Concept Notes: Transformer architecture

Luis Oala

October 15, 2018

Materials used

These notes are based on the following materials

- ...

Attention is a popular computational module in many successful neural language models but also finds application in vision tasks such as visual question answering. The purpose of this short concept note is to concisely explain the mechanics of a model architecture called Transformer which solely uses attention. This note solely focuses on the mechanics of the Transformer model. The goal is to make clear what computations are actually carried out and how the model works. I do not discuss the conceptual merits of attention per se or why it is useful for sequence modelling tasks.

The notes start with a short exposition of attention as 'classically' used in neural sequence-to-sequence models based on LSTM cells. Then the Transformer's main mechanism, as introduced by [Vaswani et al., 2017], is explained in detail. Finally, a short mention is made of the Universal Transformer idea by [Dehghani et al., 2018], who address a practical and theoretical shortcoming of the original architecture.

1 Attention as we know it

Figure 1 shows the sketch of a vanilla neural sequence-to-sequence model based on LSTM cells which utilizes an attention mechanism before the ouput layer of the decoder. We can make note of the following variables:

- $\mathbf{h}_i^{\mathrm{e}}$: the hidden state of the encoder LSTM cell at input sequence position i
- $\mathbf{h}_i^{\mathrm{d}}$: the hidden state of the decoder LSTM cell at output sequence position i
- $\mathbf{H}^{\mathrm{e}} = concat(\mathbf{h}_1^{\mathrm{e}}, \ldots, \mathbf{h}_k^{\mathrm{e}})$: a matrix of all the hidden states of the encoder LSTM cell
- $\mathbf{H}^{\mathrm{d}} = concat(\mathbf{h}_1^{\mathrm{d}}, \ldots, \mathbf{h}_l^{\mathrm{d}})$: a matrix of all the hidden states of the decoder LSTM cell
- θ: the trainable parameters of the attention function, varies depending on the design of the attention function
- $\mathbf{h}_i^{\mathrm{d}^*}$: the decoder attention state at output sequence position i

The attention function then takes as input the current hidden state of the decoder $\mathbf{h}_i^{\mathrm{d}}$, in the attention context also called *query*, as well as a matrix $\mathbf{H}$ of other states for which we desire to calculate the attention distribution. $\mathbf{H}$ is also referred to as *key* and *value*. The parameters θ

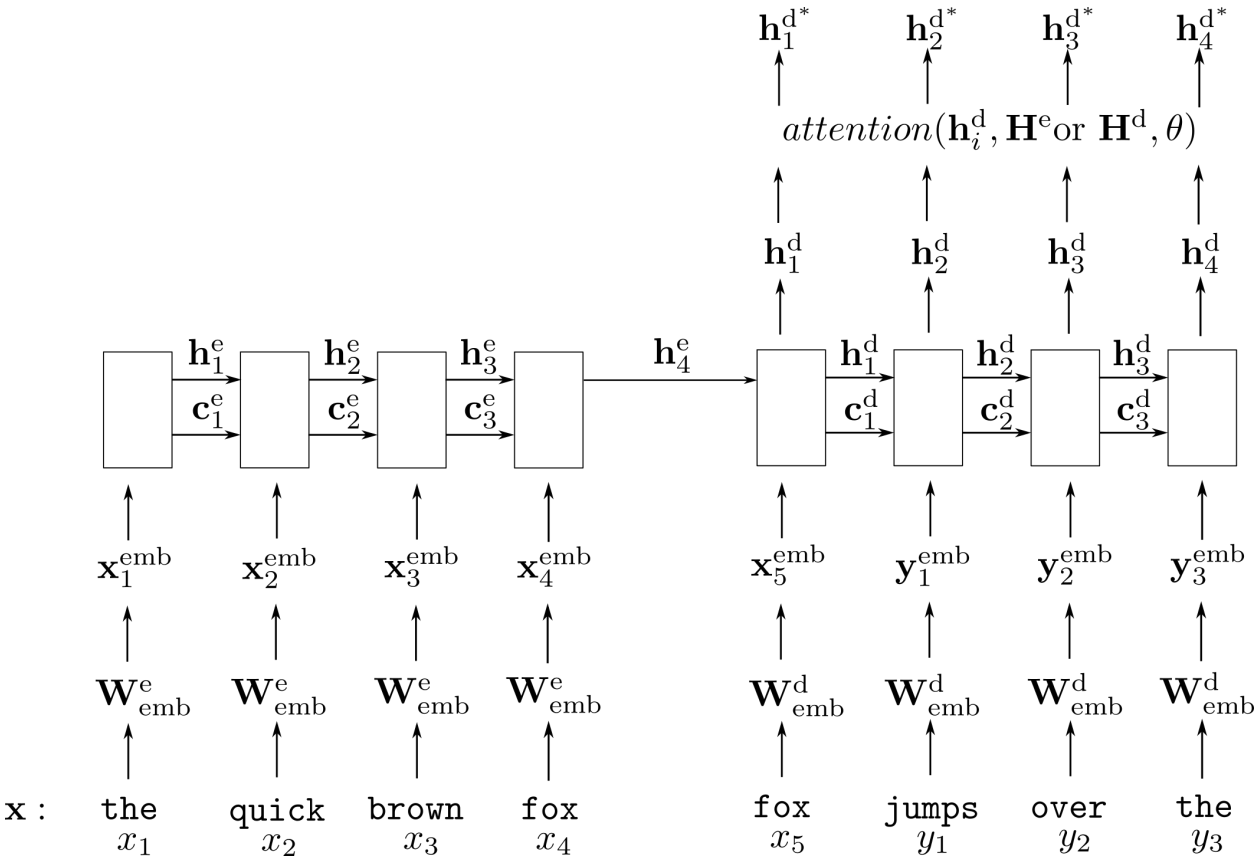

Figure 1: Illustrated example of a vanilla encoder-decoder sequence-to-sequence architecture with an attention mechanism. The ouput mechanics of the decoder after $\mathbf{h}_i^{d^*}$ are omitted for brevity.

are the final inputs to the attention function. These parameters are learned to train the model's attention mechanism.

The attention function $attention(\mathbf{h}_i^d, \mathbf{H}^e \text{or } \mathbf{H}^d, \theta)$ usually contains the following steps and components:

(a) Calculate *attention scores* $\mathbf{e}_i$,

(b) normalize the *attention scores* to an *attention distribution* $\mathbf{a}_i$ via the softmax function

(c) and finally combine the attention values, $\mathbf{H}$, into an attention weighted representation $\mathbf{h}_i^{d^*}$ using $\mathbf{a}_i$ as weights.

Let us look at a concrete example for these steps taken from [See et al., 2017]. In step (a) $\mathbf{e}_i$ is calculated as

$$\mathbf{e}_i = \mathbf{v}^T \tanh(\mathbf{W}_1\mathbf{H} + \mathbf{W}_2\mathbf{h}_i^d + \mathbf{b}_{\text{attn}}) \tag{1}$$

Note that i indicates the decoder time step in this context and for this particular kind of attention function we have $\theta = (\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_{\text{attn}}, \mathbf{v})$. Thus we now have a vector $\mathbf{e}_i$ of *attention scores*. Each position in $\mathbf{e}_i$ corresponds to a time step of the encoder LSTM cell. Next for step (b) these values

are normalized via the softmax function to yield an *attention distribution*:

$$\mathbf{a}_i = \text{softmax}(\mathbf{e}_i) \tag{2}$$

Finally, in step (c) the attention scores are used to produce the *context vector* $\mathbf{h}_i^{\mathrm{d}^*}$:

$$\mathbf{h}_i^{\mathrm{d}^*} = \mathbf{H}\mathbf{a}_i \tag{3}$$

Now we have worked through a concrete example for an attention function $attention(\mathbf{h}_i^{\mathrm{d}}, \mathbf{H}^{\mathrm{e}} \text{or } \mathbf{H}^{\mathrm{d}}, \theta)$. Note that there many variants to design the attention function[1]. Most variants only differ in the way $\mathbf{e}_i$ is calculated in Equation 1. The basic steps remain the same.

After we have obtained the *context vector* $\mathbf{h}_i^{\mathrm{d}^*}$ it is then usually concatenated to the regular decoder output state $\mathbf{h}_i^{\mathrm{d}}$. This concatenated vector is then typically passed as input to the output layer of the decoder which in sequence modelling tasks usually produces a distribution over words.

Finally a short comment on the terminology regarding attention that is used in the literature. As you may have noted we have denoted $attention(\mathbf{h}_i^{\mathrm{d}}, \mathbf{H}^{\mathrm{e}} \text{or } \mathbf{H}^{\mathrm{d}}, \theta)$ with an *or*. The reason for this is that in encoder-decoder architectures a terminological distinction is drawn between:

- *Intra-temporal attention*: This attention mechanism uses $\mathbf{H}^{\mathrm{e}}$, i.e. the encoder LSTM hidden states, as input.
- *Intra-decoder attention* or more generally *self-attention*: In this scenario the attention function takes as input $\mathbf{H}^{\mathrm{d}}$ which is a matrix of the LSTM hidden states of the *decoder* up to the current decoder position at which we calculate the attention.

Now that we have recapped the mechanics of the attention module as used in vanilla sequence-to-sequence models let us proceed to an understanding how attention is used in the Transformer model by [Vaswani et al., 2017].

2 Attention as the Transformer uses it

References

[Dehghani et al., 2018] Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. (2018). Universal Transformers. *arXiv preprint arXiv:1807.03819*.

[See et al., 2017] See, A., Liu, P. J., and Manning, C. D. (2017). Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, \., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008. read.

[1] See the slides of lecture 11 of Richard Socher's Stanford course *cs224n* for an overview of different variants `http://web.stanford.edu/class/cs224n/lectures/lecture11.pdf`