Topic 9: Conditioned sequence generators

Generating sequences blindly is fun but hardly useful. In general, a text generator needs to be supplied with information about 'what' it should generate. Examples of this auxiliary input to the sequence generator can be a photo in order to generate a description of it, a sentence in another language to generate a translation of it, an audio recording to generate a transcription of it, and so on.

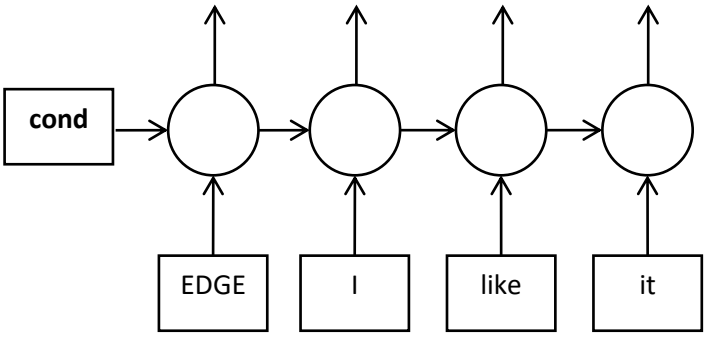
When a language model is fed extra information that effects the probabilities, we say that the language model is conditioned on that extra information. There are four different ways to insert a vector of information into a neural language model, which are outlined in this paper: init-inject, pre-inject, par-inject, and merge. We shall discuss each of these below. Note that these methods can be combined as well so that you get one model that uses both init-inject and merge for example.

In the code we will be creating a sentiment generator, that is, a language model that either generates positive sentiments or negative ones. The conditioning vector will be a one-hot vector that is two elements long that will allow us to specify whether we want a positive ('10') or a negative ('01') sentiment. Note that this is functionally equivalent to just using two separate language models, although the fact that one neural network is doing two tasks might result in having the same benefits as a multi-task neural network. In practice the conditioning vector will not be just a selection between two alternative choices but will be a vector of information such as from a photo.

1. Conditioning by init-inject

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/09_-_Conditioned_sequence_generators/01_-_Conditioning_by_init-inject.py

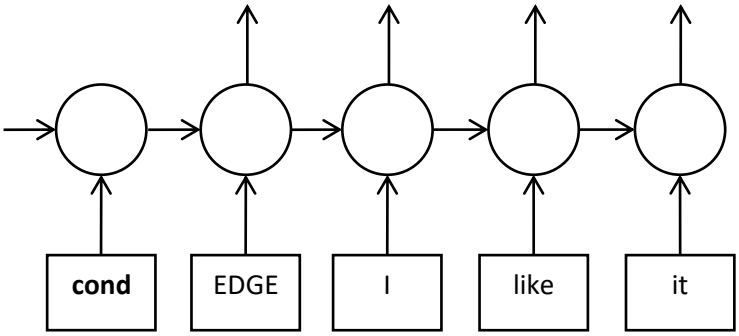
Init-inject is when the conditioning vector is used as an initial state to the RNN. This means that the conditioning vector must somehow become the same size as the RNN's state.



2. Conditioning by pre-inject

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/09_-_Conditioned_sequence_generators/02_-_Conditioning_by_pre-inject.py

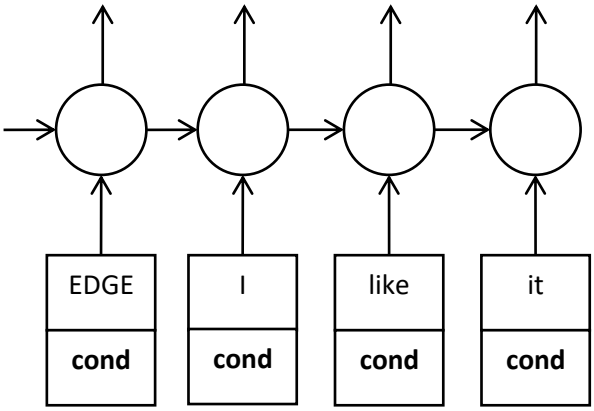
Pre-inject is when the conditioning vector is used as a first input to the RNN. This requires concatenating the conditional vector to the beginning of the embedded words, also to increase the sequence length given to the dynamic_rnn function, and removing the extra output generated by the RNN.



3. Conditioning by par-inject

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/09_-_Conditioned_sequence_generators/03_-_Conditioning_by_par-inject.py

Par-inject is when the conditioning vector is provided to the RNN as an input at each time step. It is usually just concatenated with the input vectors. Note that since the conditioning vector is inserted at every time step, you can either attach the same conditioning vector at each time step or use a different conditioning vector each time.



4. Conditioning by merge

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/09_-_Conditioned_sequence_generators/04_-_Conditioning_by_merge.py

Merge is when the conditioning vector is not included in the RNN but is instead included after the RNN processes the input sequence. It is usually just concatenated with the output vectors of the RNN. Note that since the conditioning vector is inserted at every time step, you can either attach the same conditioning vector at each time step or use a different conditioning vector each time.