On the Music Transformer

Aditya Gomatam
March 24, 2021

1 Introduction

The Music Transformer or Transformer with Relative Self-Attention[1] is an autoregressive sequence model that builds on the Transformer[2] to consider the relative distances between different elements of the sequence rather than / along with their absolute positions in the sequence. This consideration was designed to model music, recognizing that music relies heavily on repetition to construct meaning and maintain long-term structure. The Music Transformer, in my estimation, appears to be the first neural network capable of reasonably capturing any sort of meaning to a piece. Google achieved extremely compelling results that truly demonstrate this capability of the Transformer architecture with their Piano Transformer. This essay presents my thoughts on why the Self-Attention and Relative Self-Attention algorithms are so effective.

2 Self-Attention

Self-Attention is an algorithm that is exactly what it sounds like - having a sequence pay attention to itself. The Transformer achieves this by comparing each element in the sequence with the entire sequence so as to determine the amount of "attention" each element pays to every other element, as a method of capturing internal connections and thus the overall structure of the sequence. As such, unlike convolutional or recurrent neural networks, the Transformer is able to select which portions of the input sequence to "attend to", in order to best predict future tokens.

So, how does it work? Vaswani et al. 2017 describe an attention function as, "mapping a query and a set of key-value pairs to an output"[2]. I did not learn much from that. The way I see it, attention is easier to understand by exploring the math behind it.

Let's start with a simple scenario - computing Self-Attention for a single sequence of L events in a vocabulary. At input to the Transformer, we embed the sequence (create learnable variable vector representations for each event in the vocabulary, then replace each element in the sequence with the corresponding vector) to make it of shape (L, d_{model}), where d_{model} is the embedding size or hidden dimension of the Transformer. For attention to work properly, and for residual connections to be made conveniently, most tensors being handled by the Transformer will have a dimension of length d_{model}, which is why it is so named. Next, (after adding positional encoding and scaling by a factor), we pass the embedded input X through the stack of Transformer layers. Each Transformer layer, whether in the Encoder or Decoder, consists fundamentally of an Attention block followed by a fully-connected feedforward network (FFN) block, with a LayerNorm either preceding the Attention block or succeeding the FFN block and with residual connections in between.

Now, we want to determine the attention each element pays to *every* other element in the sequence, as this will allow us to capture information about the internal connections and hence overall structure of the sequence. Let us model the attention one element of the sequence pays to another as a positive scalar whose magnitude represents the amount of "attention" paid. Assume for now that we know how to calculate this. Then, we can construct a matrix M of shape (L, L) in which each entry M_{ij} is the just mentioned numerical attention X_i pays to X_j. However, we cannot use this matrix by itself - it holds information only about the attention relations between exactly two elements in the sequence in each entry, whereas we want to extract the attention each element pays to the sequence as a whole.

Notice that the matrix product MX would in principle achieve this goal, as each $(MX)_i$ would be a weighted sum over X with the weights as the attentions X_i pays to X_j:

$$(MX)_i = \sum_j M_{ij} X_j$$

Those X_j that represent elements important to X_i will correspond to high M_{ij} and will thus contribute more to the weighted sum, whereas those X_j that represent elements unimportant to X_i will correspond to low M_{ij} and will thus contribute close to nothing. Consequently, $(MX)_i$ will be a vector in the embedding space pointing closest to the elements of X that X_i attends to the most. This weighted sum will therefore capture the connections between X_i and the entire sequence, as well as store this information in a vector of the same shape as X ($MX \sim (L, L) \times (L, d_{model}) \to (L, d_{model})$). Thus, MX is a new representation of the input sequence whose every element contains information about the internal structure of the entire sequence. Furthermore, as it preserves the shape of X, outputting this representation allows us to chain Attention blocks (with FFNs and LayerNorms in between) to extract ever purer such encapsulations of the internal structure of the sequence.

Well, it's not quite so simple. But the idea is the same.

In the actual Attention block, we don't directly multiply the attentions matrix M by X. Instead we multiply it by the *values* tensor, which is another representation of the input sequence obtained by a simple fully-connected layer on X:

$$V = XW^V + B^V$$

where W^V and B^V are parameter tensors learned by gradient descent. Of course, W^V and B^V have to preserve the shape of X for MV to do the same thing as MX, which is why W^V must be shaped (d_{model}, d_{model}), and B^V must be shaped $(d_{model},)$. Assuming the parameter tensors are properly optimized by gradient descent, this replacement of X with V in the attention calculation will determine more accurately the internal connections within X.

But how do we calculate M?

Well, since M_ij determines how much V_j contributes to the output attention, the attention calculation can be vaguely seen as something like *querying* a database of *keys*, and depending on how much the query matches with that set of keys, returning that much of the *values* corresponding to them. That is, mapping a query and a set of key-value pairs to an output. I still find it confusing to think about it this way.

That still doesn't answer the question, what does attention really mean in a computational context?

A transformation to Q would capture that ask for compatibilility.

But what does attention really mean

So, the first step in the Attention block is to transform the input sequence by fully-connected layers into the queries, keys, and values (we'll get to the values later):

$$\begin{aligned} Q &= XW^Q + B^Q \\ K &= XW^K + B^K \\ V &= XW^V + B^V \end{aligned}$$

where the parameter tensors W^i and B^i are learned by gradient descent. In order to preserve the shape of X, i.e., (L, d_{model}), each W^i must be shaped (d_{model}, d_{model}), and each B^i must be shaped $(d_{model},)$. Thus, each Q_i, K_i, and V_i is a new vector representation of X_i of shape $(d_{model},)$.

Observe that Q is of shape (L, d_{model}) and so is K. Thus, $QK^\top$ is of shape (L, L). Considering Q and K to be the representations of X we seek to compare, one can imagine that if W^Q, Q^K, B^Q, and B^K were optimized properly by gradient descent, Q and K would be such that their dot product, namely $QK^\top$, would be the attention matrix we seek, where $QK^\top_{ij}$ represents the attention X_j pays to X_i as the dot product $Q_i \cdot K_j$.

Under softmax, this sum becomes a weighted sum.

But what do we do with this now?. We need to distribute this over the sequence to sum up the importances of each element to every other element, to determine the overall structure of the sequence, the overall importance of one element to all the other elements. How do we do this? Another dot product. We could do this with just X itself, however, that is bad because we could also try to extract an even more abstract, even more pure representation of the information in X by affine transformation, which is what the fully connected layer into the values space achieves. Thus, a chain of matrix multiplication along the following lines:

$$QK^\top V$$

achieves the distribution of the information of the attention of the sequence that we want, captures the structure of the sequence in terms of how important each element is to every other element in the sequence, i.e., it stores in each new element the entire sum of information about how important that element is to the entire sequence. Furthermore, this type of representation, preserving the original shape of X allows us to chain layers of the Transformer to extract ever purer representations of that attention information.

What we need is a new representation of the sequence with this information embedded into it

V extracts even higher what X means, how X is constructed, and QK^T distributes the attention information through V to determine which values are the most important. Furthermore, this allows for chaining of the Transformer Layers.

QK^T learns the dependencies between different elements of the sequence and uses that to capture the internal structure of the sequence, to distribute over V

What we want to build is extract at an even higher level, the information about which elements in the sequence are more important in constructing the entire sequence. How can we achieve that? Precisely, by distributing the information in $QK^\top$ throughout the sequence.

As the Transformer was originally designed for language translation, in each Decoder layer of the Transformer, one of those representations would come from the input sequence in the source language, and the goal would be to build the representation of the input sequence in the target language.

The greater the match between the query and the key, the more the value corresponding to that key is unlocked. It's like a key value database

What this seeks to achieve is to determine how much a query lines up with a key, and then use that much of the value in creating the next representations of the sequence

Assuming this information about importance is contained within the sequence itself, it can theoretically be learned by backpropagation.

Attention, or more formally, Scaled Dot-Product Attention, is given by:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where Q is the queries tensor, K is the keys tensor and V is the values tensor. For those who don't know what softmax is, it's a function that can convert each vector in a tensor, or each matrix in a tensor, or the tensor itself, into a probability distribution. Look it up.

note: positional information is also important for generalizing the model to arbitrary sequence length note: model did significantly better without absolute positional encoding, with only relative attention

References

[1] C. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, C. Hawthorne, A. M. Dai, M. D. Hoffman, and D. Eck, "Music Transformer: Generating music with long-term structure," *arXiv preprint arXiv:1809.04281*, 2018.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv preprint arXiv:1706.03762*, 2017.