0.1 Transformers

0.1.1 Introduction

The Transformer is a deep learning architecture introduced in [Vas+17]. It is 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 and GPT. They are also starting tog gain tracking in fields like computer vision and reinforcement learning.

TODO

Add citations to popular transformer models

In this section we will purely focus on the encoder of the transformer model. The encoder is the part of the transformer that processes the input sequence and lairs patterns.

0.1.2 Embedding

Firstly we transform data points it into a vector representation called an *embedding*. Let us denote the embedding of the i-th data point in the input sequence as $\boldsymbol{e}^{(i)} \in \mathbb{R}^{1\times D}$, where D is the feature dimension. The transformer can process these embeddings in **parallel** so we need a way to encode the position of the data in the sequence. In [Vas+17] theyo this by adding a positional encoding $\boldsymbol{p}^{(i)} \in \mathbb{R}^{1\times D}$ to the embedding $\boldsymbol{e}^{(i)}$. The positional encoding is a vector that is unique to the position of the word in the sequence. The positional encoding that is used in [Vas+17] is given by:

$$\boldsymbol{p}_j^{(i)} = \begin{cases} \sin\left(\frac{i}{10000^{j/D}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{(j-1)/D}}\right) & \text{if } j \text{ is odd} \end{cases} \tag{0.1.1}$$

where j is the dimension of the positional encoding. The positional encoding is added to the embedding as follows:

$$\boldsymbol{x}^{(i)} = \boldsymbol{e}^{(i)} + \boldsymbol{p}^{(i)} \in \mathbb{R}^{1\times D} \tag{0.1.2}$$

These are all stacked together to form the input matrix $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, where $\boldsymbol{X}_{i:} = \boldsymbol{x}^{(i)}$ and N is the number of words in the input sequence. There are other ways to encode the position of the data in the sequence, later on we will concatenate the position and value of the data instead of adding the positional encoding to the embedding.

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.

$$\begin{aligned}\boldsymbol{Q} &= \boldsymbol{X}\boldsymbol{W_q} \in \mathbb{R}^{N\times d_k}\\ \boldsymbol{K} &= \boldsymbol{X}\boldsymbol{W_k} \in \mathbb{R}^{N\times d_k}\\ \boldsymbol{V} &= \boldsymbol{X}\boldsymbol{W_v} \in \mathbb{R}^{N\times d_v}\end{aligned}$$

Where each row of the matrices is the query, key and value vectors for each word in the input sequence. The query represents the word that we want to compare to the other words in the input sequence, to do this we compare it to the key vectors. The value vectors represent the word that we want to output.

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

$$\boldsymbol{A} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right) \tag{0.1.3}$$

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 follows:

$$\boldsymbol{H} = \boldsymbol{A}\boldsymbol{V} \tag{0.1.4}$$

The overall attention function for a layer is given by:

$$\boldsymbol{H} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{0.1.5}$$

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 different 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:

$$\boldsymbol{H}^{(i)} = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_{\boldsymbol{q}}^{(i)}, \boldsymbol{K}\boldsymbol{W}_{\boldsymbol{k}}^{(i)}, \boldsymbol{V}\boldsymbol{W}_{\boldsymbol{v}}^{(i)}) \in \mathbb{R}^{N \times d_v} \tag{0.1.6}$$

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 $\boldsymbol{W_O} \in \mathbb{R}^{hd_v \times D}$ which transforms the concatenated output to the original dimensionality of the input sequence.

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

The figure below summarizes the MHSA operation.

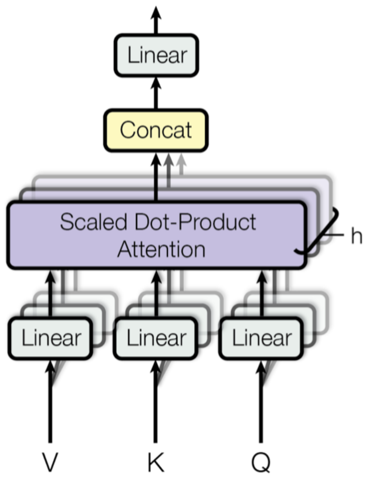

Figure 0.1.1: Multi-Head Self-Attention [Vas+17]

0.1.5 Encoder

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

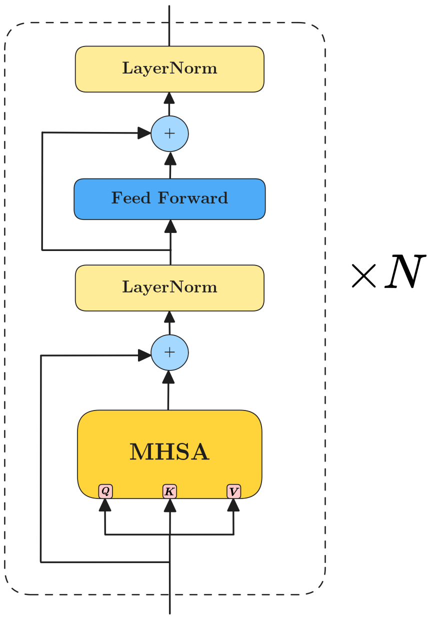

Figure 0.1.2: Transformer Encoder [Vas+17]

Figure 0.1.2 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. To dissect this, we first add the input from the sub-layer to the output of the sub-layer, this is called a *residual* connection. It allows the information from the input to bypass the sublayer and be passed to the next layer, thus preventing the network from losing information about the input. This is then passed through a layer normalization layer. The layer normalization layer normalizes the output (so each row) of the sub-layer to have a mean of 0 and a standard deviation of 1.

The final output of the encoder is the output of the last layer which shall be denoted as $\boldsymbol{Y} \in \mathbb{R}^{N \times D}$. 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.

Bibliography

[Vas+17] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: `1706.03762 [cs.CL]`.