we quickly encounter in a multi-layer model will be that the word will see itself in the other layers. The article [1] tells us about a possible improvement that is the *Masked Language Model*. To add it to ourcode, it would be very simple because, we just need to add a token in the dictionary *token2ind* ("$< mask >$":4 for example). Then, associate this value with a random number of words following a specific percentage. These masks are trained as targets in the model and us a deep two-way system. The *BERT* model thus learns by predicting the "holes" in the input sentence by giving it a loss function that has retained the original labels.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.