Transformer Details

Krishna Chaitanya Bhatraju

We assume the reader is familiar with the general concepts behind the Transformer architecture for sequence-to-sequence modeling as discussed here, and therefore we do not cover the intuitions behind the layers discussed. Rather, we delve into the specifics of how they are implemented.

1 MultiHeadAttention Block

1.1 Inputs

Input	Shape
P^Q	(B, S_q, d_m)
P^K	(B, S_k, d_m)
P^V	(B, S_k, d_m)

P^Q, P^K, and P^V are used to develop the query (Q), key (K), and value (V) inputs to the attention module, respectively. P^Q is a collection of B sequences of length S_q, with every element of each sequence being a vector of dimension d_m. P^K and P^V are defined similarly.

Note that P^K and P^V are different learned representations of the same collection of sequences, often called the context, explaining why they share the same penultimate dimension S_k. Further note that the batch size B must be equal across all three inputs. In particular, each query $P^Q[i]$ will search for context within the corresponding keys $P^K[i]$ and values $P^V[i]$ for all $i \in \{0, \ldots, B-1\}$.

As an example, in machine translation (specifically in the second MHA block of the decoder), P^Q is learned from a batch of sentences in the target language, while P^K and P^V are learned from a batch of sentences in the input language. $P^Q[i]$ therefore corresponds to a representation of the i-th translated sentence, while $P^K[i]$ and $P^V[i]$ are representations of the corresponding i-th input sentence. Thus, it is natural for $P^Q[i]$ to only ascertain context from $P^K[i]$ and $P^V[i]$ in this cross-attention block.

While in theory, this block could be easily extended to support different embedding dimensions for P^K, P^Q, and P^V, the authors of the linked paper described the architecture as supporting only a singular embedding dimension d_m shared across all the inputs (called $\texttt{d}_{\texttt{model}}$ in the reference).

1.2 Weights and Hyperparameters

Weight	Shape
W^Q	(h, d_m, d_k)
W^K	(h, d_m, d_k)
W^V	(h, d_m, d_v)
W^O	$(h \times d_v, d_m)$

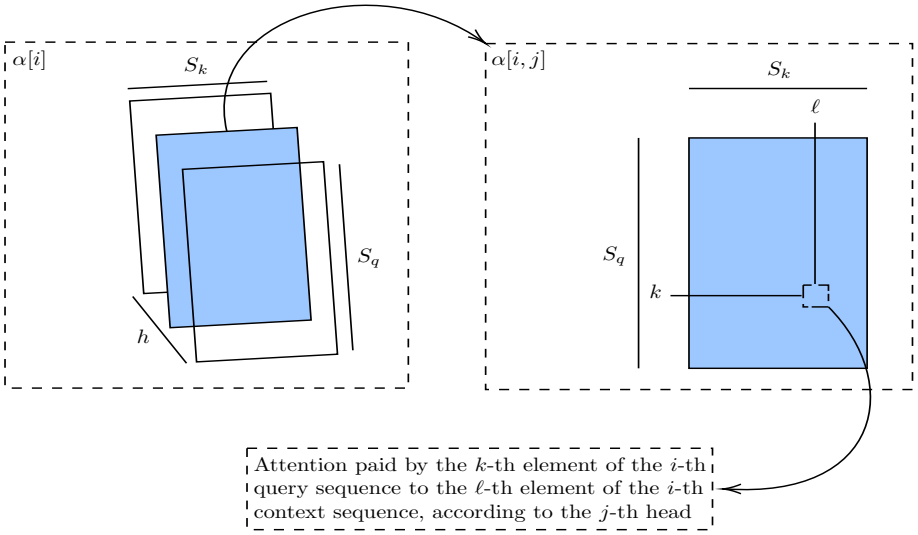
Hyperparameter	Meaning
h	Num. of attention heads
d_k	P_q/P_k projection dim.
d_v	P_v projection dim.

The definition of these weights is slightly different to how the block is implemented in Keras and PyTorch, but this formulation is computationally equivalent and much easier to understand. In most implementations, h is defined by the user, and $d_k = d_v = d_m/h$. In this formulation, d_m must be divisible by h.

1.3 Method

1.3.1 Attention Scores

Our first goal in an MHA block is to develop attention scores α of shape (B, h, S_q, S_k), where $\alpha[i, j, k, \ell]$ intuitively represents how much attention the k-th element of $P^Q[i]$ should pay to the ℓ-th elements of $P^K[i]$ and $P^V[i]$ when developing attention vectors, according to the j-th head, as shown below.

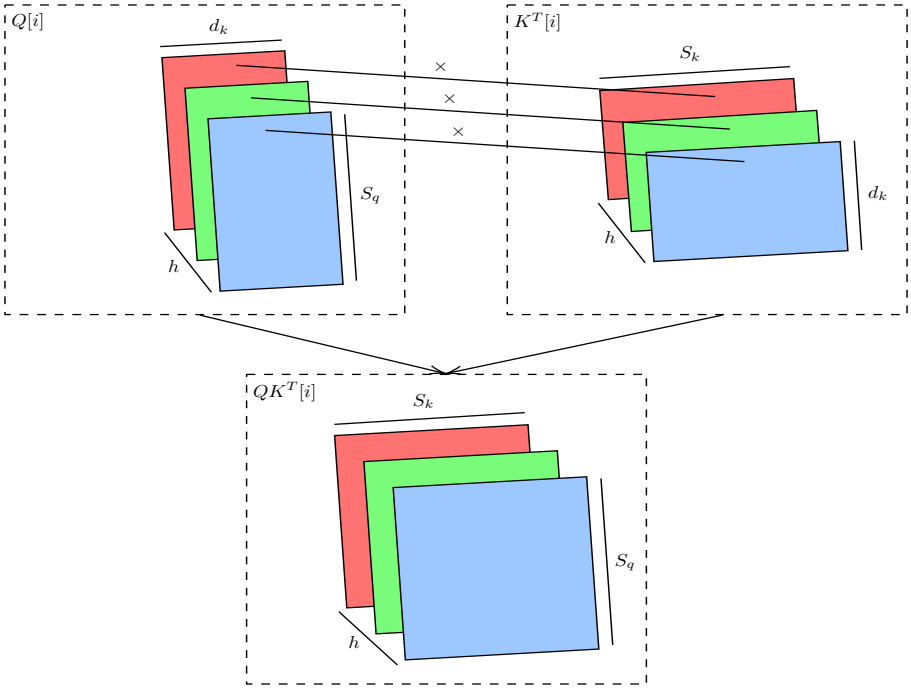


To do this, we first linearly project P^Q, P^K, and P^V to h separate learned representations as shown below. We now discuss the formulation of the query Q, with the key and value computations being completely analogous.

Projection	Shape
(Query) $Q = (P^Q[:, \texttt{None}])W^Q$	(B, h, S_q, d_k)
(Key) $K = (P^K[:, \texttt{None}])K^K$	(B, h, S_k, d_k)
(Value) $V = (P^V[:, \texttt{None}])W^V$	(B, h, S_k, d_v)

We first reshape P^Q to $(B, 1, S_q, d_m)$, to allow for broadcasting with W^Q of shape (h, d_m, d_k). For the sake of brevity, let $P^{Q'}$ be the reshaped input. When computing $Q = P^{Q'}W^Q$, each sequence in $P^{Q'}$ of shape $(1, S_q, d_m)$ is copied h times and the weight matrix W^Q is copied B times to yield tensors of shapes (B, h, S_q, d_m) and (B, h, d_m, d_k), respectively, which are then multiplied. The overall effect is that the j-th copy of the i-th sequence in $P^{Q'}$ is multiplied by the j-th head in the i-th copy of W^Q for all $i \in \{0, \ldots, B-1\}$, $j \in \{0, \ldots, h-1\}$. More specifically, the j-th projection of the i-th sequence can be written as $Q[i, j] = P^Q[i]W^Q[j]$. Overall, Q is a collection of h different learned projections of dimension d_k for each sequence in P^Q.

The raw attention scores are given by QK^T, for reasons we assume the reader is already familiar with. It is important to note, however, that each head of the MHA block computes separate independent attention scores, allowing for parallelization as shown below. As a result, each head produces its own set of attention vectors which are eventually concatenated.



The final attention scores are computed as $\alpha = \texttt{softmax}(QK^T/d_k + M)$, where d_k is used simply as a normalization factor. There are two types of masks that are generally used, padding and look-ahead/causal masks. The former disallow extraneous elements (eg., padding tokens in machine translation) to be considered for attention when learning vector representations. The latter disallow earlier elements to pay attention to future elements of the same sequence. These are discussed in the following section.

1.3.2 Masking

Padding masks are the more interesting of the two, as they usually require some pre-processing before use. It should be clear that extraneous elements of context sequences must be restricted from receiving attention as they do not contain any useful information for query sequences. More specifically, we require $\alpha[i, :, :, \ell] = 0$ if the ℓ-th element of the i-th context sequence is extraneous.

However, extraneous elements of queries generally do not need to be masked out. This is because most implementations use a loss function that prevents the final prediction associated with such elements from contributing to the loss. Thus, if the k-th element of the i-th query sequence is extraneous, the attention scores $\alpha[i, :, k, :]$ will be effectively masked out by the loss function, so they do not need to be set to 0 at this stage. (For clarity, it is perfectly correct to do so, but it is slightly more difficult to implement and leads to some minor complications.)

A padding mask is generally supplied to the attention layer as a tensor P of shape (B, S_k), where $P[i, j] = 0$ if the j-th element of the i-th context sequence is extraneous and 1 otherwise. Recall that if the ℓ-th element of the i-th context sequence is extraneous, we require $\alpha[i, :, :, \ell] = 0$. As $\alpha = \texttt{softmax}(QK^T/d_k + M)$, we therefore require $M[i, :, :, \ell] \approx -\infty$. Oftentimes, this is achieved by setting $M[i, :, :, \ell]$ to a very large negative number (e.g., -10^9).

Note that by definition, $P[i, \ell] = 0$, so define $M' = (1 - P) \times (-10^9)$ and $M = M'[:, \texttt{None}, \texttt{None}, :]$. Thus, M' shares the same shape as P and satisfies $M[i, \ell] = -10^9$, while M is of shape $(B, 1, 1, S_k)$ and satisfies $M[i, :, :, \ell] = -10^9$, as desired. Furthermore, as $P[i, \ell'] = 1$ when the ℓ'-th element of the i-th context sequence is not extraneous, $M[i, :, :, \ell'] = 0$, so attention mapped to useful context elements is not modified. More intuitively, $M[i]$ is a $1 \times 1 \times S_k$ tensor that represents the i-th context sequence with each element either -10^9 or 0. When added to QK^T/d_k, this tensor is broadcast twice. First, the $1 \times 1 \times S_k$ representation is copied length-wise S_q times into shape $1 \times S_q \times S_k$, which is then copied depth-wise h times into shape $h \times S_q \times S_k$. Ultimately, M is broadcast to shape (B, h, S_q, S_k) still satisfying the above properties so that every query sequence across every attention head has access to which context elements are extraneous.

Look-ahead/causal masks are more straightforward as they are generally constructed within the attention layer itself. Once again, this document assumes the reader is familiar with the intuition behind the inclusion of this mask and solely focuses on implementation.

The goal is to generate a mask L of shape (B, h, S_q, S_k) such that across all sequences i and attention heads j, $L[i, j]$ is an $S_q \times S_k$ lower-triangular matrix with on-diagonal and lower-diagonal elements all equal to 1. This lets us define $M = (1 - L) \times (-10^9)$ with the same intuition as the padding masks described above. Thankfully, this is easily implemented with `tf.linalg.band_part`.

1.3.3 Attention Vectors

Vectors	Shape
$A' = \alpha V$	(B, h, S_q, d_v)
$A = \texttt{reshape}(A')$	$(B, S_q, h \times d_v)$

Continuing with the theme of parallelization, for all sequences i, each attention head present in $\alpha[i]$ of shape (S_q, S_k) is independently multiplied with the corresponding linear projection in $V[i]$ of shape (S_k, d_v). Thus, for each sequence, A' is populated with h independent representations, each of shape (S_q, d_v). Finally, A' is reshaped in such a way that the attention heads are stacked width-wise, not depth-wise. This is a natural way to concatenate all h independent projections of each query element into a single representation of dimension $h \times d_v$.

1.4 Output

Output	Shape
$O = \texttt{dropout}(AW^O)$	(B, S_q, d_m)

Note that O shares the same shape as P^Q. Thus, an MHA block does not change the representation dimension of each element in P^Q, but modifies the input representation to account for context given by P^K and P^V. Typically, a dropout layer with low probability (i.e, 0.1) is added to the end of an MHA block for robustness, as shown above.

1.5 Inference

The use of Transformers at inference time has been a subject of debate, specifically the inputs to and use of a look-ahead mask in the MHA blocks of the decoder. This section attempts to clarify these topics, using machine translation as an example throughout.

At inference time, the Transformer is much less efficient than at training as teacher forcing is no longer a possibility. Thus, at this stage, the generation of the output occurs autoregressively: the decoder takes as input all of the previously generated elements in order to predict the next token of the sequence. This is repeated until termination (in machine translation, this would be an `[END]` token being predicted or the sequence limit being reached). Note that the encoder outputs are fixed throughout this process and are not modified after the first pass-through.

Suppose that a Transformer has already generated $g \geq 0$ elements for B input sequences at inference time. These, along with some indicator element (a `[START]` token in machine translation) are passed to the initial embedding and positional encoding layers of the decoder to yield an input I of shape $(B, g+1, d_m)$. There are two general ways to proceed from here, although the second is usually preferred because it is less computationally expensive. Let E be the context from the encoder of shape (B, S_k, d_m).

1. Let $P^Q = P^K = P^Q = I$ for the first MHA block in the decoder and let $P^Q = I$ with $P^K = P^Q = E$ for the second MHA block. Rather counter-intuitively, for the purposes of machine translation, it does not matter whether or not the look-ahead mask is applied in the first MHA block. This is because only the final embedding within each sequence is used to make a prediction about the next token.

   In particular, if the decoder output is O of shape $(B, g+1, d_m)$, only $O[:, -1, :]$ of shape (B, d_m) is used to make the next prediction for each sequence. Thus, the only attention vectors that contribute to the prediction are the ones associated with the last elements of each sequence of the input (i.e., the latest generated elements). The look-ahead mask does not affect those vectors, so its use does not affect the end result for machine translation tasks. However, if the desired output is a rich representation of the entire output generated thus far, look-ahead masks once again become necessary as they greatly affect attention vectors for earlier elements.

2. Let $P^Q = I[:, -1, :]$ for both MHA blocks in the decoder with $P^K = P^Q = I$ for the first block and $P^K = P^Q = E$ for the second block. This should be a straightforward optimization over the previous method (at least for machine translation), as the only attention vectors computed now are ones that will be used for prediction. Thus, as mentioned before, this is much less computationally prohibitive. In this case, a look-ahead mask cannot be used in the first MHA block as the only elements that comprise the context are those that have been generated before each query element.