A Brief Overview of Attention Mechanism

Synced Follow
Sep 25, 2017 · 5 min read

What is Attention?

Attention is simply a vector, often the outputs of dense layer using softmax function.

Before Attention mechanism, translation relies on reading a complete sentence and compress all information into a fixed-length vector, as you can image, a sentence with hundreds of words represented by several words will surely lead to information loss, inadequate translation, etc.

However, attention partially fixes this problem. It allows machine translator to look over all the information the original sentence holds, then generate the proper word according to current word it works on and the context. It can even allow translator to zoom in or out (focus on local or global features).

Attention is not mysterious or complex. It is just an interface formulated by parameters and delicate math. You could plug it anywhere you find it suitable, and potentially, the result may be enhanced.

Why Attention?

The core of Probabilistic Language Model is to assign a probability to a sentence by Markov Assumption. Due to the nature of sentences that consist of different numbers of words, RNN is naturally introduced to model the conditional probability among words.

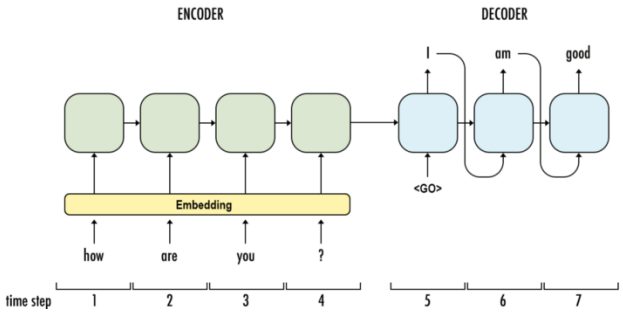
$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i \mid w_{i-k} \dots w_{i-1})$$

Vanilla RNN (the classic one) often gets trapped when modeling:

1. Structure Dilemma: in real world, the length of outputs and inputs can be totally different, while Vanilla RNN can only handle fixed-length problem which is difficult for the alignment. Consider an EN-FR translation examples: “he doesn’t like apples” → “Il n’aime pas les pommes”.
2. Mathematical Nature: it suffers from Gradient

Vanishing/Exploding which means it is hard to train when sentences are long enough (maybe at most 4 words).

Translation often requires arbitrary input length and out put length, to deal with the deficits above, encoder-decoder model is adopted and basic RNN cell is changed to GRU or LSTM cell, hyperbolic tangent activation is replaced by ReLU. We use GRU cell here.

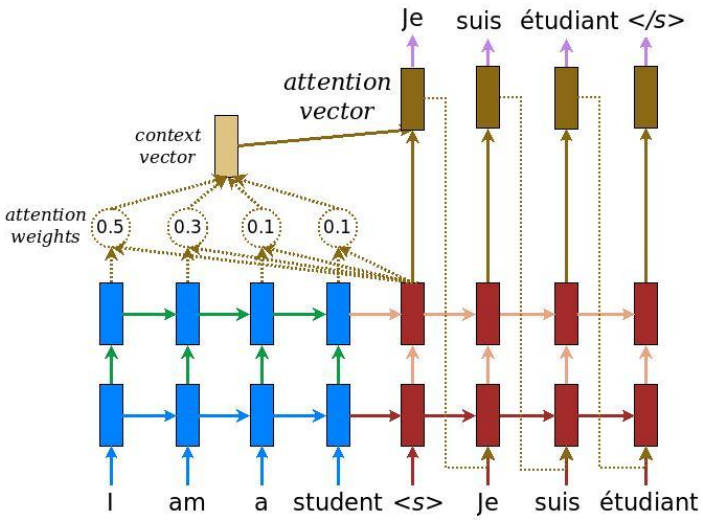


Embedding layer maps discrete words into dense vectors for computational efficiency. Then embedded word vectors are fed into encoder, aka GRU cells sequentially. What happened during encoding? Information flows from left to right and each word vector is learned according to not only current input but also all previous words. When the sentence is completely read, encoder generates an output and a hidden state at timestep 4 for further processing. For encoding part, decoder (GRUs as well) grabs the hidden state from encoder, trained by teacher forcing (a mode that previous cell's output as current input), then generate translation words sequentially.

It seems amazing as this model can be applied to N-to-M sequence, yet there still is one main deficit left unsolved: is one hidden state really enough?

Yes, Attention here.

How does attention work?



Similar to the basic encoder-decoder architecture, this fancy mechanism plug a context vector into the gap between encoder and decoder. According to the schematic above, blue represents encoder and red represents decoder; and we could see that context vector takes all cells' outputs as input to compute the probability distribution of source language words for each single word decoder wants to generate. By utilizing this mechanism, it is possible for decoder to capture somewhat global information rather than solely to infer based on one hidden state.

And to build context vector is fairly simple. For a fixed target word, first, we loop over all encoders' states to compare target and source states to generate scores for each state in encoders. Then we could use softmax to normalize all scores, which generates the probability distribution conditioned on target states. At last, the weights are introduced to make context vector easy to train. That's it. Math is shown below:

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

To understand the seemingly complicated math, we need to keep three key points in mind:

> During decoding, context vectors are computed for every output word. So we will have a 2D matrix whose size is # of target words multiplied by # of source words. Equation (1) demonstrates how to compute a single value given one target word and a set of source word.
>
> Once context vector is computed, attention vector could be computed by context vector, target word, and attention function `f`.
>
> We need attention mechanism to be trainable. According to

equation (4), both styles offer the trainable weights (W in Luong's, W1 and W2 in Bahdanau's). Thus, different styles may result in different performance.

Conclusion

We hope you understand the reason why attention is one of the hottest topics today, and most importantly, the basic math behind attention. Implementing your own attention layer is encouraged. There are many variants in the cutting-edge researches, and they basically differ in the choice of score function and attention function, or of soft attention and hard attention (whether differentiable). But basic concepts are all the same. If interested, you could check out papers below.

[1] Vinyals, Oriol, et al. Show and tell: A neural image caption generator. *arXiv:1411.4555* (2014).
[2] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473* (2014).
[3] Cho, Kyunghyun, Aaron Courville, and Yoshua Bengio. Describing Multimedia Content using Attention-based Encoder–Decoder Networks. arXiv:1507.01053 (2015)
[4] Xu, Kelvin, et al. Show, attend and tell: Neural image caption generation with visual attention. *arXiv:1502.03044* (2015).
[5] Sukhbaatar, Sainbayar, Jason Weston, and Rob Fergus. End-to-end memory networks. *Advances in Neural Information Processing Systems*. (2015).
[6] Joulin, Armand, and Tomas Mikolov. Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets. *arXiv:1503.01007* (2015).
[7] Hermann, Karl Moritz, et al. Teaching machines to read and comprehend. *Advances in Neural Information Processing Systems*. (2015).
[8] Raffel, Colin, and Daniel PW Ellis. Feed-Forward Networks with Attention Can Solve Some Long-Term Memory Problems. *arXiv:1512.08756* (2015).
[9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., & Gomez, A. et al. . *Attention Is All You Need*. *arXiv: 1706.03762* (2017).

. . .

Tech Analyst: **Qingtong Wu**