Lecture notes on attention & transformers

In sequence-to-sequence models, the goal is to learn the following model

$$p_{\boldsymbol{\theta}}(\mathbf{y}_{1:L}|\boldsymbol{x}_{1:T}) = \prod_{l=1}^{L} p_{\boldsymbol{\theta}}(\mathbf{y}_l|\mathbf{y}_{<l}, \boldsymbol{v}), \quad \boldsymbol{v} = enc(\boldsymbol{x}_{1:T}), \tag{1}$$

with $p_{\boldsymbol{\theta}}(\mathbf{y}_l|\mathbf{y}_{<l}, \boldsymbol{v})$ and $enc(\boldsymbol{x}_{1:T})$ defined by LSTMs. Although LSTMs suffers less from the gradient vanishing/explosion problems, it can still be challenging for LSTMs to learn long-term dependencies. A practical trick to boost the performance is to reverse the order of the input, i.e. using $\boldsymbol{x}_{T:1}$ instead of $\boldsymbol{x}_{1:T}$ as the input to the encoder LSTM [Sutskever et al., 2014]. Still the difficulty of learning long-term dependencies remains unsolved even with this trick. Furthermore, it is possible that different $\mathbf{y}_l$ words require different information extracted from $\boldsymbol{x}_{1:T}$, so a shared global representation $\boldsymbol{v}$ for the input sequence might be sub-optimal.

1.1 *Attention in Bahdanau et al.

Bahdanau et al. [2015] proposed an attention-based approach (the authors called it as "alignment") to address the above issues. Recall that at time t the encoder LSTM updates its internal recurrent states $\boldsymbol{c}_t^e$ and $\boldsymbol{h}_t^e$ using the current input $\boldsymbol{x}_t$:

$$\boldsymbol{h}_t^e, \boldsymbol{c}_t^e = LSTM_{\boldsymbol{\theta}}^{enc}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}^e, \boldsymbol{c}_{t-1}^e). \tag{2}$$

Similarly for the decoder LSTM, we need to maintain its internal recurrent states $\boldsymbol{c}_l^d$ and $\boldsymbol{h}_l^d$:

$$p_{\boldsymbol{\theta}}(\mathbf{y}_l|\mathbf{y}_{<l}, \boldsymbol{v}) = p_{\boldsymbol{\theta}}(\mathbf{y}_l|\boldsymbol{h}_l^d, \boldsymbol{c}_l^d), \quad \boldsymbol{h}_l^d, \boldsymbol{c}_l^d = LSTM_{\boldsymbol{\theta}}^{dec}(\mathbf{y}_{l-1}, \boldsymbol{h}_{l-1}^d, \boldsymbol{c}_{l-1}^d). \tag{3}$$

In the original sequence-to-sequence model, the global representation $\boldsymbol{v}$ is obtained by transforming the last hidden state $\boldsymbol{h}_T^e$ of the encoder LSTM, and it is used to initialise $\boldsymbol{h}_0^d$ and $\boldsymbol{c}_0^d$. Instead, Bahdanau et al. [2015] proposes using different representations of the input sequence $\boldsymbol{x}_{1:T}$ at different steps for predicting $\mathbf{y}_l$, i.e.

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{y}_{1:L}|\boldsymbol{x}_{1:T}) &= \prod_{l=1}^{L} p_{\boldsymbol{\theta}}(\mathbf{y}_l|\mathbf{y}_{<l}, \boldsymbol{v}_l), \\ p_{\boldsymbol{\theta}}(\mathbf{y}_l|\mathbf{y}_{<l}, \boldsymbol{v}_l) &= p_{\boldsymbol{\theta}}(\mathbf{y}_l|\boldsymbol{h}_l^d, \boldsymbol{c}_l^d), \quad \boldsymbol{h}_l^d, \boldsymbol{c}_l^d = LSTM_{\boldsymbol{\theta}}^{dec}([\mathbf{y}_{l-1}, \boldsymbol{v}_l], \boldsymbol{h}_{l-1}^d, \boldsymbol{c}_{l-1}^d). \end{aligned} \tag{4}$$

where the representation $\boldsymbol{v}_l$ is obtained as follows:

$$\boldsymbol{f}_t = T_{\boldsymbol{\theta}}(\boldsymbol{h}_t^e), \quad \text{(feature output of the encoder at time } t\text{)} \tag{5}$$

$$e_{lt} = a(\boldsymbol{h}_{l-1}^d, \boldsymbol{f}_t), \quad \text{(compute similarity/alignment score)} \tag{6}$$

$$\alpha_l = softmax(\boldsymbol{e}_l), \quad \boldsymbol{e}_l = (e_{l1}, ..., e_{lT}), \tag{7}$$

$$\boldsymbol{v}_l = \sum_{t=1}^{T} \alpha_{lt} \boldsymbol{f}_t. \quad \text{(weighted aggregation of input features)} \tag{8}$$

The key idea of using $\boldsymbol{v}_l$ as a weighted average of individual features $\boldsymbol{f}_t$ is to allow the decoder LSTM to directly access the representation for each input $\boldsymbol{x}_t$, therefore the issue of lacking long-term dependencies between $\mathbf{y}_l$ and $\boldsymbol{x}_t$ is addressed. This is in contrast with the global representation vector $\boldsymbol{v}$ in the original Seq2Seq model: since $\boldsymbol{v}$ is computed using the last recurrent state of the encoder LSTM, it is questionable whether $\boldsymbol{v}$ can capture long-term dependencies within $\boldsymbol{x}_{1:T}$. Another notable difference is that in Bahdanau et al. [2015] the features $\boldsymbol{v}_l$ are used as the input to the decoder LSTM (together with $\mathbf{y}_{l-1}$), while in the original Seq2Seq model the global representation $\boldsymbol{v}$ is used to initialise the decoder LSTM's recurrent states.

The alignment score $e_{lt} = a(\boldsymbol{h}_{l-1}^d, \boldsymbol{f}_t)$ is computed between $\boldsymbol{h}_{l-1}^d$ (which summarises $\mathbf{y}_{<l}$, required for the auto-regressive model to predict $\mathbf{y}_l$) and $\boldsymbol{f}_t$ (a feature representation for $\boldsymbol{x}_t$ but is also dependant on $\boldsymbol{x}_{<t}$). These scores are then passed through a softmax layer to obtain the *attention weight* α_l, with larger attention weight value α_{lt} the final representation $\boldsymbol{v}_l$ will focus more on the input feature $\mathbf{f}_t$ for $\boldsymbol{x}_t$.

1.2 Attention in transformers

Single-head attention

The *scaled dot product attention* method is introduced by Vaswani et al. [2017]. Intuitively it can be understood from an information retrieval point of view. Queries are submitted to the system which are represented by the *query vectors* $\boldsymbol{q}_i \in \mathbb{R}^{d_q}$, and the system will first check the matching/alignment/"similarity" between the query and the keys (which are represented by *key vectors* $\boldsymbol{k}_j \in \mathbb{R}^{d_q}$), then return the retrieved values to the user. Each key vector $\boldsymbol{k}_j$ is associated with a *value vector* $\boldsymbol{v}_j \in \mathbb{R}^{d_v}$, so that if $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$ are aligned, the value vector $\boldsymbol{v}_j$ will be returned in some form.

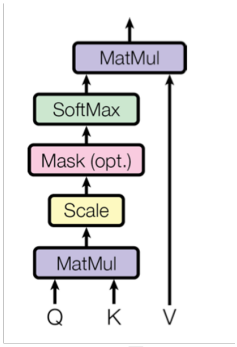
In detail, the mathematical form for single-head attention is the following:

$$Attention(Q, K, V; a) = a\left(\frac{QK^\top}{\sqrt{d_q}}\right)V \tag{9}$$

$$Q = (\boldsymbol{q}_1, ..., \boldsymbol{q}_N)^\top \in \mathbb{R}^{N \times d_q} \tag{10}$$

$$K = (\boldsymbol{k}_1, ..., \boldsymbol{k}_M)^\top \in \mathbb{R}^{M \times d_q} \tag{11}$$

$$V = (\boldsymbol{v}_1, ..., \boldsymbol{v}_M)^\top \in \mathbb{R}^{M \times d_v} \tag{12}$$



There are two important ingredients in the scaled dot product attention process. First, an *attention matrix* $A = a\left(\frac{QK^\top}{\sqrt{d_q}}\right)$ is computed to indicate the alignment of each query vector $\boldsymbol{q}_i$ to the key vectors $\boldsymbol{k}_j$. Then given the attention matrix A, the attention output for each query vector $\boldsymbol{q}_i$ is the weighted sum of the value vectors $\sum_{j=1}^M A_{ij}\boldsymbol{v}_j$, which is similar to the final output of the attention method by Bahdanau et al. Here $a(\cdot)$ is an activation function applied row-wise, and in *soft attention*, $a(\cdot)$ is the softmax function, i.e. $A_{ij} = softmax((\langle \boldsymbol{q}_i, \boldsymbol{k}_1\rangle, ..., \langle \boldsymbol{q}_i, \boldsymbol{k}_M\rangle)/\sqrt{d_q})$. This is in contrast with *hard attention* where $a(\cdot)$ returns a one-hot vector for each row with the $j^* = \arg\max_j \langle \boldsymbol{q}_i, \boldsymbol{k}_j\rangle$ element equals to 1. Hard attention can be interpreted exactly as an information retrieval system since it returns the corresponding value for the best key match to the query.

When the dimensionality of the query/key vector d_q is large, the dot product $\langle \boldsymbol{q}_i, \boldsymbol{k}_j\rangle$ can be large as well. This is likely to increase the gap between the dot product values so that the softmax output could be dominated by a single entry (i.e. close to hard attention). Normalisation of the logits $\langle \boldsymbol{q}_i, \boldsymbol{k}_j\rangle$ could be useful to address this issue. For the specific choice of $\sqrt{d_q}$, this comes from the assumption that the elements in $\boldsymbol{q}_i$ and $\boldsymbol{k}_j$ are independently distributed with variance 1. If so then the variance of $\langle \boldsymbol{q}_i, \boldsymbol{k}_j\rangle$ is d_q, so by normalising the dot product with $\sqrt{d_q}$, the logit will have its variance equal to 1.

In some cases, *masking* is applied to the attention procedure. For example, a typical masking strategy will define a mask matrix $M \in \{0, 1\}^{N \times M}$, so that if $M_{ij} = 0$, then the corresponding attention weight $A_{ij} = 0$, so that $\boldsymbol{v}_j$ does not contribute to the final output for query $\boldsymbol{q}_i$.

Self-attention is also in wide usage which sets $K = Q$.

Complexity figures

The time complexity for scaled dot-product attention is $\mathcal{O}(MNd_q + MNd_v)$. These include the dot product $QK^\top$ which has $\mathcal{O}(MNd_q)$ run-time cost, the computation of A matrix given the dot product which has $\mathcal{O}(MN)$ cost, and the dot product AV which has $\mathcal{O}(MNd_v)$ cost.

The space complexity for the scaled dot-product is $\mathcal{O}(MN+Nd_v)$ since back-propagation requires storing some intermediate results. These include the dot product $QK^\top$ which has $\mathcal{O}(MN)$ memory cost and the final output AV which has $\mathcal{O}(Nd_v)$ cost.

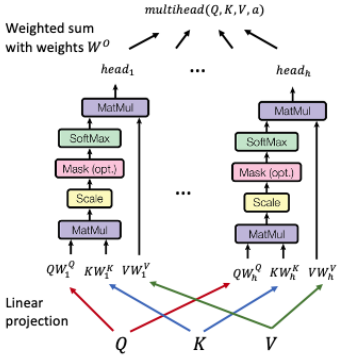
For self-attention, since $Q = K$, then the time complexity and space complexity figures are $\mathcal{O}(N^2 d_q + N^2 d_v)$ and $\mathcal{O}(N^2 + Nd_v)$, respectively.

Multi-head attention

Multi-head attention repeats the single-head attention process with different "views". Intuitively, consider information retrieval with the input query representing "cat". Then depending on the definition of similarity, it can match to key "tiger" or key "pet" and so on.

Multi-head attention allows the definition of multiple alignment processes, by projecting the inputs into different sub-spaces then performing dot product attention in such sub-spaces. The outputs of each attention head are concatenated and projected to produce the final output.

$$\begin{aligned} Multihead(Q, K, V; a) &= concat(head_1, ..., head_h)W^O, \\ head_i(Q, K, V; a) &= Attention(QW_i^Q, KW_i^K, VW_i^V; a). \end{aligned} \tag{13}$$



It is clear that the time and space complexity figures of multi-head attention are h times of those for a single head plus the extra costs for linear projections. Assume the projection matrices have sizes $W_i^Q \in \mathbb{R}^{d_q \times \tilde{d}_q}$, $W_i^K \in \mathbb{R}^{d_q \times \tilde{d}_q}$, $W_i^V \in \mathbb{R}^{d_v \times \tilde{d}_v}$ and $W^O \in \mathbb{R}^{h\tilde{d}_v \times d_{out}}$. This means

$$\text{time complexity:} \quad \mathcal{O}(\underbrace{h(MN\tilde{d}_q + MN\tilde{d}_v)}_{\text{attention heads}} + \underbrace{h(\tilde{d}_q d_q(M+N) + \tilde{d}_v d_v M)}_{\text{input projections}} + \underbrace{Nh\tilde{d}_v d_{out}}_{\text{combined outputs}}),$$

$$\text{space complexity:} \quad \mathcal{O}(\underbrace{h(MN + N\tilde{d}_v)}_{\text{attention heads}} + \underbrace{h(\tilde{d}_q(M+N) + \tilde{d}_v M)}_{\text{input projections}} + \underbrace{Nd_{out}}_{\text{combined outputs}}).$$

To keep the costs close to performing single-head attention in the original space, often the $\tilde{d}_q$, $\tilde{d}_v$ dimensions are set to be $\tilde{d}_q = \lfloor \frac{d_q}{h} \rfloor$ and $\tilde{d}_v = \lfloor \frac{d_v}{h} \rfloor$, respectively.

1.3 Ingredients in transformers

Transformer (proposed in Vaswani et al. [2017]) is an encoder-decoder type of architecture, which is visualised in Figure 1. We discuss some of the key ingredients as follows.

Position encoding

From the equations of scaled dot-product attention, we see that attention is equivariant to row permutations in the query matrix Q. To see this, notice that permuting rows in a matrix is equivalent to left multiplying a permutation matrix P to Q. Since the non-linearity $a(\cdot)$ is applied row-wise, this leads to the permutation equivariance result:

$$Attention(PQ, K, V; a) = a\left(\frac{PQK^\top}{\sqrt{d_q}}\right)V = Pa\left(\frac{QK^\top}{\sqrt{d_q}}\right)V = PAttention(Q, K, V; a). \tag{14}$$

Therefore, for an input sequence of queries $\boldsymbol{q}_1, ..., \boldsymbol{q}_N$ in which the ordering information (i.e. the subscript n) is useful for the task (e.g. time series regression), these ordering information needs to

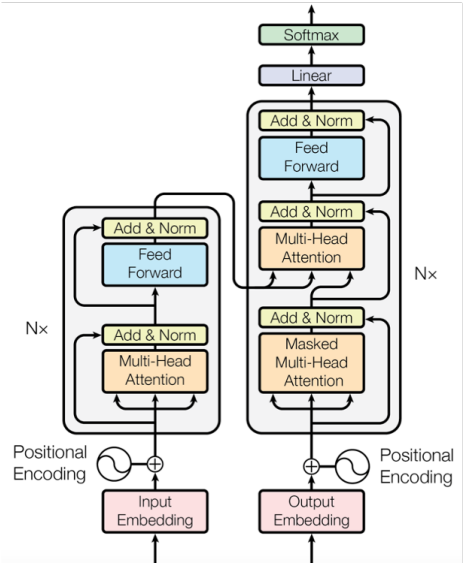


Figure 1: The Transformer architecture in Vaswani et al. [2017].

be added in some form to the attention inputs. This issue is addressed by position encoding, which constructs the query input $\tilde{Q}$ to the attention module as:

$$\tilde{Q} = (\tilde{\boldsymbol{q}}_1, ..., \tilde{\boldsymbol{q}}_N)^\top, \quad \tilde{q}_n = f(\boldsymbol{q}_n, PE(n)), \tag{15}$$

where the f transformation is often set to be simple operations such as summation and concatenation, etc. $PE(n)$ is called the *position encoding* of input index n, which can either be learned (e.g. having a set of learnable parameters $\{\boldsymbol{p}_1, ..., \boldsymbol{p}_n\}$ or using an NN-parameterised function), or computed using a pre-defined function. Learned embedding of the form $\{\boldsymbol{p}_1, ..., \boldsymbol{p}_n\}$ is well suited if the maximum value of the index is known. Otherwise, in the case where unseen index can occur in test time, a pre-defined function might be preferred.

A popular choice of such pre-defined functions is the *sinusoid embedding* [Vaswani et al., 2017]:

$$\begin{aligned} PE(n) &= (PE(n,0), ..., PE(n,2I)), \\ PE(n,2i) &= sin(n/10000^{2i/d_q}), \\ PE(n,2i+1) &= cos(n/10000^{2i/d_q}). \end{aligned} \tag{16}$$

We see that sinusoid embeddings use multiple periodic functions to embed the input index n, where the frequency of each sine/cosine wave is determined by i. The authors of Vaswani et al. [2017] did not give a formal justification of this approach and instead stated that "we hypothesized it would allow the model to easily learn to attend by relative positions". My personal hypothesis is that, in the case of using concatenation to construct the new query (i.e. $\tilde{q}_n = f(\boldsymbol{q}_n, PE(n)) = [\boldsymbol{q}_n, PE(n)]$) which is then multiplied by learnable weight matrices, the sinusoid embedding allows the network to learn a very flexible position encoding function as a combination of many sine/cosine waves of different frequencies (think about the Fourier series of a continuous function).

Layer normalisation

Layer normalisation [Ba et al., 2016] is a normalisation technique used to stabilise neural network training. Assuming a feed-forward layer with the pre-activation computed as $\boldsymbol{a} = (a_1, ..., a_H) = W\boldsymbol{x} + \boldsymbol{b}$, layer normalisation can be applied to the pre-activation vector before feeding to the non-

linearity:

$$\boldsymbol{a} \leftarrow \frac{\boldsymbol{a} - \mu}{\sigma}, \quad \mu = \frac{1}{H}\sum_{h=1}^{H} a_h, \quad \sigma = \sqrt{\frac{1}{H}\sum_{h=1}^{H}(a_h - \mu)^2}. \tag{17}$$

In transformers, layer normalisation is applied together with a residual link: $Add\&Norm(x) = LayerNorm(x + Sublayer(x))$, where $Sublayer(\cdot)$ can either be a multi-head attention block or a point-wise feed-forward network.

Point-wise feed-forward network

The point-wise feed-forward network is used as the "feed forward" layer in Transformer's architecture (see Figure 1). The multi-head attention (after "$Add\&Norm$") returns a matrix of size $N \times d_{out}$, representing the attention results for N queries. These output can be processed "point-wise" (i.e. row-wise) with a feed-forward neural network, which effectively treat the rows in the attention outputs as "datapoint" inputs for the next layer.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27:3104–3112.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.