Stanford CS224N NLP with Deep Learning

Winter 2021

Lecture 8 – Attention Revisited

Attention

The basic idea of attention, as we have seen, is to allow the decoder to learn which part of the input it needs to focus on. More concretely, each encoder hidden state is given an *attention score*, and the hidden states are averaged according to the softmax of all these scores. This averaged vector is used (along with the decoder hidden state) to predict the next word of the output.

Formally, suppose we have the encoder hidden states $h_1, \ldots, h_N$, and the decoder hidden state s_t at timestep t.
We get the attention scores for this step using dot products:

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N],$$

and convert this into a probability distribution using softmax $\$\alpha^t = \text{softmax}(e^t)$.

We use this distribution to take a weighted sum of the encoder hidden states

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i,$$

which gives the attention output. This is concatenated with s_t and used to predict the next output word.

Attention has a number of advantages: it solves the bottleneck and the vanishing gradient problems, and it provides some interpretability (inspecting the attention scores gives us the alignment between the sentences).

Variants of Attention

All variants of attention involve three basic steps: computing the attention scores e, getting the distribution α, and taking the weighted sum a. The chief variation among methods of implementing attention lies in the first step.

The simplest way is to use a dot product between the hidden states of the decoder and the encoder:

$$e_i = s^T h_i.$$

Another common way is *multiplicative attention*, where we use a weight matrix:

$$e_i = s^T W h_i.$$

Reduced-rank multiplicative attention uses low-rank matrices $U \in \mathbb{R}^{k \times d_1}$, $V \in \mathbb{R}^{k \times d_2}$:

$$e_i = s^T (U^T V) h_i = (Us)^T (V h_i).$$

Finally, *additive attention* uses a weighted sum of the two vectors, followed by a nonlinearity:

$$e_i = v^T \tanh(W_1 h_i + W_2 s).$$