Attention Model

1. Introduction

Before attention model, machine translation relies on an encoder-decoder architecture. The encoder reads in the whole sentence and compress it into a fixed-length latent representation. The decoder outputs the translated sentence based on the latent representation. The drawback of this model is significant:

i. Sentence Length: sentences tend to have different length and different types of information stored, it is unlikely for then to share the same latent representation

ii. Gradient: it suffers from vanishing / exploding gradient which means it is hard to train when sentences are long enough.

In real practice, the encoder-decoder architecture works quite well for short sentences, but the performance comes down for very long sentences (e.g. longer than 30 or 40 words) leading to information loss and hence inadequate translation, etc.

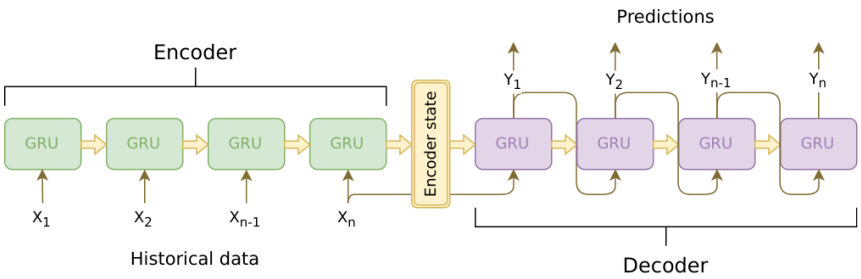

Figure 1. Encoder & Decoder Network

- **Word Embedding**

The vanilla encoder-decoder model can be improved by adding an embedding layer before encoder. The embedding layer maps discrete word representation into continuous vector representation.

2. Attention Model

Instead of reading in the entire input sentence and generating the output sentence all at once, the attention mechanism allows a network to focus on the most relevant parts of the input and produces output translation one at a time. A standard attention model consists of a pre-attention LSTM, an attention block and a post-attention LSTM. The function of each block is explained as follows:

- **Pre-attention LSTM**

The pre-attention LTSM is the one at the bottom that is evaluated before feeding into the attention block. The pre-attention LSTM goes through T_x number of time-steps that is equal to the length of the input sentence. Each time-step evaluates an activation $a^{(t)}$ using the input $x^{(t)}$ and the activation $a^{(t-1)}$ from the previous time-step. This is nothing different from a standard LSTM.

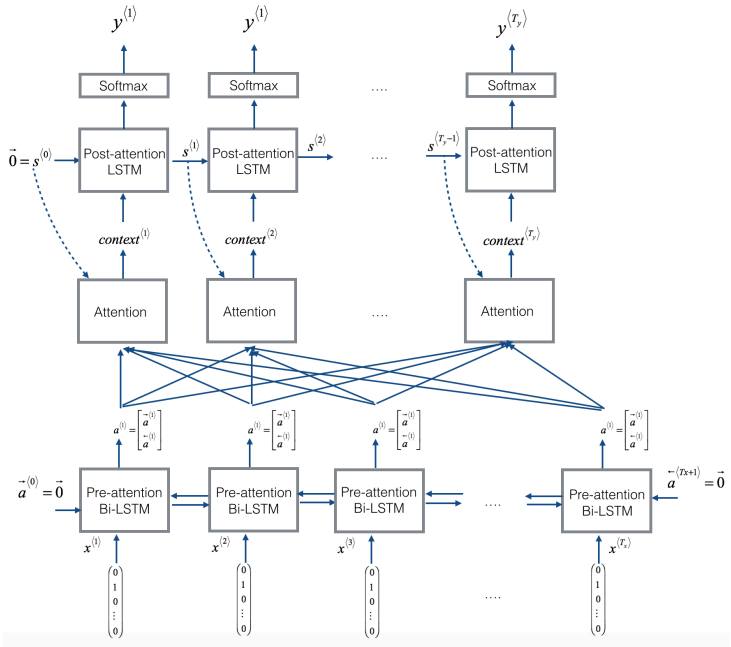

Figure 2. Attention Model

- **Post-attention LSTM**

The post-attention LSTM is the one at the top that is evaluated using the output $\text{context}^{(t)}$ from the attention block. The post-attention LSTM goes through T_y time-steps equal to the output length. The activation at each time-step of the post-attention LSMT is evaluated using the previous cell state $\text{s}^{(t-1)}$, hidden state $\text{c}^{(t-1)}$ and context vector $\text{context}^{(t)}$. We could also optionally pass in the generated output $y^{(t-1)}$ as input.

3. Attention Block

The key to the attention model is the attention block. The attention block consists of T_y time-steps, each time-step inputs the activation $a^{(1)}, a^{(2)}, \ldots, a^{(\mathrm{T}_x)}$ from the pre-attention LSTM and outputs a context vector $\mathrm{context}^{(t)}$ that provides semantic context for generating the current translation. The context vector is calculated by adding up the activations from pre-attention activation weighted by an **attention weight** $\alpha^{(t,t')}$, which tells us the attention that should be paid to $x^{(t')}$ in order to generate $y^{(t)}$

$$c^{(t)} = \sum_{t'} \alpha^{(t,t')} \cdot a^{(t')}$$

The attention weight $\alpha^{(t,t')}$ should depend on the cell state $\mathrm{s}^{(t-1)}$ of the post-attention LSTM and the activation $a^{(t')}$ of the pre-attention LSTM. This makes sense as the we expect the amount of attention paid to each word to be dependent on the word itself and the information from the previous translated word.

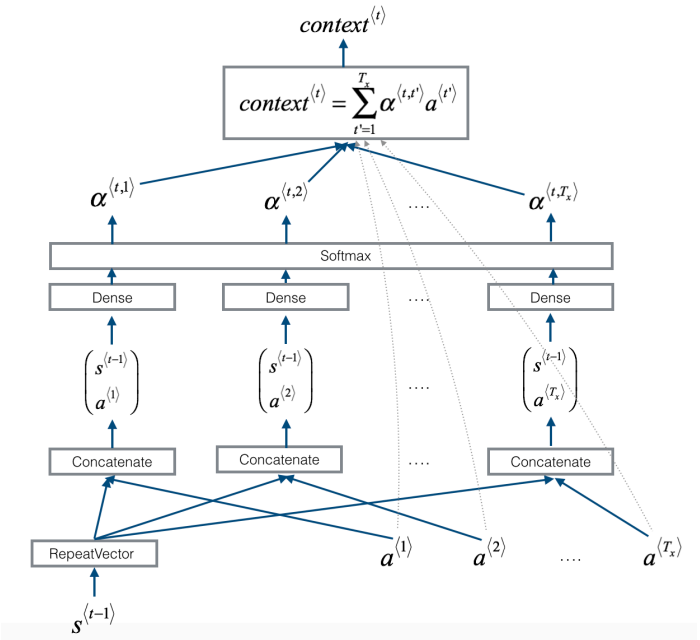

Figure 3. The Attention Block

To evaluate the attention weight, we use a dense layer to combine $s^{(t-1)}$ and $a^{(t)}$ input a single scalar $e^{(t,t')}$, which is then normalized using a softmax function to yield $\alpha^{(t,t')}$

$$\alpha^{(t,t')} = \frac{\exp\left(e^{(t,t')}\right)}{\sum_t \exp\left(e^{(t,t')}\right)} \qquad \sum_{t'} \alpha^{(t,t')} = 1 \quad \text{for all t}$$

- **Summary of Notations**

$y^{(t)}$: output at the t^{th} time-step of the post-attention LSTM network

$s^{(t)}$: activation at the t^{th} time-step of the post-attention LSTM network

$c^{(t)}$: context that is feed into the t^{th} time-step of the post-attention LSTM network

$x^{(t')}$: input at the t'^{th} time-step of the pre-attention LSTM network

$a^{(t')}$: activation at the t'^{th} time-step of the pre-attention LSTM network

$\alpha^{(t,t')}$: amount of attention $y^{(t)}$ should pay to $x^{(t')}$, it's dependent on $s^{(t-1)}$ and $a^{(t')}$

4. Attention Map

The attention weight $\alpha^{(t,t')}$ representing the attention that the network pays to the t'^{th} time-step of the pre-attention LSTM when generating the t^{th} word in the translation can be visualized using an attention map.

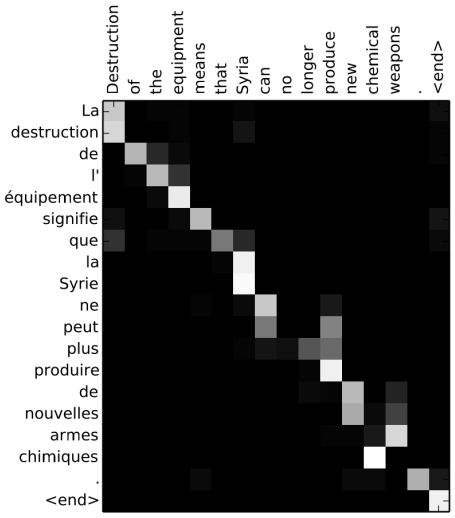

Figure 4. Visualization of Attention Weights

5. Computational Cost

The downside of the attention algorithm is that it has a quadratic complexity. If there are T_x words in the input and T_y words in the output, the total number of attention parameters is $O(\mathrm{T}_x \cdot \mathrm{T}_y)$. In real practice, we don't need to generate context using every time-step of the pre-attention LSTM. To reduce the cost, we could choose an appropriate window size T_w that is enough to include the most relevant input words to the current translation that we want to generate.

6. Implementation

In this section, we use the attention model to build a machine translator that is able to translate from one language to another, such as translating English to French. However, training such a translator on natural language to high accuracy requires a very large dataset and is difficult to train. For illustration purpose, we implement a simpler "date translation" task that translates human readable dates written in a variety of form ("the 25th of June 2009", "25/06/2009", "25 June 2009") into machine readable form ("2009-06-25").

- **Data Preparation**

For the dataset created, we set the maximum length of the human readable date $T_x = 30$, if we get a longer input, we could have to truncate it; and we set $T_y = 10$ as "YYYY-MM-DD" is exactly 10 characters long. In addition, we convert the sparse numerical encoding to continuous one-hot encoding for better performance.

- **Pre-attention LSTM**

The activation $a^{(t)}$ at time-step t of the bidirectional LSTM is a concatenation of the activation of both the forward-direction part $\vec{a}^{(t)}$ and backward-direction part $\overleftarrow{a}^{(t)}$, such that $a^{(t)} = \left[\vec{a}^{(t)};\ \overleftarrow{a}^{(t)}\right]$.

- **Attention Block & Post-attention LSTM**

The following procedure is iterated for every but the last time-step $t = 0, 1, \dots, T_y - 1$ of the post-attention LSTM. The attention block uses a $\mathrm{RepeatVector}$ node to copy the value $\mathrm{s}^{(t-1)}$ by T_x times and then $\mathrm{Concatenation}$ to concatenate $\mathrm{s}^{(t-1)}$ and $a^{(t)}$ to compute $e^{(t,t')}$ which is then passes through the softmax to compute the attention weight $\alpha^{(t,t')}$. The attention block outputs the context vector

$$\mathrm{context}^{(t)} = \sum_{t'=0}^{T_x} \alpha^{(t,t')} a^{(t')}$$

Next, we feed the context vector as the input to the t^{th} time-step of the post-attention LSTM together with the $\mathrm{initial\_state} = \left[\mathrm{s}^{(t-1)}, \mathrm{c}^{(t-1)}\right]$ to get back the new hidden state $\mathrm{s}^{(t)}$ and the new cell state $\mathrm{c}^{(t)}$. Finally, we apply a softmax function to $\mathrm{s}^{(t)}$ to get the output. We repeat the procedure for T_y time-steps and save all outputs to a dictionary.

- **Attention Map**

Let’s now visualize the attention values in the network. The position where the network pays attention to when generating each translation is exactly makes sense to you. In the date translation application, most of the time attention helps predict the year and hasn’t much impact on predicting the day/month.

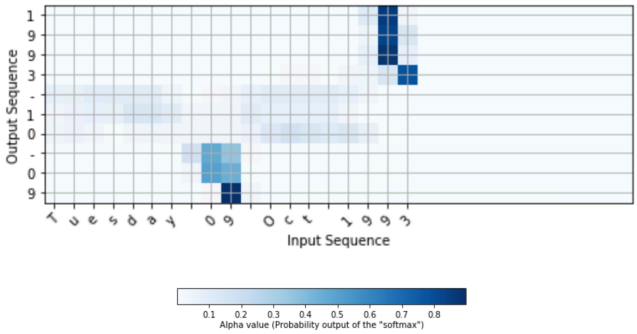

Figure 5. Attention Map for Date Translation