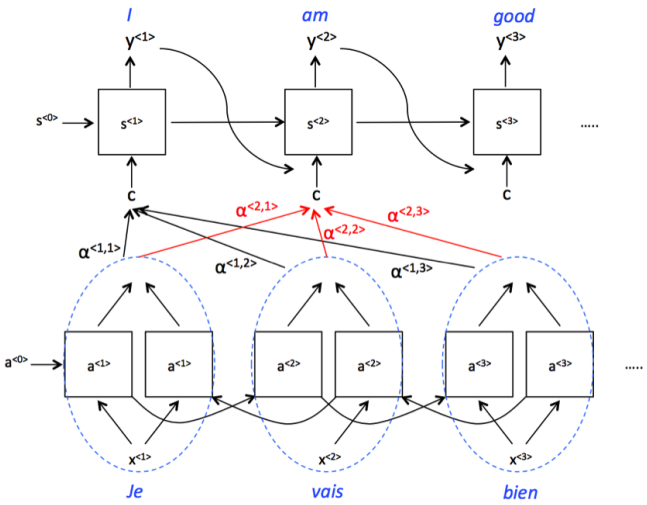
DeepLearning series: Attention Model and Speech Recognition

Michele Cavaioni Follow
Mar 14, 2018 · 4 min read

This model is an alternative to the encoder-decoder RNN architecture (see previous blog), and acts similar to how humans translate. Therefore, not waiting for the whole input sentence before translating, but starting doing it on the go, looking at one part of the original sentence at a time.

To architect a model in such a way, we need to have a context of words that the network pays attention to, in order to generate the subsequent words. The "attention weights" (α) denote how much attention should be paid to each piece of the original sentence.

The model is composed of two RNN networks. A bidirectional RNN to compute set of features for each of the input words and another RNN to generate the English translation. The input to the second RNN will be governed by the attention weights, which denote the context that is considered by every single word to compute the translation.



The alpha parameters tell us how much the context would depend on the features we are getting, so the activations we are receiving from the different time steps.

The "c" context is the weighted sum of the attention weights.

$$c^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$$

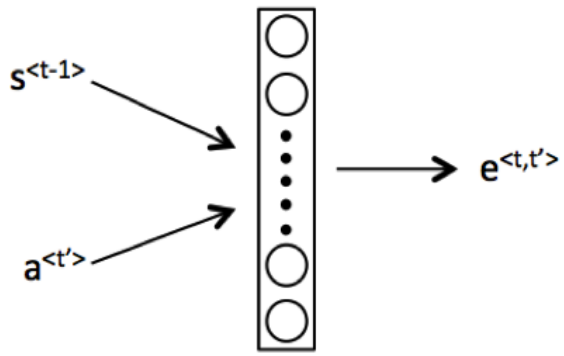
Therefore alpha<t,t'> is the amount of "attention" that y<t> should

pay to a<t'>.

$$\alpha^{<t,t'>} = \frac{\exp\left(e^{<t,t'>}\right)}{\sum_{t'=1}^{Tx} \exp\left(e^{<t,t'>}\right)}$$

Now we need to know how to compute these parameters alpha<t,t'>.

One way to do it is to train a small neural network where the inputs are represented by the network state in the previous time step (s<t-1>) and the features from the time step t' (a<t'>).



The downside of this algorithm is the time cost to run it. In fact, if we have Tx words as input and Ty for the output, the total number of attention parameters is Tx* Ty.

SPEECH RECOGNITION

The attention model is used for applications related to speech recognition, where the input is an audio clip and the output is its transcript. Before inputting the audio clip, there is one pre-processing step that is in charge of generating a spectrogram out of the raw audio clip.

In the past, speech recognition used to be built using an additional step, which computes phonemes before entering a neural network. Nowadays, we use end-to-end deep learning, which makes this step unnecessary. We take different time frames of the audio input and then the attention model outputs the transcript.

One successful model that is used is the CTC, Cost Per Speech Recognition.

CTC Cost Per Speech Recognition

Normally the input data Tx is much larger than the output Ty since an audio clip for example of 10 seconds at 100 Hz results in a 1,000 input sequence. While, instead, the output, the transcript, is not composed of 1,000 characters.

This model uses a simple RNN network where Tx = Ty.

The function of the model generates an output composed of repeated characters separated by blanks. (Like: "*ttttttt__h_eeeee____bbb_rrrr_oo_w_nnnn*"). Finally, it collapses repeated characters not separated by blanks to obtain the output sequence. ("*the brown*").

Trigger word detection system

You might be using one of those speakers in the house that respond to you when you call them by "name" and perform a particular action required at your request.

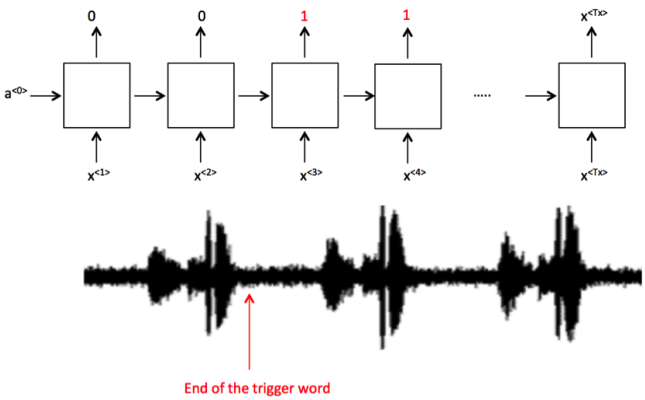
You know what I'm talking about, right? The exotic-named assistants? I just don't want to put names out there and discriminate against a single company

Anyway, those systems react to you when you say a "trigger" word. For example let's use the word espresso as our trigger word.… When you say "espresso" the voice assistant "wakes up" and listens to what you ask and does something for you.

Ever wondered how does that work? Well, the wait is over.

Here is how you use an RNN model and train it to learn a trigger word.

Architecturally we have the audio clip transformed into spectrogram features that pass through an RNN. We define the target labels y as 0 or 1. In the training dataset, we put the target labels to be 0 for everything before the trigger word and 1 right after.



If we just put 1 after the end of the trigger word and 0 everywhere else it will cause the training set to be unbalanced, with a lot more of

zeros than ones. A way to overcome this is to output 1 not just for a single time step, but for several times before reverting to 0.

Now the model is trained to “wake up” to a trigger word and be ready to listen to what you say!

. . .

This blog is based on Andrew Ng’s lectures at DeepLearning.ai