Bahdanau attention

Posted on November 14, 2017 by Praveen Narayanan

In an earlier post, I had written about seq2seq without attention by way of introducing the idea. This time, we extend upon that by adding attention to the setup. In the regular seq2seq model, we embed our input sequence $x = x_1, x_2, ..., x_T$ into a context vector c, which is then used to make predictions. In the attention variant, the context vector c is replaced by a customized context c_i for the hidden decoder vector s_{i-1}. The result is the summed over contribution over all of the input hidden vectors. Attention is important for the model to generalize well to test data, in that our model might learn to minimize the cost function during train time, but it is only when it learns attention that we know that it has an idea that it knows exactly where to look (and put that knowledge into the context) for it to generalize well to test data.

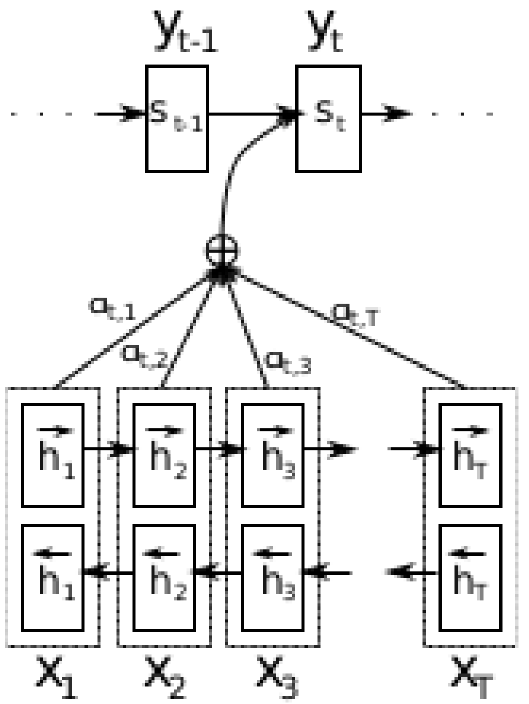


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

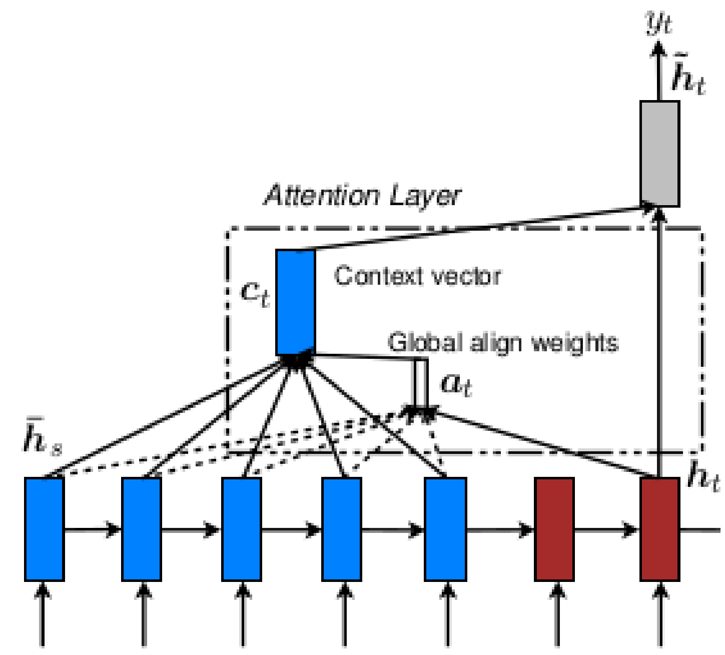


Figure 2: **Global attentional model** – at each ti
step t, the model infers a *variable-length* ali
ment weight vector a_t based on the current tar
state h_t and all source states $\bar{h}_s$. A global cont
vector c_t is then computed as the weighted av
age, according to a_t, over all the source states.

$c_i = \sum_j \alpha_{ij} h_j$

This operation computes how we weight the input hidden vectors h_j (this could be bidirectional, in which case we concatenate the forward and backward hidden states). Naturally, if the input and output hidden states are 'aligned' then α_{ij} would be quite high for those states. In practice, more than one input word could be aligned with their output counterparts. For example, for some words in English, we will have a direct correspondences in French (de == of; le, la == the), but some words can have multiple correspondences (je ne suis pas == I am not), so the alignment should register these features. Here we know that (je/i) form a pair; (suis/am) form another pair, but we also know that (ne suis pas/am not) should occur together, and they will have non-zero alphas when we group them together, not to mention the difference in sequence lengths.

The attention/alignment parameters α_{ij} are computed as a non-linear function of the hidden units, yielding an attention parameter a_{ij} which is then softmaxed to make it lie between 0 and 1.

$$a_{ij} = f(s_{i-1}, h_j) = v' \tanh(W_1 s_{i-1} + W_2 h_j + b)$$

$$\alpha_{ij} = softmax(a_{ij}) = \frac{\exp(a_{ij})}{\sum_j \exp(a_{ij})}$$

Once we compute the context c_i we can make it produce predictions for the output hidden states

$$s_i = F(s_{i-1}, c_i, y_i)$$

Alternative forms

Since attention/alignment is essentially a similarity measure between a decoder and encoder hidden vector, we can invoke dot products to compute it.

$$a_{ij} = F(s_{i-1}, h_j) =$$

where $latex $ is the dot product between vectors a, b.

A development on this idea (Luong's multiplicative attention) is to transform the vector before doing the dot product.

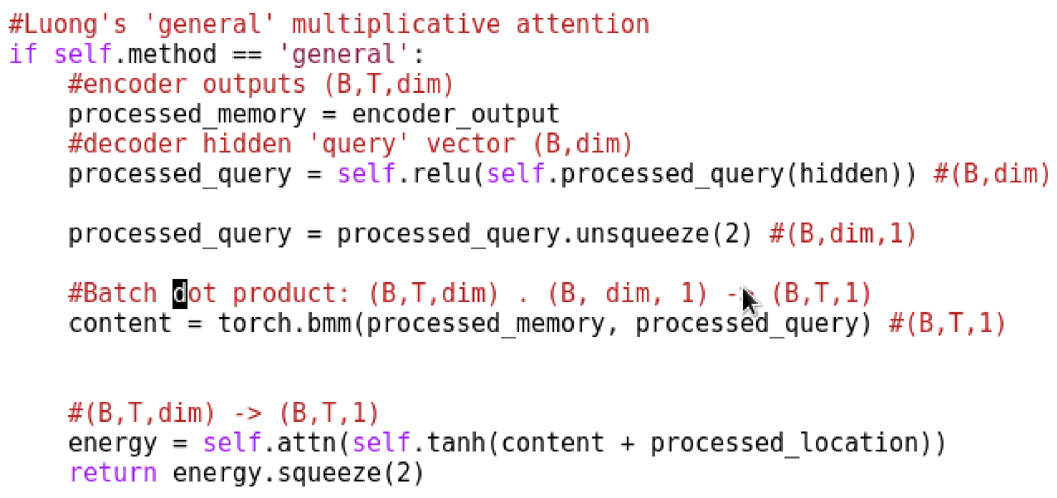
$$a_{ij} =$$

The form $F s_{i-1}$ indicates that we can apply a linear transformation to the decoder hidden unit without a bias term and then take dot product (which in torch would be through torch.bmm() for batched quantities).

[From Luong's paper]

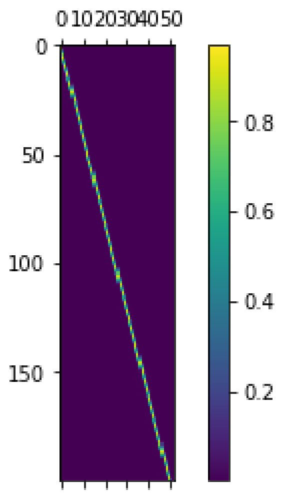
$$\mathrm{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

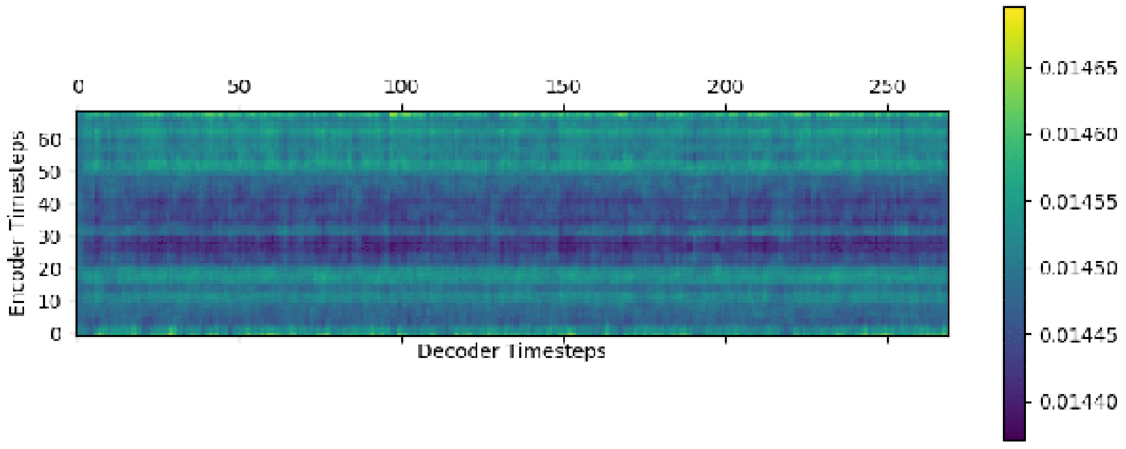
-



```
        #Luong's 'general' multiplicative attention
        if self.method == 'general':
            #encoder outputs (B,T,dim)
            processed_memory = encoder_output
            #decoder hidden 'query' vector (B,dim)
            processed_query = self.relu(self.processed_query(hidden)) #(B,dim)

            processed_query = processed_query.unsqueeze(2) #(B,dim,1)

            #Batch dot product: (B,T,dim) . (B, dim, 1) -> (B,T,1)
            content = torch.bmm(processed_memory, processed_query) #(B,T,1)


            #(B,T,dim) -> (B,T,1)
            energy = self.attn(self.tanh(content + processed_location))
            return energy.squeeze(2)
```

Multiplicative attention with components for location and content.





Attention curve evolution from a voice conversion model. Here, the encoder maps input speech (mels) to a context, and the decoder inflates this context to output speech representation [9].

Computing the hidden decoder state

As we can make out from the equations above, we would like to formulate the decoder state as an RNN. One way of doing that is to concatenate the input with the context and compute the next hidden state.

$$s_i = RNN(s_{i-1}, [c_i, y_i])$$

For example, in the Tacotron code, we have a multilayer decoder stack, with the first layer being the so-called attention RNN which is exactly what we have above. But there area two layers after this with residual connections.

$$s_i^1 = RNN^1(s_i^1, s_i) + s_i$$

$$s_i^2 = RNN^2(s_i^2, s_i^1) + s_i^1$$

Annotations and bidirectionality

The Bahdanau paper uses a bidirectional RNN for the encoder. This computes hidden units for the sequence with the normal ordering (left to right) and reversed ordering (right to left) (see CS224D notes by Richard Socher).

$$\overrightarrow{h}_i = f(\overrightarrow{W} x_i + \overleftarrow{V} h_{i-1} + \overrightarrow{b})$$
$$\overleftarrow{h}_i = f(\overleftarrow{W} x_i + \overleftarrow{V} h_{i+1} + \overleftarrow{b})$$

The so called ‘annotations’ h are a concatenation of the forward and backward hidden vectors which are then used to compute context vectors.