Group Name: Athanasios

group's members:
1- Golnoosh Sharifi - 50011414
2- Mahdi Rahimianaraki - 50014390
3- Siarhei Sheludzko - 3092139
4- Aleksei Zhuravlev - 50104961
5- Marcel Melchers - 2897058

1. Transformers from scratch

Our solution link in google colab:

Solution 1 colab link

Also, the notebook file of the solution is in the zip file.

2. Relation Extraction with Transformers

Our solution link in google colab:

[Solution 2 colab link]

Also, the notebook file of the solution is in the zip file.

3)

3.1

An autoencoder is a type of artificial neural network used to learn data encodings. The autoencoder consists of two parts, an encoder, and a decoder. The encoder compresses the data from a higher-dimensional space to a lower-dimensional space (also called the latent space), while the decoder does the opposite i.e., convert the latent space back to higher-dimensional space.

Sequence to Sequence models is a special class of Recurrent Neural Network architectures that we typically use to solve complex Language problems like Machine Translation. Sequence to Sequence model architectures are similar to autoencoder architectures in that both of them use encoder and decoder parts.

Transformers are a type of artificial neural network architecture that is used to solve the problem of input sequences into output sequences and is also similar to autoencoders and Sequence to Sequence models, which use encoder-decoder architectures. The encoder extracts features from an input sequence, and the decoder uses the features to produce an output sequence.

3.2

First difference is that during batch normalization, each feature is normalized independently across the mini-batch, whereas each input in the batch is normalized independently across all features in layer normalization.
Second difference is that batch normalization is dependent on batch size, so it’s not effective for small batch sizes. In contrast, as layer normalization is independent of batch size, it can be applied to batches of any size.
Third difference is that during training and inference, batch normalization requires different processing, while when layer normalization is done along the length of the input to a specific layer, the same set of operations can be used for both training and inference.

3.3

Autoregressive models are pretrained on the classic language modeling task: guess the next token having read all the previous ones. Autoregressive models rely on the decoder part of the original transformer and use an attention mask so that at each position, the model can only look at the tokens before the attention heads. Although those models can be fine-tuned and achieve great results on many tasks, the most natural application is text generation.

An autoregressive model can therefore be seen as a model that utilizes its previous predictions for generating new ones. In doing so, it can continue infinitely, or - in the case of NLP models - until a stop signal is predicted.

The class of Transformers called GPT (GPT-2 and GPT-3) is autoregressive (Radford et al., 2018). GPT is heavily inspired by the decoder segment of the original Transformer.

1) The input is first embedded. This embedding is a matrix (position embedding matrix) and hence the actual input is a vector with multiple tokens (meaning that it can be used time and time again, i.e., have an autoregressive property).
2) 12 decoder segments with masked multi-head attention segments, feedforward segments, and layer normalization segments interpret the input values.
3) The output can be a text prediction; in that case, the task is to model language. However, it can also be used for other tasks, such as similarity detection and multiple choice answering.

By means of pretraining, the model learns to model language. It can subsequently be fine-tuned for the additional tasks mentioned above

3.4

In RNNs, the order of the input data is respected, since the input data are given to the network respectively. But in the transformer we are giving all our input words in a sentence at the same time to the transformer. So, what happens then is that it is really hard to know where a word belongs in a sentence that is why we use positional encoding to pass the information of location to the transformer.
In positional embedding, a unique vector that depends on the position is added to each input vector. One of the methods of producing these unique vectors is the sin-cos based method.

Here we want to explain two more methods. First one is Relative Position Encodings. They are a type of position embeddings for Transformer-based models that attempts to exploit pairwise, relative positional information. Relative positional information is supplied to the model on two levels: values and keys. This becomes apparent in the two modified self-attention equations shown below. First, relative positional information is supplied to the model as an additional component to the keys

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^\top}{\sqrt{d_z}}$$

The softmax operation remains unchanged from vanilla self-attention.

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$$

Lastly, relative positional information is supplied again as a sub-component of the values matrix.

$$z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V + a_{ij}^V)$$

In other words, instead of simply combining semantic embeddings with absolute positional ones, relative positional information is added to keys and values on the fly during attention calculation.

According to the site.

Second one is RNN based positional encoding. In order to endow input sequences with order context, recurrent layers are typically used before the Transformer or any other attention mechanism. These RNN+Attention architectures are usually trained jointly, which often results in the RNN doing much of the computation required for solving the problem and not just providing the positional embedding.

3.5

In the attention mechanism added to RNNs, the output hidden state of each decoder is checked with the output hidden states of all encoders separately in the attention module. In fact, in the attention module, these two hidden states are Dot Product together to calculate the similarity between them, and then after passing through the softmax, this value, which is called alpha, is multiplied by the value of the hidden state of each encoder. In fact, Encoder's hidden states are weighted. Then these values are added together and the result is concatenated to the hidden state of the decoder. In fact, attention made us pay attention not only to the last words but also to the beginning words.
The development of the Transformer architecture revealed that attention mechanisms were powerful in themselves and that sequential recurrent processing of data was not necessary to achieve the quality gains of RNNs with attention. Transformers use an attention mechanism without an RNN, processing all tokens simultaneously and calculating attention weights between them in successive layers. Since the attention mechanism only uses information about other tokens from lower layers, it can be computed for all tokens in parallel, which leads to improved training speed.
The attention mechanism combined in the RNN allows it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning of higher quality. The Attention mechanism enables this model to have extremely long-term memory.
by multi-head attention in Transformers, input data can be processed in parallel and at the same time, and this increases the speed and performance of Transformers. In fact, with multi-head attention, we can use the parallelization power of hardware (GPU) to increase the speed and performance of the training stage and use Transformers.

3.6

The BERT model is actually a group of transformer model encoders that have been trained. Both BERT models have a large number of encoder layers.
The BEST BASE model contains 12 encoder layers (called Transformer Blocks in the original article) and the larger model, which is the BEST LARGE model, contains 24 encoder layers. The basic model has a total of 110 million parameters and the large model has 345 million parameters.
These two methods have been used to train BERT:
1- Masked Language Model
In this method, the model must guess the deleted words. The training of this network is bidirectional and is sensitive to previous and next words.
2- Next Sentence Prediction
The task of the model this time is to learn whether these two sentences are two consecutive sentences or not.
Methods of using BERT model

BERT can be used in two ways:
1- Fine-tuning
2- Feature Extraction

XLNET is a generalized autoregressive model where the next token depends on all previous tokens. As a generalized model, XLNET captures bi-directional context through a mechanism known as permutation language modeling. By incorporating auto-regressive models and bi-directional context modeling, it overcomes the disadvantages of BERT.
The difference between BERT and Transformer is that Transformers are composed of encoders and decoders, while BERTs are only encoders.
BERT is an Autoencoding based model, while XLNet is an Auto-Regressive. There is a difference between the Masked language modeling task, in which tokens for randomly masked languages must be predicted by the model.