1 Transformers

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.

In this report we will follow the notation that is used in [Vas+17] where embeddings are represented as rows $\in \mathbb{R}^{1\times D}$ and all matrix multiplications are right multiplications. Subscripts that are not bolded and lowercase are used to index vectors. Superscripts that are surrounded by parenthesis are used to index the position of a vector/matrix within a sequence, layer or head.

1.2 Embedding

A machine is not capable of understanding wordings, hence we need to transform it into a vector representation called an *embedding*. Let us denote the embedding of the i-th word 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 word in the sequence. We do 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:

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

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

$$\boldsymbol{x}^{(i)} = \boldsymbol{e}^{(i)} + \boldsymbol{p}^{(i)} \in \mathbb{R}^{1\times D} \tag{2}$$

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.

Alternative Positional Encoding

There are many ways to go about positional encoding. Another way is to use a learned positional encoding. This is done by adding a learnable vector $\boldsymbol{p}^{(i)} \in \mathbb{R}^{1\times D}$ to the embedding. Alternatively, to achieve translation equivariance, we can use *Relative Positional Encoding* [SUV18; Wu+21]. These positional encoding schemes will be useful when trying to build equivariance into the Transformer model. See [Kaz19] for a comparison of different positional encoding schemes.

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 words in the input sequence itself (Note when we say attention from now on, we mean self-attention). The example sentence `The quick brown fox jumps over the lazy dog` has strong attention between the words 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 words `quick` and `dog` representing the lack of relationship between the two words. Using self-attention we can learn these relationships between the words 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 used 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:

$$\boldsymbol{A} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right) \tag{3}$$

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:

$$\boldsymbol{H} = \boldsymbol{A}\boldsymbol{V} \tag{4}$$

Attention Mechanisms

There are many ways to compute the attention matrix $\boldsymbol{A}$. The one that is used in the original transformer paper is called *Scaled Dot-Product Attention*. Other attention mechanisms may be of interest, see [Wen18] for a comparison of different attention mechanisms.

The overall attention function for a layer is given by:

$$\boldsymbol{H} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{5}$$

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:

$$\boldsymbol{H}^{(i)} = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_{\boldsymbol{q}}^{(i)}, \boldsymbol{K}\boldsymbol{W}_{\boldsymbol{k}}^{(i)}, \boldsymbol{V}\boldsymbol{W}_{\boldsymbol{v}}^{(i)}) \in \mathbb{R}^{N \times d_v} \tag{6}$$

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

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

The figure below summarizes the MHSA operation.

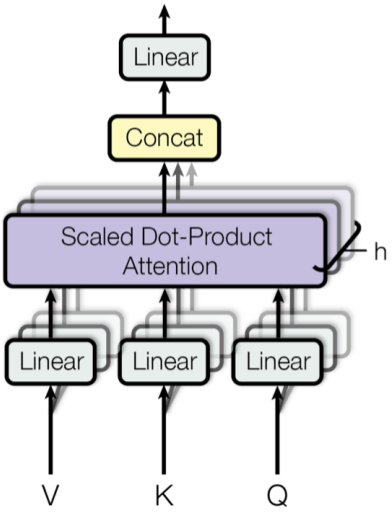

Figure 1.1: Multi-Head Self-Attention [Vas+17]

1.5 Encoder

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

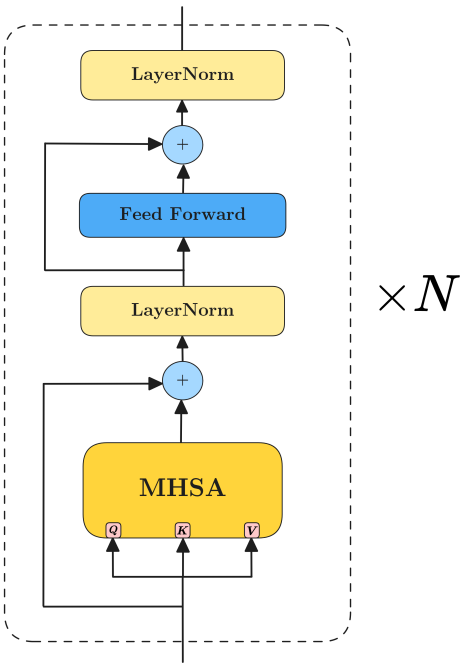

Figure 1.2: Transformer Encoder [Vas+17]

Figure 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. e.g consider a row of the output of the sub-layer $\boldsymbol{h} \in \mathbb{R}^{1 \times D}$, the layer normalization layer will normalize this row as follows:

$$\boldsymbol{h}_i^{(norm)} = \frac{\boldsymbol{h}_i - \mu}{\sigma}$$
$$\mu = \frac{1}{D}\sum_{i=1}^{D} h_i$$
$$\sigma = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(h_i - \mu)^2}$$

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

1.6 Decoder

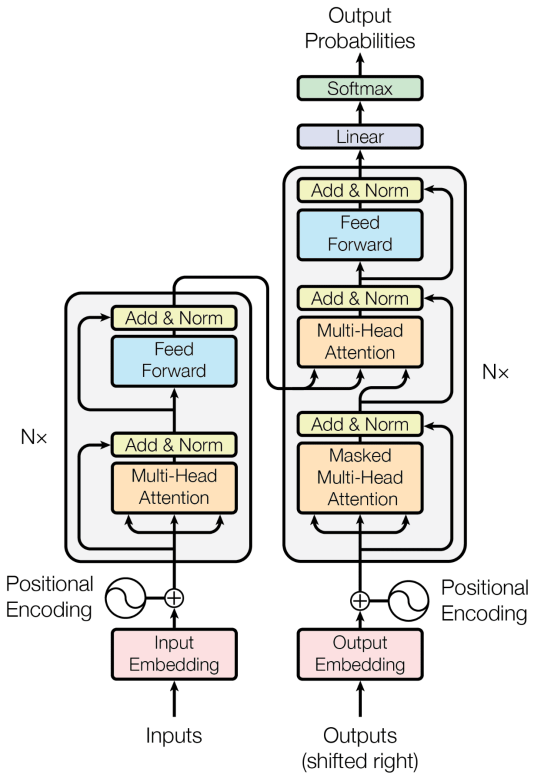

Figure 1.3: Transformer [Vas+17]

The transformer architecture is shown in Figure 1.3, the decoder is very similar to the encoder, with a few small differences. We now have an additional block called the *Masked Multi-Head Self-Attention* (M-MHSA). The M-MHSA is identical to the MHSA except that the attention matrix $\boldsymbol{A}$ is masked so that the decoder can only attend to previous positions in the sequence. This is done by setting the value of the scaled dot product $\boldsymbol{QK^T}/\sqrt{d_k}$ to $-\infty$ for all positions in the sequence that are greater than the current position. Then when the softmax is applied, these values will be 0 and thus the decoder will not attend to these positions. An example of this is shown below:

$$\begin{bmatrix} 0.2 & 0.3 & 0.5 & 0.1 \\ 0.1 & 0.2 & 0.7 & 0.0 \\ 0.3 & 0.4 & 0.2 & 0.1 \\ 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} + \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 \\ 0.4759 & 0.3092 & 0.2148 & 0 \\ 0.2824 & 0.3072 & 0.3333 & 0.0771 \end{bmatrix} \tag{7}$$

This ensures that the decoder can only attend to previous positions in the sequence. The M-MHSA is used in the first sub-layer of the decoder.

The second sub-layer of the decoder is the MHSA, this is identical to the MHSA in the encoder however the keys and values come from the **encoder output $\boldsymbol{Y}$** and the queries come from the **output of the M-MHSA sub-layer**. Intutively the encoder learns the attention of the input sequence and the decoder learns how to use query the encoder output to generate the output sequence.

$$\text{MHSA}_{dec} = \text{MHSA}(\boldsymbol{Q_{dec}}, \boldsymbol{K_{enc}}, \boldsymbol{V_{enc}})$$

The output of the MHSA_{dec} is then passed through a feed-forward network and layer normalization layer as per usual. This is repeated N times and the output of the final layer is the output of the decoder. This final output is then passed through a linear layer which transforms the decoder output $\in \mathbb{R}^{N \times D}$ to the output vocabulary size $\in \mathbb{R}^{N \times \mathcal{V}}$ where $\mathcal{V}$ is the vocabulary of the output language i.e the set of all possible words in the output language. The output of this linear layer is then passed through a softmax layer to generate the final predictive probability distribution over the output vocabulary. We select the word with the highest probability as the predicted word.

1.7 Training

The transformer is trained using teacher forcing. Teacher forcing is a technique used in sequence-to-sequence models where the model is trained to predict the next token in the sequence given the previous tokens. This is done by feeding the model the previous tokens and then the next token in the sequence. For example if we want to translate the sentence `I live in Paris` to French, we would feed the model the tokens `I` in the encoder and a start token `<start>` in the decoder and see if it can predict the translated token `Je`, we then feed correct token `Je` into the decoder and see if it can predict the next token `vis` and so on. This forces the model to learn the correct translation of the sentence as we use the correct target tokens to generate a loss function ($\mathcal{L}$) which the transformer decoder optimizes.

The table below shows the training procedure for the translation example.

Encoder Input	Decoder Input	Loss
`I`	`<start>`	$\mathcal{L}$(`pred1`, `Je`)
`live`	`Je`	$\mathcal{L}$(`pred2`, `vis`)
`in`	`vis`	...
...	...	...

1.8 Inference

After we have trained our model, we can use it for inference. The methodology is similar to the training, however, the decoder output receives the previously predicted token as input instead of the correct target token. This is shown in the table below.

Encoder Input	Decoder Input	Decoder Output
`I`	`<start>`	`pred1`
`live`	`pred1`	`pred2`
`in`	`pred2`	...
...	...	...

References

[Kaz19] Amirhossein Kazemnejad. "Transformer Architecture: The Positional Encoding". In: *kazemnejad.com* (2019). URL: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[SUV18] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. *Self-Attention with Relative Position Representations*. 2018. arXiv: 1803.02155 [cs.CL].

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

[Wen18] Lilian Weng. "Attention? Attention!" In: *lilianweng.github.io* (2018). URL: https://lilianweng.github.io/posts/2018-06-24-attention/.

[Wu+21] Kan Wu et al. *Rethinking and Improving Relative Position Encoding for Vision Transformer*. 2021. arXiv: 2107.14222 [cs.CV].