13.3 Natural language processing (NLP): Pretrained language models

13.3.1 Transformer

13.3.1.1 *Introduction*

Transformer [2] is one of most successful architecture in tackling difficult seq2seq NLP tasks like translation, question-answering, etc.

Traditionally, seq2seq tasks heavily uses RNN based encoder-decoder architecture, plus attention mechanism, to transform one sequence into another one. Transformer, on the other hand, does not rely on any recurrent structure and is able to process all tokens in a sequence at the same time.

On a high level, transformer also resembles the encoder-decoder architecture [Figure 13.3.1], where encoder converts input sequences into low-dimensional embeddings and decoders outputs output sequence probabilities based on input sequence embedding and previous output token. The sequential information, original stored in recurrent structure, is also stored in position encoding.

The encoder module on the left consists of blocks that are stacked on top of each other to obtain the embeddings. Multi-head attentions are used in each block to enable the extraction of contextual information. Similarly, the decoder module on the right also consists of blocks that are stacked on top of each other to obtain the embeddings. Two types of multi-head attentions are used in each block, one is to capture contextual information among output sequence and one is to capture the dynamic information between input and output sequence.

In the following, we will present detailed analysis on each component of transformer architecture.

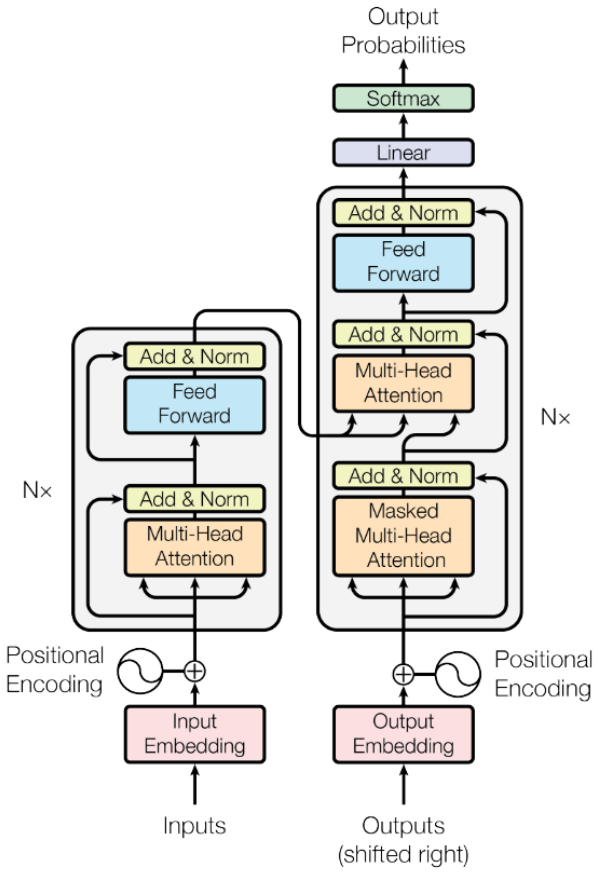

Figure 13.3.1: The transformer architecture [2].

13.3.1.2 *Input output conventions*

In the transformer architecture, we need to deal with different types of sequences

- Input sequence $s = (i_1, i_2, ..., i_p, ..., i_n), i_p \in \mathbb{N}$.
- Input position sequence $s^p = (1, 2, ..., n)$.
- Output sequence $o = (o_1, o_2, ..., o_p, ..., o_m), o_p \in \mathbb{N}$.
- Output position sequence $o^p = (1, 2, ..., m)$.
- Target sequence $t = (t_1, t_2, ..., t_p, ..., t_m), t_p \in \mathbb{N}$.

For example, consider a translational task with input and target sequence given by 'Ich möchte eine Flasche Wasser' and 'I want a bottle of water'. In the training, we can get a typical input sequence, target sequence, and output sequence in the following form

- $s = (Ich, mchte, eine, Flasche, Wasser, PAD, PAD)$
- $t = (I, want, a, bottle, of, water, EOS, PAD)$
- $o = (SOS, I, want, a, bottle, of, water, EOS)$

The output sequence is the right-shifted target sequence with a starting token. Because sequence data is fed into transformer via mini-batches, all input sequence in the same mini-batch will be pad to the maximum length of its batch.

13.3.1.3 *Multihead attention with marks*

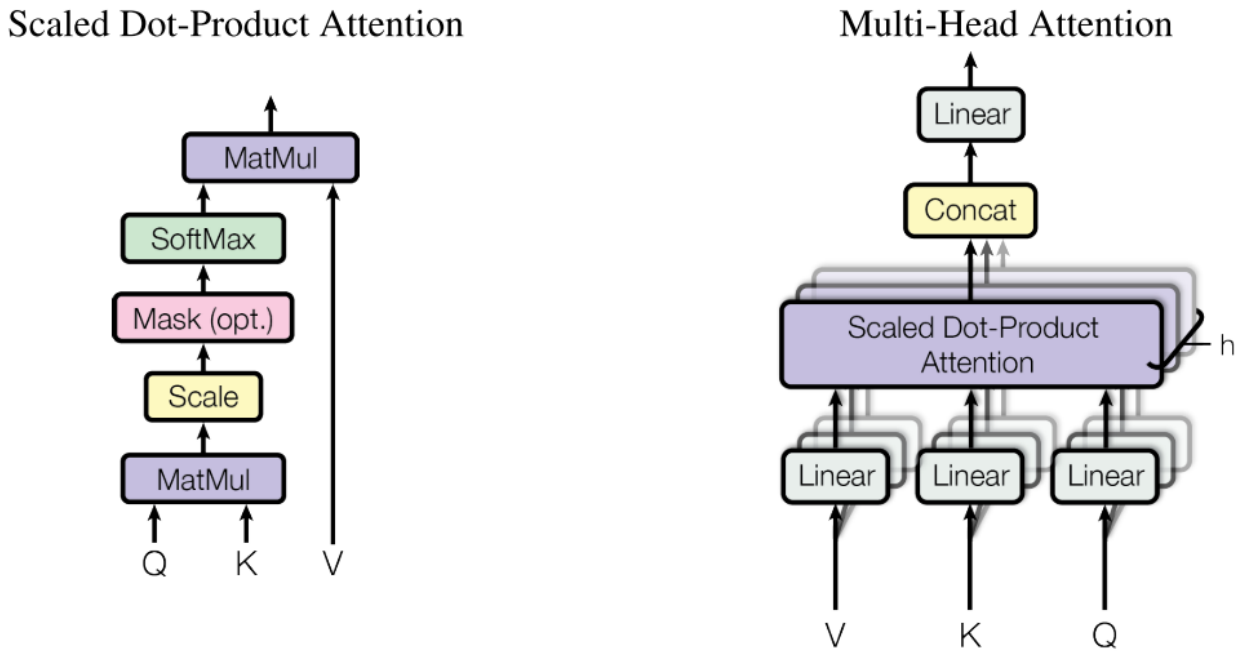

Figure 13.3.2: The multi-head attention architecture [2].

Multi-head attention mechanism plays important role in both encoder and decoder side; particularly, they are used in three places:

- In the encoder module, Multi-head attention is used to construct embedding for each token that depends of its context (i.e., self-attention of the input sequence).
- In the decoder module, Multi-head attention is used to construct embedding for each token that depends of its *seen* output context (i.e., masked self-attention of the output sequence).
- From the encoder module to the decoder module, Multi-head attention is used to construct embedding for each output token that depends of its *input* context (i.e., attention between input and output sequences).

Given a query matrix $Q \in \mathbb{R}^{n \times d_{model}}$, a key matrix $K \in \mathbb{R}^{m \times d_{model}}$, and a value matrix $V \in \mathbb{R}^{m \times d_{model}}$, the multi-head ($h$ heads) attention associated with (Q, K, V) is given by

$$\text{MultiHeadAttention}\,(Q, K, V) = Concat\,(head_1, \cdots, head_h)\, W^O$$

where $head_i \in \mathbb{R}^{n \times d_v}$

$$head_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

with $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{h \times d_v \times d_{model}}$. In general, we require $d_k = d_v = d_{model}/h$ such that the output of MultiHeadAttention (Q, K, V) has the dimensionality of $n \times d_{model}$.

The attention among (Q, K, V) is given by

$$\text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) = \text{softmax}\left(\frac{QW_i^Q\left(KW_i^K\right)^T}{\sqrt{d_k}}\right)VW_i^V.$$

Note that the softmax applies to each row such that the softmax$(\cdot)$ produces a weight matrix $w^{att} \in \mathbb{R}^{n \times m}$, with each row summing up to unit 1.

Explicitly, we have

$$\begin{bmatrix} w_{11}^{att} & w_{12}^{att} & \cdots & w_{1m}^{att} \\ w_{21}^{att} & w_{22}^{att} & \cdots & w_{2m}^{att} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{att} & w_{n2}^{att} & \cdots & w_{nm}^{att} \end{bmatrix} \begin{bmatrix} [VW^V]_1 \\ [VW^V]_2 \\ \vdots \\ [VW^V]_m \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^m w_{1j}^{att}[VW^V]_j \\ \sum_{j=1}^m w_{2j}^{att}[VW^V]_j \\ \vdots \\ \sum_{j=1}^m w_{nj}^{att}[VW^V]_j \end{bmatrix}$$

where $[VW^V]_i$ is the ith row vector of value matrix VW^V.

Sometimes, for each query associated with a symbol, we like to only allow a subset of keys and values to be queried via a binary **mask** $mask \in \{0, 1\}^m$, where 1 indicates exclusion of the key. We can set the values in the intermediate matrix $QW_i^Q(KW_i^K)^T$ to be negative infinity if this column index is associated with mask value 1.

13.3.1.4 *Encoder anatomy*

Given an input sequence represented by integer sequence $s = (i_1, ..., i_p, ..., i_n), i_p \in \mathbb{N}$, *e.g.*, $s = (3, 5, 30, 2, ..., 21)$, the goal of the encoder module [Figure 13.3.1] is to map s to n dense vectors that captures semantic, position, and contextual information. For the input sequence s, we first project s into a dense word embedding space via $WE(s) \in \mathbb{R}^{n \times d_{model}}$. The word embedding $WE(s)$ does not encode positional information in the sequence, so we utilize a position encoding PE maps an integer index representing the position of the token in the sequence, $s^p = (1, 2, ..., n)$ to n dense vector with same dimension d_{model}, i.e.,

$\text{PE}(s^p) \in \mathbb{R}^{n \times d_{model}}$. Notably, given the token position $i \in \{1, ..., n\}$ $\text{PE}(i) \in \mathbb{R}^{d_{model}}$ is given by

$$[\text{PE}(i)]_j = \begin{cases} \sin\left(\frac{i}{10000^{2j/d_{model}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{2j-1/d_{model}}}\right) & \text{if } j \text{ is odd} \end{cases}$$

Intuitively, each dimension of the positional encoding corresponds to a sinusoid of different wavelengths ranging from 2π to $10000 \cdot 2\pi$.

The whole computation in the encoder module can be summarized in the following.

Definition 13.3.1 (computation in encoder module). *Given an input sequence represented by integer sequence $s = (i_1, ..., i_p, ..., i_n)$ and its position $s^p = (1, ..., p, ..., n)$. The encoder module takes s, s^p as inputs and produce $e_N \in \mathbb{R}^{n \times d_{model}}$.*

$$\begin{aligned} e_0 &= \text{WE}(s) + \text{PE}(s^p) \\ e_1 &= \text{EncoderLayer}(e_0) \\ e_2 &= \text{EncoderLayer}(e_1) \\ &\cdots \\ e_N &= \text{EncoderLayer}(e_{N-1}) \end{aligned}$$

where $e_i \in \mathbb{R}^{n \times d_{model}}$, $\text{EncoderLalyer} : R^{n \times d_{model}} \to R^{n \times d_{model}}$ is an encoder sub-unit, N is the number of encoder sub-units. Specifically, this encoder sub-unit can be decomposed into following calculation procedures

$$\begin{aligned} e_{mid} &= \text{LayerNorm}(e_{in} + \text{MultiHeadAttention}(e_{in}, e_{in}, e_{in}, padMask)) \\ e_{out} &= \text{LayerNorm}(e_{mid} + \text{FFN}(e_{mid})) \end{aligned}$$

where $e_{mid}, e_{out} \in \mathbb{R}^{n \times d_{model}}$,

$$FFN(e_{mid}) = \max(0, e_{mid}W_1 + b_1)W_2 + b_2,$$

with $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}, b_1 \in \mathbb{R}^{d_{ff}}, b_2 \in \mathbb{R}^{d_{model}}$, and the padMask excludes padding symbols in s.

13.3.1.5 *Decoder anatomy*

In the decoder side, we are similarly given an output sequence represented by integer sequence $o = (o_1, ..., o_p, ..., o_m), o_p \in \mathbb{N}$, *e.g.*, $o = (5, 10, 30, 2, ..., 21)$. The decoder module [Figure 13.3.1] aims to converts an output sequence o, combining with resulting

embedding e_N in the encoder module [Definition 13.3.1] to its corresponding probabilities over the vocabulary. The output probability can be used to compute categorical loss that drives the learning process of the encoder and the decoder.

The whole computation in the encoder module can be summarized in the following.

Definition 13.3.2 (computation in decoder module). *Given an input sequence represented by integer sequence $o = (o_1, ..., o_p, ..., o_n)$ and its position $o^p = (1, ..., p, ..., m)$. The encoder module takes o, o^p as inputs, combines resulting embedding e_N from the encoder, and produce $d_N \in \mathbb{R}^{n \times d_{model}}$ and probabilities over the vocabulary.*

$$
\begin{aligned}
d_0 &= \text{WE}(o) + \text{PE}(o^p) \\
d_1 &= \text{DecoderLayer}(d_0, e_N) \\
d_2 &= \text{DecoderLayer}(d_1, e_N) \\
&\cdots \\
d_N &= \text{DecoderLayer}(d_{N-1}, e_N) \\
output\ prob &= \text{Softmax}(d_N W)
\end{aligned}
$$

where $d_0 \in \mathbb{R}^{m \times d_{model}}$, $\text{DecoderLayer} : R^{m \times d_{model}} \to R^{m \times d_{model}}$ is a decoder sub-unit, N is the number of decoder sub-units, $W \in \mathbb{R}^{d_{model} \times |V|^O}$. Specifically, this decoder sub-unit can be decomposed into following calculation procedures

$$
\begin{aligned}
d_{mid1} &= \text{LayerNorm}(d_{in} + \text{MaskedMultiHeadAttention}(d_{in}, d_{in}, d_{in}, padMask)) \\
d_{mid2} &= \text{LayerNorm}(d_{mid1} + \text{MaskedMultiHeadAttention}(d_{mid1}, e_N, e_N, padMask|seqMask)) \\
d_{out} &= \text{LayerNorm}(d_{mid2} + \text{FFN}(d_{mid2}))
\end{aligned}
$$

where $d_{mid1}, d_{mid2}, d_{out} \in \mathbb{R}^{m \times d_{model}}$,

$$FFN(d_{out}) = \max(0, d_{mid} W_1 + b_1) W_2 + b_2,$$

with $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}, b_1 \in \mathbb{R}^{d_{ff}}, b_2 \in \mathbb{R}^{d_{model}}$.

Note that there are two type of attention in the decoder module, one is self-attention among output sequence and one is attention between encoder output and decoder output. The encoder-decoder attention uses a mask that excludes padding symbol in the input sequence; the self-attention uses a mask that excludes padding symbol and future symbol, which can be computed via logical OR between *padMask* and *seqMask*. *seqMask* for a symbol at position i is a binary vector $seqMask_i \in \mathbb{R}^m$ whose value is 1 at position equal

or greater than i. Therefore, for a sequence of length m, the complete *seqMask* would be a upper triangle matrix value 1.