Notes on the Transformer

Xavier Garcia

February 2, 2019

The purpose of these notes is to explain a few of the confusing parts of the paper "Attention is all you need". We begin by discussing the simplest setup and expanding, drawing analogies between the Transformer model and our intermediate models.

1 Seq2Seq Framework

We begin by establishing the setting. We have an input sequence $\mathbf{x} = (x_1, ..., x_T)$, where each $x_i \in \mathbb{R}^{d_{\text{input}}}$ as well as a target sequence $\mathbf{y} = (y_1, ..., y_{T'})$ where we'll think of the y_i as being integer-valued, representing the index of some word in some dictionary of size d_{output}.

For simplicity, consider the simplest RNN encoder-decoder scheme where both the encoder and the decoder are vanilla RNN. The encoder consumes the sequence $\mathbf{x}$, produces a sequence of hidden states $\mathbf{h} = (h_1, ..., h_T)$ where $h_i \in \mathbb{R}^{d_{\text{hidden}}}$. We then produced a new hidden state $q_0 = q_0(\mathbf{h})$, depending on the previous the hidden states. Traditionally, $q_0 = h_T$, but we don't need to make this assumption. We use this as the new hidden state for another RNN, termed the decoder, which will autoregressively produce $\mathbf{y}$. More precisely, the input to the decoder will be the sequence given by $(\langle \text{SOS} \rangle, y_1, ...y_{T'-1})$, where the $\langle \text{SOS} \rangle$ token stands for "start of sentence" with some predetermined index in our dictionary. With this input, the decoder then produces its own set of hidden states $\mathbf{Q} = (q_1, ..., q_{T'})$, and we use this to compute the logits for our predictions. Explicitly, we have the formula:

$$\mathbb{P}(y_i = \cdot) = \text{softmax}(F(q_i))$$

where $F : \mathbb{R}^{d_{\text{hidden}}} \to \mathbb{R}^{d_{\text{output}}}$ is usually a simply fully-connected layer i.e. an affine transformation. Notice that by our choice of input for the decoder, the logits of y_i only depend on $\mathbf{h}$ and $y_1, ..., y_{i-1}$ i.e. we never peek into the future.

1.1 Attention for RNNs

In this section, I'll explain first what attention is in the RNN case, and use that as a motivation for the self-attention used in the paper.

One of the problems with the formulation stated above is that the entire essence of the input sequence $\mathbf{x}$ must be captured within q_0. While this seems reasonable for a small sequence, this could quickly become out of hand for very long sequences, especially since all such vector representations must be the same dimension, indpendent of T. To make matters worse, it may be the case that in our decoding, we may not be interested in all parts of the input sequence for any one particular decoding. For example, if we were translating from English to Spanish, it is reasonable to expect that to produce the first word in the translation, we only need the first word or two from the English sentence. This is the problem of alignment, as discussed in natural language processing. Unfortunately, it is not as simple as simply pairing the words one by one, especially since the sequences may have different lengths. The idea of attention is to endow our decoder RNN with this mechanism directly. More explicitly, to compute each y_j, we first compute an *alignment score* α_{ij} between the decoder's hidden state q_j and the encoder's hidden state h_i for each i. In particular,

$$\alpha_{ij} := \text{softmax}(\langle h_i, q_j \rangle) := \frac{e^{\langle h_i, q_j \rangle}}{\sum_{k=1} e^{\langle h_k, q_j \rangle}} \tag{1}$$

We use these alignment scores to produce an attention vector

$$A_j = \sum_{i=1}^{T} \alpha_{ij} h_i.$$

We should think that y_j is using its key q_j to "query" each of the previous x_i through their "keys" h_i by computing their inner product then normalizing the values through a softmax. For future convenience, we rewrite this equation in matrix form. We view the sequence of hidden states $\mathbf{h}$ and $\mathbf{Q} := (q_j)$ and the alignment scores α_{ij} as matrices i.e. $\mathbf{h} \in \mathbb{R}^{T \times d_{\text{hidden}}}$, $\mathbf{Q} \in \mathbb{R}^{T' \times d_{\text{hidden}}}$ and $\alpha := (\alpha_{ij}) \in \mathbb{R}^{T \times T'}$, which are connected by the following equation:

$$\alpha = \text{softmax}\left(\mathbf{h} Q^T\right).$$

In particular, $\alpha^T = \text{softmax}\left(Q\mathbf{h}^T\right)$ and $\mathbf{A} = \alpha^T \mathbf{h} = \text{softmax}\left(Q\mathbf{h}^T\right)\mathbf{h}$, where $\mathbf{A} = (A_{ij})$. We call these values the *attention values*, which are then passed by a fully-connected layer followed by a softmax to produce the probabilities for $\mathbf{y}$.

1.2 Self-Attention and Multiple Heads

To define self-attention, we first summarize the computations we performed in the previous subsections. To each element of the output, we assigned to it a "query" vector (the hidden states), which we used it to "query" other words in the input by computing the inner product with their "keys" (the hidden states), and after normalization, these became the alignment scores. We then used these alignment scores as weights to sum each element's contribution (henceforth called "values"), to obtain attention vector for that element of the output.

Upon further examination, it is clear that to define attention, we need to assign to each element of the input a “query” vector, a “key” vector and “value” vector. We group these up into matrices $\mathbf{Q} \in \mathbb{R}^{T \times d}$, $\mathbf{K} \in \mathbb{R}^{T \times d}$, and $\mathbf{V} \in \mathbb{R}^{T \times d}$ respectively,and use these to create attention values in an analogous way as before i.e.

$$\mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^T)V.$$

As a sanity check, the reader can check that if set $\mathbf{K} = \mathbf{V} = \mathbf{h}$ and $d = d_{\text{hidden}}$, then this is exactly the definition of attention described in the previous subsection.

It thus remains to devise a way to compute these matrices. By viewing $\mathbf{x}$ as a matrix $\mathbf{X} \in \mathbb{R}^{T \times d_{input}}$, we can construct these matrices by a linear transformation i.e. we define $Q = \mathbf{X}W_Q$ for some matrix W_Q and similarly for V and K. We choose right multiplication because we want to act on the columns, as that will allow us to examine every timestep at once. For example, if we chose W_Q as a block matrix, where the first m rows consisted of ones in the diagonal and zero elsewhere, then $\mathbf{Q} = \mathbf{X}W_Q$ would consist of the the m dimensions of the all features accross time.

The obvious problem is that any such restriction of features would be unable to capture global behavior. It would be fantastic if instead of simply relying one set of matrices (W_Q, W_V, W_K), we could rely on multiple, say $(W_{Q_1}, W_{V_1}, W_{K_1}), ..., (W_{Q_h}, W_{V_h}, W_{K_h})$ and combine them in a meaningful way so as to capture various of types of phenomena. This is exactly what Vaswani does with his idea of "multiple heads". Moreover, to reduce computation constraints, the matrices have scaled their dimensions by a factor of h. In particular, instead of using $W_Q \in \mathbb{R}^{d_{\text{input}} \times d_{\text{input}}}$, Vaswani uses $W_{Q_i} \in \mathbb{R}^{d_{\text{input}} \times d_{\text{input}}/h}$ for $i = 1, ..., h$ and uses this to create attention vectors $\mathbf{A}_{ij}$ then concatenates them.

In the traditional encoder-decoder scheme, only the output would query the input, not the other way around. However, in our definition of attention, we constructed both keys and queries from the input $\mathbf{x}$ with no reference to the output i.e. each word in the input is quering other words in the input. This is why its called *self*-attention. The Transformer uses this mechanism on both the input and the output.

1.3 Masked Self-Attention

In our traditional encoder-decoder scheme, it was very clear that that during the prediction of the output, we were not peeking into the future. However, due to the notion of self-attention, if we were to feed the input $(\langle\text{SOS}\rangle, y_1, ...y_{T'-1})$, we would be peeking into the future for all predictions. To remedy this, while predicting y_i, we will replace the input with $(\langle\text{SOS}\rangle, y_1, ..., y_{i-1}, \mathbf{0}, ..., \mathbf{0})$ i.e. we only keep the words that we've already seen, and replace the rest with 0. In particular, if we were to make a matrix out of our inputs, where the ith row represents the input our decoder takes while predicing y_i, we would get:

$$\begin{bmatrix} \langle\text{SOS}\rangle & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \langle\text{SOS}\rangle & y_1 & \mathbf{0} & \dots & \mathbf{0} \\ \langle\text{SOS}\rangle & y_1 & y_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \langle\text{SOS}\rangle & y_1 & y_2 & \dots & y_{T'-1} \end{bmatrix}$$

In particular, you could copy the input to the decoder $T' - 1$ times, then apply mask out the upper-triangular elements, and this would yield this matrix.