Théo VIEL
theo.viel.enpc@gmail.com

Lab session # 7
ALTEGRAD 2019

01/22/20

1 Question 1

- Attention of y over x : $Q = x, K = V = y$
- Self-attention over x : $Q = K = V = x$

2 Question 2

- **Self-attention**:
  - Complexity per layer: We do n matrix multiplications in $O(n \times d)$, giving $O(n^2 \times d)$
  - Sequential operations: O(1) since the layer is not sequential
  - Maximum path length: Since we consider the whole sentence at once, it is in $O(1)$
- **Recurrent**:
  - Complexity per layer: we perform n matrix matrix multiplications in $O(d \times d)$, thus the complexity is $O(n \times d^2)$
  - Sequential operations: We need to go through each state of the sentence : $O(n)$
  - Maximum path length: Each token is treated at once hence, going from the first to the last one : $O(n)$
- **Convolutional**:
  - Complexity per layer: For each filter, we perform n matrix multiplications in $O(d \times k)$. Assuming we have d filters, complexity is $O(k \times n \times d^2)$
  - Sequential operations: No sequential operation : $O(1)$
  - Maximum path length: Convolutions consider k tokens at once, therefore the receptive field will be of size n after $\frac{n}{k}$ iterations : $O(\frac{n}{k})$

3 Question 3

The complexity of the attention layer is quadratic in the dimension of the layer.

Using several small heads can capture as much information as using only a big one, but the implementation is going to be much more efficient, on both time and memory aspects.

4 Question 4

For any fixed k, using $a_i = 10000^{\frac{2i}{d}}$.
We have :

$$PE_{x,2i} = \sin\left(\frac{x}{a_i}\right) \quad \text{and} \quad PE_{x,2i+1} = \cos\left(\frac{x}{a_i}\right)$$

First :

$$\begin{aligned} PE_{x+k,2i} &= \sin\left(\frac{x}{a_i} + \frac{k}{a_i}\right) && (1)\\ &= \cos\left(\frac{x}{a_i}\right)\sin\left(\frac{k}{a_i}\right) + \sin\left(\frac{x}{a_i}\right)\cos\left(\frac{k}{a_i}\right) && (2)\\ &= (PE_{x,2i}, PE_{x,2i+1}) \cdot \left(\cos\left(\frac{k}{a_i}\right), \sin\left(\frac{k}{a_i}\right)\right) && (3) \end{aligned}$$

Then :

$$PE_{x+k,2i+1} = \cos\left(\frac{x}{a_i} + \frac{k}{a_i}\right) \tag{4}$$

$$= \cos\left(\frac{x}{a_i}\right)\cos\left(\frac{k}{a_i}\right) - \sin\left(\frac{x}{a_i}\right)\sin\left(\frac{k}{a_i}\right) \tag{5}$$

$$= (PE_{x,2i}, PE_{x,2i+1}) \cdot \left(-\sin\left(\frac{k}{a_i}\right), \cos\left(\frac{k}{a_i}\right)\right) \tag{6}$$

Hence PE_{x+k} can be represented as a linear function of PE_x. The point of having this linear function is that it allows the model to learn information from relative positions, which is what makes it suitable.

5 Question 5

This modification causes the values of tokens of position above i to be 0 after the softmax layer. It prevents the attention mechanism to use those tokens for computing the output.

6 Question 6

The base Transformer has 65 million parameters, the big one 213. Base models were trained for 12 hours on 8 Nvidia P100 GPUs, it took 3.5 days for big models.

The GPU I have available is a 2080Ti, which is a bit faster than a P100 in my experiments. However, they have a bit less memory. Therefore, I should be able to train the normal architecture in about 4 days. Which is doable but a bit long...

7 Question 7

Training losses :
Epoch 1/10: train_loss=1.73
Epoch 2/10: train_loss=1.07
Epoch 3/10: train_loss=0.865
Epoch 4/10: train_loss=0.759
Epoch 5/10: train_loss=0.685
Epoch 6/10: train_loss=0.63
Epoch 7/10: train_loss=0.593
Epoch 8/10: train_loss=0.558
Epoch 9/10: train_loss=0.538
Epoch 10/10: train_loss=0.498

Translations :
I am a student. → ne je je je je pas un un un un un un un un un un un un je je est tout
I have a red car. → ai d d d d d d d d d d d d d rouge
I love playing video games. → j ' , , , , , , de les les cuisiner .
This river is full of fish. → ce ce ce ce ce ce ce ce ce ce ce ce que que que que que que que ce est est hommes
The fridge is full of food. → ne ne ne le le le , , que que que que que que que que que est est le tout
The cat fell asleep on the mat. → est est ce que que que que de de de est d . . .
my brother likes pizza. → frère , , , , , , , , , , , , affaires
I did not mean to hurt you → je n ai ai ai de de de de de de de de de le faire
She is so mean → , , , , , , , , , , , , , que que que que que que ce ce ce ce [OOV] .
Help me pick out a tie to go with this suit! → je il il il je je il il je je un un goût . . .
I can't help but smoking weed → n n que que que que , d d d , , , , , , , le en [OOV]
The kids were playing hide and seek → en en train en en en en en en train train train train train en le mariage

The cat fell asleep in front of the fireplace → chat chat est est est est est de de de de le le manière

8 Question 8

The input to the Bert model usually consists of a `[CLS]` token followed by the token ids of each sentence, obtained using the WordPiece tokenizer, delimited by a `[SEP]` token. Similarly, the sentence is the padded with a `[PAD]` token.

The sentence representation is a combination of 3 embeddings:

- **Token Embeddings** : Same as here, with the subtitlity that BERT uses the WordPiece token vocabulary.
- **Position Embeddings** : Same as here, although they are learned during training using an embedding layer.
- **Segment Embeddings** : For tasks with several sequences (ex: next sentence prediction) we want to embed the part we are considering as well.

We then sum the three embeddings to obtain the final representation. This representation is adapted to the Multi-Head self-Attention one, and is indeed efficient.

Position embeddings provide information about the position of the words which transformers do not know, token embeddings have proven encode words efficiently, and segment embeddings indeed are necessary for multiple sentences tasks.