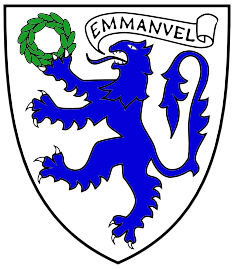
Unifying Transformers and Convolutional Neural Processes

Lakee Sivaraya (ls914)

June 15, 2023

Emmanuel College



Contents

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.

1.2 Embedding

A machine is not capable of understand 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 $\boldsymbol{x}^{(i)} \in \mathbb{R}^{1\times D}$, where D is the feature dimension. The transformer is able to 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 $\boldsymbol{x}^{(i)}$. 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:

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

These are all stacked together to form the input matrix $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, where $\boldsymbol{X}_{i:} = \boldsymbol{x}^{(i)}$ where 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 encodings schemes will be useful when trying to build in 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 a 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).

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 are the query, key and value vectors for each word 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:

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

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 $\boldsymbol{Y} \in \mathbb{R}^{N \times d_v}$ as follows:

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

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*. There are other attention mechanisms which may be of interest, see [Wen18] for a comparison of different attention mechanisms.

The overall attention function for a layer is given by:

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

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 individual attention head is computed using simple dot product attention however the learnable matrices are unique for each head of the MHSA, $\boldsymbol{W_i^Q} \in \mathbb{R}^{D \times d_k}$, $\boldsymbol{W_i^K} \in \mathbb{R}^{D \times d_k}$ and $\boldsymbol{W_i^V} \in \mathbb{R}^{D \times d_v}$ where $i \in [1, h]$ for a head count of h. Then the attention for the particular head is computed as follows:

$$\boldsymbol{H}_i = \text{Attention}(\boldsymbol{X}\boldsymbol{W_i^Q}, \boldsymbol{X}\boldsymbol{W_i^K}, \boldsymbol{X}\boldsymbol{W_i^V}) \in \mathbb{R}^{N \times d_v} \tag{6}$$

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

$$\text{MHSA} = \text{concat}(\boldsymbol{H}_1; \boldsymbol{H}_2; \ldots; \boldsymbol{H}_h)\boldsymbol{W^O} = \begin{bmatrix} \boldsymbol{H}_1 \\ \boldsymbol{H}_2 \\ \vdots \\ \boldsymbol{H}_h \end{bmatrix} \boldsymbol{W^O} \in \mathbb{R}^{N \times D}$$

2 Neural Processes

Neural Processes are an expansion of Gaussian Processes using Neural Networks introduced in [Gar+18]

References

[Gar+18] Marta Garnelo et al. *Neural Processes*. 2018. arXiv: 1807.01622 [cs.LG].

[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].