0.1 Transformers

0.1.1 Introduction

The Transformer is a deep learning architecture introduced in Vaswani et al. 2017. Originally devised as a sequence-to-sequence model that uses attention to learn the relationships between the input and output sequences. Transformers have been revolutionary in the field of natural language processing (NLP) leading to the development of models like BERT Devlin et al. 2019 and GPT Brown et al. 2020. They are also starting to gain traction in fields like computer vision Dosovitskiy et al. 2021 reaching becoming the new state of the art. The attractiveness of transformers is their general purpose nature and ability to learn relationships with an infinite context window whilst being massively parallelizable. A brief overview of the transformer model is given below.

0.1.2 Embedding

Data points it into a vector representation called an *embedding* or *token* ensuring postional information is added to these tokens via a positional encoding. The embedding is a simple linear transformation of the input data $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ where N is the number of data points and D is the dimensionality of the data.

0.1.3 (Self-)Attention

The attention mechanism is a way to learn the relationships between the input and output sequences. 'Normal' attention infers what the most important word/phrase in an input sentence is which is not very powerful. This is where the idea of *self-attention* comes in. Self-attention is a way to learn the relationships between the data in the input sequence itself (Note when we say attention from now on, we mean self-attention). Consider a language modelling task, the sentence `The quick brown fox jumps over the lazy dog` has strong attention between the data like `fox` and `jumps` representing an action, `brown` and `fox` representing the color of the fox and so on, then there are very weak attentions between data `quick` and `dog` representing the lack of relationship between the two data. Using self-attention we can learn these relationships between the data in the input sequence, this gives a powerful mechanism to learn the relationships between the input and output sequences and thus allows the model to learn the translation of the input sequence.

In the transformer models we will use the embeddings $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ as the input to generate a query $\boldsymbol{Q} \in \mathbb{R}^{N \times d_k}$, a key $\boldsymbol{K} \in \mathbb{R}^{N \times d_k}$ and a value $\boldsymbol{V} \in \mathbb{R}^{N \times d_v}$ matrices via a simple linear transformation matrix $\boldsymbol{W_q} \in \mathbb{R}^{D \times d_k}$, $\boldsymbol{W_k} \in \mathbb{R}^{D \times d_k}$ and $\boldsymbol{W_v} \in \mathbb{R}^{D \times d_v}$ respectively.

Where each row of the matrices is the query, key and value vectors for each data point in the input sequence.

The query, key and value matrices are then used to compute the attention matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ as follows:

$$\boldsymbol{A} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right) \tag{0.1.1}$$

The intuition behind this is that we want to compute the similarity between the query and the key vectors as such we use the dot product between the query and key vectors. The softmax is used to normalize the attention matrix so that the rows sum to 1. The softmax is also scaled by $\sqrt{d_k}$ to prevent the softmax from saturating. The attention matrix is then used to compute the output matrix $\boldsymbol{H} \in \mathbb{R}^{N \times d_v}$ as $\boldsymbol{H} = \boldsymbol{A}\boldsymbol{V}$

The overall attention function for a layer is given by:

$$\boldsymbol{H} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{0.1.2}$$

0.1.4 Multi-Head Self-Attention

So far we have only computed the attention matrix $\boldsymbol{A}$ once so the model only learns one attention relationship, however, we can take advantage of using multiple attention 'heads' in parallel to learn many attention relationships, this scheme is called the *Multi-Head Attention* (MHSA).

Each attention head is computed using simple dot product attention of a transformed query, key and value matrix. They are transformed by a simple linear layer (a matrix) which is unique for each head of the MHSA, $\boldsymbol{W}_{\boldsymbol{q}}^{(i)} \in \mathbb{R}^{d_k \times d_k}$, $\boldsymbol{W}_{\boldsymbol{k}}^{(i)} \in \mathbb{R}^{d_k \times d_k}$ and $\boldsymbol{W}_{\boldsymbol{v}}^{(i)} \in \mathbb{R}^{d_v \times d_v}$ where $i \in [1, h]$ for a head count of h. Then the attention for the particular head is computed as follows:

$$\boldsymbol{H}^{(i)} = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_{\boldsymbol{q}}^{(i)}, \boldsymbol{K}\boldsymbol{W}_{\boldsymbol{k}}^{(i)}, \boldsymbol{V}\boldsymbol{W}_{\boldsymbol{v}}^{(i)}) \in \mathbb{R}^{N \times d_v} \tag{0.1.3}$$

Then the output of the MHSA is the concatenation of the outputs of each head $\boldsymbol{H}^{(i)}$ (stacked on to of each other) multiplied by a learnable matrix $\boldsymbol{W}_{\boldsymbol{O}} \in \mathbb{R}^{hd_v \times D}$ which transforms the concatenated output to the original dimensionality of the input sequence.

$$\text{MHSA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{concat}(\boldsymbol{H}^{(1)}; \boldsymbol{H}^{(2)}; \ldots; \boldsymbol{H}^{(h)})\boldsymbol{W}_{\boldsymbol{O}} = \begin{bmatrix} \boldsymbol{H}^{(1)} \\ \boldsymbol{H}^{(2)} \\ \vdots \\ \boldsymbol{H}^{(h)} \end{bmatrix} \boldsymbol{W}_{\boldsymbol{O}} \in \mathbb{R}^{N \times D}$$

0.1.5 Encoder

Now that we have covered the MHSA block, we can move on to the encoder of the transformer.

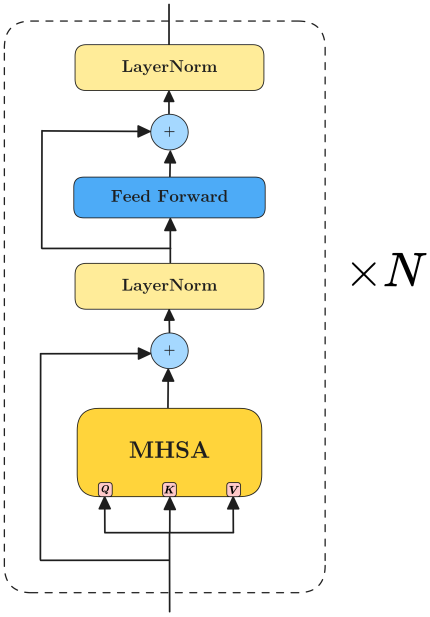

Figure 0.1.1: Transformer Encoder Vaswani et al. 2017

Figure 0.1.1 shows the encoder of the transformer model. The encoder is composed of a stack of N identical layers. Each layer is composed of two sub-layers, the MHSA and a simple feed-forward network. The output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$ where x is the input to the sub-layer. The final output of the encoder is the output of the last layer which shall be denoted as $\boldsymbol{Y} \in \mathbb{R}^{N \times D}$.

Key Points

The Transformer Encoder Layer takes an input set of embeddings $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ and outputs a set of embeddings $\boldsymbol{Y} \in \mathbb{R}^{N \times D}$ of the **same dimensionality** but with the **patterns of the input sequence learned**. It can be viewed as a function that takes a set and outputs a set of the same dimensionality.

Bibliography

Brown, Tom B. et al. (2020). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL].

Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].

Dosovitskiy, Alexey et al. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv: 2010.11929 [cs.CV].

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].