Transformers

At present the transformer architecture underlies the pre-eminent models utilised for the majority of natural language processing (NLP) tasks. The transformer model was introduced in the 2017 paper *Attention Is All You Need*. Prior to transformers, NLP models typically used recurrent neural networks (RNNs) which processed input tokens sequentially and maintained a fixed-length, context vector of the sequence up to the current token. In theory, the information from a token can continue to propagate through the sequence. However, in practice, as the input sequence grows in length, information from the earlier tokens is lost and only the information from the more recent tokens is retained. RNNs also suffer from the vanishing gradient problem, and since token computations depend on the results of previous token computations, parallelisation is challenging and training, therefore, inefficient. The key building block of the transformer, and the enabler of efficient parallelisation, is the attention mechanism, specifically self-attention, which can access all previous tokens, and weigh them according to relevance, and not proximity in the sequence.

1 Attention

The attention mechanism attempts to mimic the human cognitive process of selectively concentrating on relevant information whilst ignoring other less pertinent information. It permits the model to consider the whole input sequence, and focus on, or attend to, the most important parts. The attention mechanism takes inspiration from retrieval systems, in which a user's query is matched to the most similar key to return the corresponding value. Consider the database D, consisting of m tuples of keys, $\mathbf{k}_i$, and values, $\mathbf{v}_i$. For a query $\mathbf{q}$, the attention is defined as

$$A(\mathbf{q}, D) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \tag{1}$$

where $\alpha(\mathbf{q}, \mathbf{k}_i)$ are scalar attention weights, and measure the similarity, or compatibility, between the query and the keys. The attention is therefore a linear combination of values contained in the database, with more attention paid to values with larger weights, i.e., to values for which there is a greater similarity between the query and the key attached to the value. A traditional database query occurs if one of the weights is one, whilst all others are zero. The computation of attention is known as attention pooling. In transformers the attention mechanism is usually chosen to be differentiable, i.e., the attention weights (and the matrices used in their computation) are learned together with the rest of the model parameters.

2 Model Architecture

The transformer model consists of two components: an encoder and a decoder (Figure 1). The encoder produces a representation of the input sequence, and the decoder uses this representation to generate an output sequence.

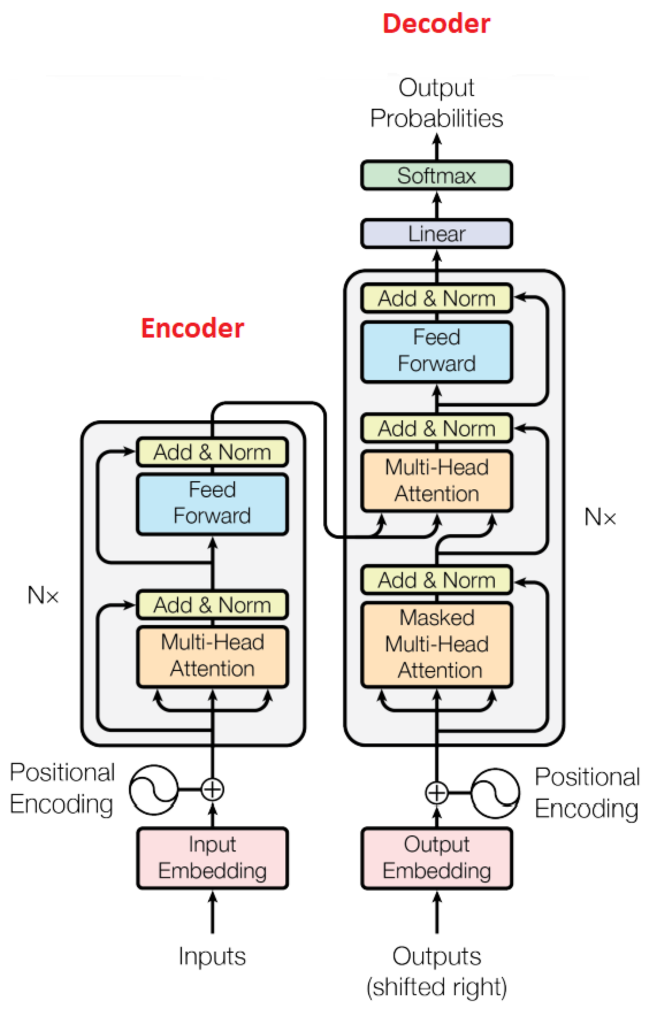


Figure 1: The transformer architecture. Image taken from Vaswani, et al. (2017).

3 The Encoder

Each encoder layer consists of two sub-layers: a multi-head self-attention mechanism and a position-wise, feed-forward network.

To begin with, vector embeddings are produced for the input sequence of tokens. Positional encoding is added to these embeddings, before they enter the multi-head attention layer where they are processed in parallel (simultaneously). A residual connection is added to the output which is subsequently normalised. The feed-forward network follows. A residual connection is added to its output, which is then normalised.

The output of an encoder layer is a vector representation for each position in the input sequence. There are multiple layers in the encoder (the original paper employs six), with the output of each layer acting as the input for the following layer.

3.1 Input Embedding

The encoder input is a sequence of tokens, typically words. Since machine learning models cannot process raw text, these words are converted into vectors.

A vocabulary dictionary is created by extracting all the words present in the training data. Each word is assigned a numeric index. The indices of the words in the input sequence are added to a list and presented to the encoder. For each word in the vocabulary dictionary the embedding layer contains an associated dense vector (a vector with mostly non-zero components). The components of these vectors are randomly initialised and updated during training. The embedding layer retrieves the vectors corresponding to the indices that are passed into it, therefore producing a vector representation of the original input sequence.

3.2 Positional Encoding

Unlike RNNs which process word embeddings sequentially, transformers process them simultaneously (in parallel). A consequence of this is that transformers have no awareness of the order of the tokens in the input. Therefore, some information about the relative or absolute positions of the tokens in the sequence must be inserted into the model. There are many different methods of positional encoding; in the original paper the authors utilise

$$\mathrm{PE}_{i,2j} = \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \tag{2}$$

$$\mathrm{PE}_{i,2j+1} = \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \tag{3}$$

where i is the position of the token in the sequence, j is the dimension index (current feature index), and d is the embedding dimension (total number of features). One of the advantages of this method is that the positional encodings have values in a normalised range, i.e., values in $[-1, 1]$.

As an example, say we have the input sequence “Show me the money”. Let’s say that the embeddings for the words in this sequence are

$$\text{show} \rightarrow \begin{bmatrix} 0.12 \\ 0.63 \\ 0.29 \\ 0.41 \end{bmatrix} \qquad \text{me} \rightarrow \begin{bmatrix} 0.83 \\ 0.34 \\ 0.04 \\ 0.53 \end{bmatrix} \qquad \text{the} \rightarrow \begin{bmatrix} 0.39 \\ 0.77 \\ 0.64 \\ 0.09 \end{bmatrix} \qquad \text{money} \rightarrow \begin{bmatrix} 0.41 \\ 0.08 \\ 0.51 \\ 0.87 \end{bmatrix}$$

The positional encoding of “show” will be

$$\begin{aligned}
\mathrm{PE}_{0,0} &= \sin\left(\frac{0}{10000^{\frac{2(0)}{4}}}\right) = 0\\
\mathrm{PE}_{0,1} &= \cos\left(\frac{0}{10000^{\frac{2(0)}{4}}}\right) = 1\\
\mathrm{PE}_{0,2} &= \sin\left(\frac{0}{10000^{\frac{2(1)}{4}}}\right) = 0\\
\mathrm{PE}_{0,3} &= \cos\left(\frac{0}{10000^{\frac{2(1)}{4}}}\right) = 1\\
\mathrm{PE}_{0} &= \begin{bmatrix} 0\\ 1\\ 0\\ 1 \end{bmatrix}
\end{aligned}$$

The positional encoding of "me" will be

$$\begin{aligned}
\mathrm{PE}_{1,0} &= \sin\left(\frac{1}{10000^{\frac{2(0)}{4}}}\right) = 0.02\\
\mathrm{PE}_{1,1} &= \cos\left(\frac{1}{10000^{\frac{2(0)}{4}}}\right) = 1\\
\mathrm{PE}_{1,2} &= \sin\left(\frac{1}{10000^{\frac{2(1)}{4}}}\right) = 0\\
\mathrm{PE}_{1,3} &= \cos\left(\frac{1}{10000^{\frac{2(1)}{4}}}\right) = 1\\
\mathrm{PE}_{1} &= \begin{bmatrix} 0.02\\ 1\\ 0\\ 1 \end{bmatrix}
\end{aligned}$$

The positional encodings of "the" and "money" will be, respectively,

$$\mathrm{PE}_{2} = \begin{bmatrix} 0.04\\ 1\\ 0\\ 1 \end{bmatrix} \qquad \mathrm{PE}_{3} = \begin{bmatrix} 0.05\\ 1\\ 0\\ 1 \end{bmatrix}$$

Unique positional encodings are generated for each token in the sequence, permitting absolute positions to be determined. As i gets smaller, the frequency of the trigonometric functions increases. This means that the first components of the positional encoding vector exhibit the largest variations and determine the position more precisely than the latter components.

This encoding scheme also enables relative positioning, in which the relative distances and pairwise relations between input tokens are learned. Since only the offsets between positions are considered, relative positioning performs well on sequences longer than those seen during training. Relative positional encodings are often invariant to absolute positional changes that impart minimal additional semantic context. Consider the sentences “the bank told the man he was bankrupt” and “then the bank told the man he was bankrupt”. The relative positions of the semantically important words “bank” and “man” are invariant in both sentences.

For any fixed position offset δ, the encoding at position $i + \delta$ can be represented by a linear transformation of the encoding at position i:

Denoting $\omega_j = \frac{1}{10000^{\frac{2j}{d}}}$, and using the identities $\sin(A + B) = \sin(A)\cos(B) + \cos(A)\sin(B)$ and $\cos(A + B) = \cos(A)\cos(B) - \sin(A)\sin(B)$

$$\begin{aligned}
M \begin{bmatrix} \mathrm{PE}_{i,2j} \\ \mathrm{PE}_{i,2j+1} \end{bmatrix} &= \begin{bmatrix} \cos(\delta\omega_j) & \sin(\delta\omega_j) \\ -\sin(\delta\omega_j) & \cos(\delta\omega_j) \end{bmatrix} \begin{bmatrix} \mathrm{PE}_{i,2j} \\ \mathrm{PE}_{i,2j+1} \end{bmatrix} \\
&= \begin{bmatrix} \cos(\delta\omega_j)\mathrm{PE}_{i,2j} + \sin(\delta\omega_j)\mathrm{PE}_{i,2j+1} \\ -\sin(\delta\omega_j)\mathrm{PE}_{i,2j} + \cos(\delta\omega_j)\mathrm{PE}_{i,2j+1} \end{bmatrix} \\
&= \begin{bmatrix} \cos(\delta\omega_j)\sin(i\omega_j) + \sin(\delta\omega_j)\cos(i\omega_j) \\ -\sin(\delta\omega_j)\sin(i\omega_j) + \cos(\delta\omega_j)\cos(i\omega_j) \end{bmatrix} \\
&= \begin{bmatrix} \sin[(i+\delta)\omega_j] \\ \cos[(i+\delta)\omega_j] \end{bmatrix} \\
&= \begin{bmatrix} \mathrm{PE}_{i+\delta,2j} \\ \mathrm{PE}_{i+\delta,2j+1} \end{bmatrix}
\end{aligned} \tag{4}$$

Therefore, given any two positional encoding vectors PE_i and $\mathrm{PE}_{i+\delta}$, the offset δ can be determined, enabling relative positioning.

In the encoder, positional encodings are added to the input representation. Returning to our example sentence “Show me the money”, for the word “money” the positional encoding is added to the word embedding as shown below:

$$\begin{bmatrix} 0.41 \\ 0.08 \\ 0.51 \\ 0.87 \end{bmatrix} + \begin{bmatrix} 0.05 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.46 \\ 1.08 \\ 0.51 \\ 1.87 \end{bmatrix} \tag{5}$$

In practice, the positional encoding matrix $P \in \Re^{n \times d}$, where n is the number of tokens in the sequence, and d is the embedding dimension, is added to the input representation matrix $X' \in \Re^{n \times d}$:

$$X = X' + P \tag{6}$$

3.3 The Self-Attention Mechanism

Transformers utilise an attention mechanism known as self-attention. Attention as it was originally conceived, provided a way for the decoder producing the output sequence to attend on the input sequence, and to focus on different parts of the input at different decoding steps, i.e., decoder steps attending to encoder steps. Self-attention, in contrast, allows the encoder to attend to other tokens in the input sequence during processing, i.e., each token in the input sequence is able to attend to every other token.

Following the input embedding and the positional encoding we have the input representation matrix $X \in \Re^{n \times d}$, where n is the number of tokens in the sequence, and d is the embedding dimension. To get the query, key, and value matrices, we multiply X by the weight matrices $W^q \in \Re^{d \times d_q}$, $W^k \in \Re^{d \times d_k}$, and $W^v \in \Re^{d \times d_v}$, respectively:

$$XW^q = Q, \qquad Q \in \Re^{n \times d_q} \tag{7}$$

$$XW^k = K, \qquad K \in \Re^{n \times d_k} \tag{8}$$

$$XW^v = V, \qquad V \in \Re^{n \times d_v} \tag{9}$$

Note that the dimensions of W^q and W^k are identical, i.e., $d_k = d_q$. This of course means that the dimensions of Q and K are also identical. The above matrix multiplications are undertaken in the encoder by a linear layer.

The attention mechanism utilised in transformer self-attention is the scaled dot product attention (Figure 2). The dot product is commonly used to measure the similarity between two vectors, and is employed in our case to measure the similarity between query and key vectors. The rows of the matrices Q and K, respectively, contain these vectors, so the dot product computation for all query and key vectors can be performed simultaneously, but independently from one another, i.e., in parallel, by a matrix multiplication of Q and K:

$$A(K, Q, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V = SV \in \Re^{n \times d_v} \tag{10}$$

Denoting $F = \left(\frac{QK^T}{\sqrt{d_k}}\right) \in \Re^{n \times n}$, the elements of the matrix $S = \text{softmax}(F)$ are given by

$$S_{ij} = \frac{\exp(F_{ij})}{\sum_{k=1} \exp(F_{ik})} \tag{11}$$

i.e., each row of S sums to one. The scaled dot product attention is differentiable, which enables X and the weight matrices W^i to be updated during training. The scaling factor $\frac{1}{\sqrt{d_k}}$ ensures that for large values of d_k, where QK^T may also be large, the softmax function isn't used in regions where it has extremely small gradients.

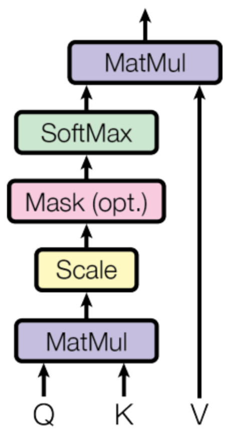
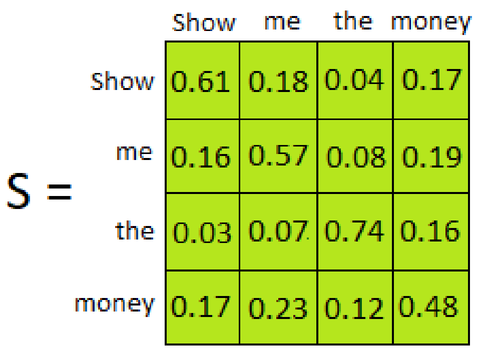


Figure 2: Scaled dot product attention. Image taken from Vaswani, et al. (2017).

Let's consider an example for an input sequence of four tokens: "Show me the money". The matrix S (sometimes called the attention filter) for this sequence is



$$S = \begin{array}{c|cccc} & \text{Show} & \text{me} & \text{the} & \text{money} \\ \hline \text{Show} & 0.61 & 0.18 & 0.04 & 0.17 \\ \text{me} & 0.16 & 0.57 & 0.08 & 0.19 \\ \text{the} & 0.03 & 0.07 & 0.74 & 0.16 \\ \text{money} & 0.17 & 0.23 & 0.12 & 0.48 \end{array}$$

The rows hold the softmax attention scores for each word in the sentence, therefore indicating the extent to which each word in the sentence attends to every other word. Scores between the same word are usually highest in practice, with the next highest score going to the next most similar, or important word. For example, the word "the" attends to itself primarily, and then the word "money".

3.4 Multi-Head Attention

The previous section discussed the self-attention mechanism. Transformers utilise multi-head self-attention, in which as the name suggests, there are multiple heads of the self-attention mechanism. The difference between each of the heads is that they employ their own distinct W^q, W^k and W^v matrices, and therefore produce unique Q, K and V matrices. Each attention head aims to learn a unique linguistic feature of the input sequence. The attention heads are calculated in parallel and their outputs subsequently concatenated together (Figure 3). To negate the concatenated output growing in length for each head used it is passed through a linear layer to reduce the dimensionality to that of the input representation.

Mathematically, each attention head, $h_i (i = 1, 2, ..., h)$, is computed as

$$h_i = A(K, Q, V) \in \Re^{n \times d_v} \tag{12}$$

$$K = XW_i^k, \qquad Q = XW_i^q, \qquad V = XW_i^v \tag{13}$$

where A is the self-attention mechanism and the projections are the parameter matrices W_i^q, W_i^k and W_i^v.

Multi-head attention is then

$$M(K, Q, V) = \text{Concat}(h_1, h_2, ..., h_h)W^0 \in \Re^{n \times d} \tag{14}$$

where $W^0 \in \Re^{hd_v \times d}$ is a trainable parameter matrix representing a linear layer projection.

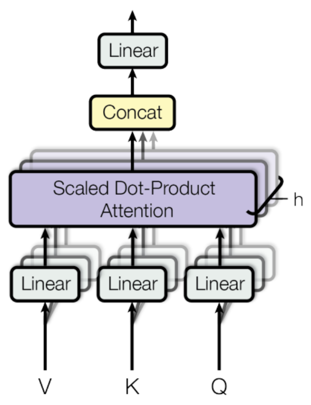


Figure 3: Multi-head attention. Note that here V, K, and Q are copies of X. Image taken from Vaswani, et al. (2017).

3.5 Residual Connections

In the encoder, residual connections are made to the output of the multi-head attention sub-layer, and to the output of the feed-forward network. Residual connections serve two main purposes: they allow data to bypass certain network sub-layers, preserving knowledge of useful information that would otherwise be lost as it is propagated forward, and by enabling the gradient to skip layers during backpropagation, help to address the vanishing gradient problem (by reducing the number of potentially very small partial derivative factors that occur in the chain rule formula for the partial derivative of the cost function with respect to a weight in an earlier layer of the network).

3.6 Add and Norm

Following the addition of residual connections, the results undergo a process known as layer normalisation. The i-th row and j-th column element of the matrix input, Y is standardised according to

$$Y'_{ij} = \frac{Y_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{15}$$

where μ_i and σ_i are the i-th row mean and standard deviation, respectively, and ϵ is included for numerical stability in case the standard deviation is zero. Layer normalisation stabilises training by normalising (re-scaling and re-centring) the gradients used in backward propagation. Consequently, it improves model performance and reduces training time.

The output of the add and norm operation is

$$\text{LayerNorm}[x + \text{Sublayer}(x)] \tag{16}$$

where x is the sub-layer input. The output of the first encoder add and norm operation, for example, is

$$\text{LayerNorm}[X + M(X)]$$

where X is the input representation after positional encoding and M is multi-head attention.

3.7 Feed-Forward Network

A position-wise feed forward network follows the multi-head attention sub-layer. It consists of two linear transformations with a ReLU activation in between. Mathematically, the network can be expressed as

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{17}$$

where x is the input to the network, W_1 is the weight matrix utilised for the first linear transformation (performed by the first fully-connected layer), b_1 is an added bias term, W_2 is the weight matrix used for the second linear transformation (performed by the second fully-connected layer), and b_2 is another added bias term.

Position-wise refers to the linear transformations being identical across different positions, i.e., the same weights are applied to every token representation in the sequence.

4 The Decoder

The role of the transformer decoder is to take the vectorised representation of the input sequence produced by the encoder and generate a new output sequence. Whereas the encoder only takes a single input, the decoder takes two: the encoder output, and its own previously generated output (during inference). The decoder is composed of multiple identical layers (the original paper employed six), with the output from one layer serving as input to the next. Each layer consists of three sub-layers (Figure 1); two are identical to those used in the encoder, and the other is a modified version of the multi-head attention mechanism which ensures that the decoder remains autoregressive during training.

4.1 Inference

To begin, the input sequence is processed by the encoder to generate a representation. The first input to the decoder is the <start> token, which is converted to a vector embedding before a positional encoding is added. The result serves as the input for the masked multi-head attention sub-layer. A residual connection is made to the sub-layer output before it undergoes layer normalisation. The result is then input into a normal multi-head attention sub-layer as Q, whilst identical copies of the representation produced by the encoder stack are input as K and V. This mechanism is known as encoder-decoder attention (or cross-attention) and allows the input and target sequences to be related to one another, i.e., each position in the decoder can attend over all positions in the input sequence. A residual connection from the previous sub-layer is then added prior to layer normalisation. A feed-forward network (with residual connection and layer normalisation) follows, with the output used as the input for the masked multi-head attention sub-layer in the next decoder layer. After all decoder layers have been processed, the output is fed into a linear layer of fully-connected neurons which are equal in number to the vocabulary dictionary size. A logits vector (a vector of raw unnormalised scores, equal in size to the vocabulary dictionary) is produced for each decoder input token. The logits vector corresponding to the most recent input token is passed to a softmax layer which generates a probability distribution over the vocabulary dictionary. If employing a greedy sampling approach, the decoder output is the token in the vocabulary with the highest assigned probability. Random sampling methods, such as top-k, and top-p sampling are often preferred to avoid issues such as repetitive text. The token output by the decoder is then added to the decoder input for the next iteration.

For example, let's say that the decoder initially predicts the word "not". When predicting the next token in the target sequence, the inputs to the decoder are now <start> and "not". If the next word predicted is "on", then the decoder inputs on the subsequent iteration are <start>, "not" and "on". The decoder continues adding the token predicted at the previous time step to the input for the current time step until an <end> token is produced.

4.2 Training

During training, we have known (input sequence, output sequence) pairs. The input sequence is processed by the encoder ready to be used by the decoder in the cross-attention mechanism. In contrast to inference, where one token is added to the decoder input and one token predicted at each time step, in training, the complete target sequence is input into the decoder, and the entire predicted sequence is computed in parallel, massively improving training efficiency (Figure 4). For training to imitate inference, the model has to employ some method to keep the decoder autoregressive, i.e., some method to ensure that a token can only attend on the tokens that precede it in the sequence, and not those that follow.

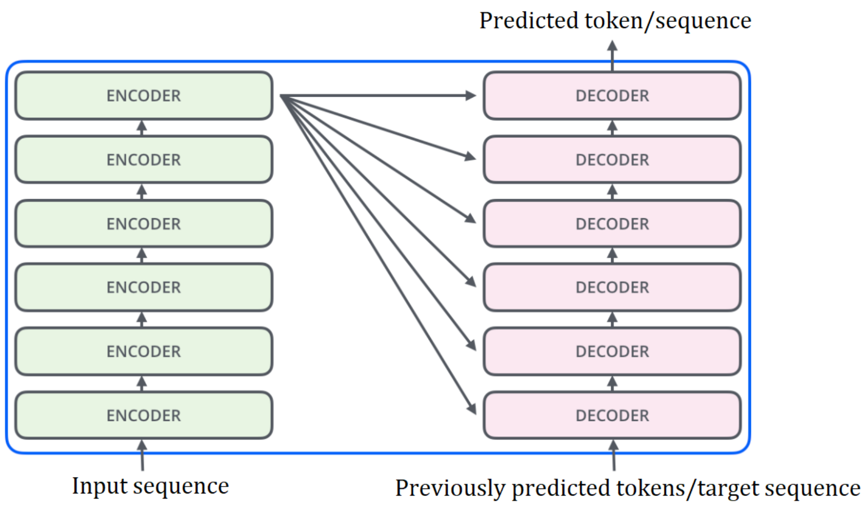


Figure 4: During both training and inference the first encoder layer receives the input sequence and generates a representation, which is input into the following encoder layer. In turn, the output of this layer is the input of the next, and so on. The output from the final layer is used in the cross-attention mechanism in each decoder layer. During inference the decoder predicts a single token at each time step, which is then added to the decoder input at the next time step. During training, the entire target sequence is input into the decoder and a sequence is predicted. As with the encoder, the output of one decoder layer is the input of the next. Image taken from Alammar (2018).

The masked multi-head attention sub-layer functions similarly to the normal multi-head attention sub-layer; its input is a representation of an entire sequence and it outputs an improved version. It calculates self-attention via the same paralellised mechanism (matrix multiplication between Q and K), but with one crucial difference: the introduction of a look-ahead mask into the scaled dot product attention. A masking matrix, $B \in \Re^{n\times n}$, is inserted into equation (10):

$$A(K,Q,V) = \text{softmax}\left(\frac{QK^T + B}{\sqrt{d_k}}\right) V = S'V \in \Re^{n\times d_v} \tag{18}$$

The elements of B above the main diagonal are set to $-\infty$, whereas the main diagonal and the elements below it are set to zero. The masking matrix acts to ensure that a token can only attend on others that precede it in the sequence, and not those that follow.

Let's consider an example to see how this works in practice. Say the correct target sequence associated with the input sequence "Show me the money" is "Not on your life!". The masked attention computation is shown below:

$$
QK^T + B = \begin{array}{c|cccccc}
 & \text{<start>} & \text{not} & \text{on} & \text{your} & \text{life} & \text{<end>} \\
\hline
\text{<start>} & 16.2 & 7.7 & 2.0 & 4.6 & 3.1 & 8.9 \\
\text{not} & 3.2 & 21.5 & 6.1 & 8.4 & 8.1 & 5.2 \\
\text{on} & 1.9 & 4.3 & 18.7 & 6.5 & 7.2 & 3.3 \\
\text{your} & 2.7 & 4.1 & 7.6 & 22.3 & 9.0 & 3.8 \\
\text{life} & 4.4 & 5.8 & 6.9 & 5.6 & 24.5 & 4.7 \\
\text{<end>} & 4.0 & 3.9 & 6.7 & 7.3 & 6.1 & 16.8
\end{array}
\quad + \quad
\begin{array}{c|cccccc}
 & \text{<start>} & \text{not} & \text{on} & \text{your} & \text{life} & \text{<end>} \\
\hline
\text{<start>} & 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\
\text{not} & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\
\text{on} & 0 & 0 & 0 & -\infty & -\infty & -\infty \\
\text{your} & 0 & 0 & 0 & 0 & -\infty & -\infty \\
\text{life} & 0 & 0 & 0 & 0 & 0 & -\infty \\
\text{<end>} & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

$$
= \begin{array}{c|cccccc}
 & \text{<start>} & \text{not} & \text{on} & \text{your} & \text{life} & \text{<end>} \\
\hline
\text{<start>} & 16.2 & -\infty & -\infty & -\infty & -\infty & -\infty \\
\text{not} & 3.2 & 21.5 & -\infty & -\infty & -\infty & -\infty \\
\text{on} & 1.9 & 4.3 & 18.7 & -\infty & -\infty & -\infty \\
\text{your} & 2.7 & 4.1 & 7.6 & 22.3 & -\infty & -\infty \\
\text{life} & 4.4 & 5.8 & 6.9 & 5.6 & 24.5 & -\infty \\
\text{<end>} & 4.0 & 3.9 & 6.7 & 7.3 & 6.1 & 16.8
\end{array}
$$

After scaling, and applying the softmax function to each row, the result is

$$
S' = \begin{array}{c|cccccc}
 & \text{<start>} & \text{not} & \text{on} & \text{your} & \text{life} & \text{<end>} \\
\hline
\text{<start>} & 1 & 0 & 0 & 0 & 0 & 0 \\
\text{not} & 0.13 & 0.87 & 0 & 0 & 0 & 0 \\
\text{on} & 0.06 & 0.16 & 0.78 & 0 & 0 & 0 \\
\text{your} & 0.03 & 0.06 & 0.11 & 0.80 & 0 & 0 \\
\text{life} & 0.05 & 0.06 & 0.09 & 0.06 & 0.74 & 0 \\
\text{<end>} & 0.02 & 0.04 & 0.14 & 0.15 & 0.13 & 0.52
\end{array}
$$

The matrix S' is then multiplied with the values matrix V.

Since the elements above the main diagonal of S' are zero, a token in the target sequence cannot attend on others that follow it in the sequence. For instance, in the above example, the token "on" cannot attend to the "your", "life" and <end> tokens. If the masking matrix B was not used in training, a token could cheat and attend completely on the next token in the sequence, i.e., attend on the token the decoder is trying to predict. Also notice that because the decoder input is the target sequence, a token can only attend on correct tokens and not on, possibly erroneous, predicted tokens. This training strategy is known as teacher forcing.

The output exiting the masked multi-head attention layer passes through the rest of the decoder as was described for inference, the only difference being that all the logits vectors are passed to the softmax layer, with a probability distribution over the vocabulary being computed for each one. In this manner, an entire sequence is generated.

Cross-entropy loss is calculated between the predicted token and target token labels and averaged across the sequence. In practice this can be done with batches of (input sequence, target sequence) pairs.

5 Performance

Considering the matrix S in section 3.3 it is clear that the self-attention computation is quadratic with respect to the length of the input, i.e. for a sequence of n tokens, the self-attention mechanism performs n^2 calculations. This makes it extremely expensive for the transformer to take long documents of text as its input, and in practice the input has to be limited to at most a paragraph, or page of text.

For efficient computation the encoder input is fixed in size. However, by adding padding tokens to the input sequence so that its size is equal to the predefined fixed size, input sequences of variable length can be processed. The decoder output is equal in size to the decoder input during training (n tokens to n tokens), due to the self-attention mechanism. During inference, the decoder produces variable length output - one token per time step

References

Alammar, J. (2018). *The Illustrated Transformer [Blog post].* Retrieved from https://jalammar.github.io/illustrated-transformer/

Jurafsky, D., Martin, J.H. (2023). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 3rd ed. draft.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I. (2017). *Attention Is All You Need.* arXiv:1706.03762v5.

Zhang, A., Lipton, Z.C., Li, M., Smola, A.J. (2023). *Dive into Deep Learning.* arXiv:2106.11342v4.

Appendix

Self-Attention in Vector Form

To gain more intuition into how the self-attention mechanism works, it can be useful to review the computation in vector form. Let's say we have the input sequence "help me!" which consists of the tokens "help" and "me". The former has the embedding vector $\mathbf{x}_1$, and the latter $\mathbf{x}_2$, with each consisting of four embedding features, i.e., $d = 4$. In practice the steps outlined below occur in parallel, since the individual embedding vectors form the rows of the matrix X.

Input: "help"
Embedding: $\mathbf{x}_1 = [x_{11}, x_{12}, x_{13}, x_{14}]^T$
Query: $W^q\mathbf{x}_1 = \mathbf{q}_1$, $d_q = 16$
Key: $W^k\mathbf{x}_1 = \mathbf{k}_1$, $d_k = d_q$
Value: $W^v\mathbf{x}_1 = \mathbf{v}_1$
Scores: $\mathbf{q}_1 \cdot \mathbf{k}_1 = 6$, $\mathbf{q}_1 \cdot \mathbf{k}_2 = 4$
Scaled scores $\frac{1}{\sqrt{16}}$: 1.5, 1
Softmax: 0.62, 0.38
Attention: $A_1 = 0.62\mathbf{v}_1 + 0.38\mathbf{v}_2$

Input: "me"
Embedding: $\mathbf{x}_2 = [x_{21}, x_{22}, x_{23}, x_{24}]^T$
Query: $W^q\mathbf{x}_2 = \mathbf{q}_2$, $d_q = 16$
Key: $W^k\mathbf{x}_2 = \mathbf{k}_2$, $d_k = d_q$
Value: $W^v\mathbf{x}_2 = \mathbf{v}_2$
Scores: $\mathbf{q}_2 \cdot \mathbf{k}_1 = 3$, $\mathbf{q}_2 \cdot \mathbf{k}_2 = 8$
Scaled scores $\frac{1}{\sqrt{16}}$: 0.75, 2
Softmax: 0.22, 0.78
Attention: $A_2 = 0.22\mathbf{v}_1 + 0.78\mathbf{v}_2$