Bidirectional RNN

Need for bidirectionality

- All of the recurrent networks we have considered up to now have a “causal” structure, meaning that the state at time t only captures information from the past, x(1), . . . , x(t−1), and the present input x(t) .
- Some of the models we have discussed also allow information from past y values to affect the current state when the y values are available.
- However, in many applications we want to output a prediction of y(t) which may depend on the whole input sequence.
- For example, in speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because of co-articulation and potentially may even depend on the next few words because of the linguistic dependencies between nearby words: if there are two interpretations of the current word that are both acoustically plausible, we may have to look far into the future (and the past) to disambiguate them.

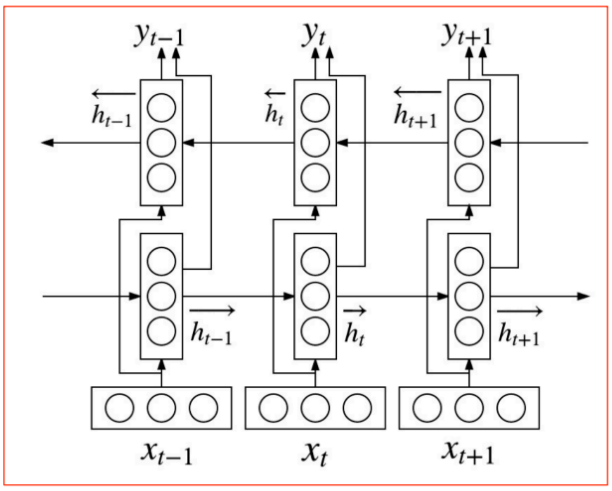
A bidirectional RNN

- As the name suggests, bidirectional RNNs combine an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backward through time beginning from the end of the sequence.

- h(t) standing for the state of the sub-RNN that moves forward through time and

- g(t) standing for the state of the sub-RNN that moves backward through time.

- This allows the output units o(t) to compute a representation that depends on both the past and the future but is most sensitive to the input values around time t, without having to specify a fixed-size window around t (as one would have to do with a feedforward network, a convolutional network, or a regular RNN with a fixed-size look-ahead buffer).


y_{t-1}
y_t
y_{t+1}
$\overleftarrow{h}_{t-1}$
$\overleftarrow{h}_t$
$\overleftarrow{h}_{t+1}$
$\overrightarrow{h}_{t-1}$
$\overrightarrow{h}_t$
$\overrightarrow{h}_{t+1}$
x_{t-1}
x_t
x_{t+1}

In the case of the bidirectional network, we have separate forward and backward parameter matrices. The forward matrices for the input-hidden, hidden-hidden, and hidden-output interactions are denoted by $W_{xh}^{(f)}$, $W_{hh}^{(f)}$, and $W_{hy}^{(f)}$, respectively. The backward matrices for the input-hidden, hidden-hidden, and hidden-output interactions are denoted by $W_{xh}^{(b)}$, $W_{hh}^{(b)}$, and $W_{hy}^{(b)}$, respectively.

The recurrence conditions can be written as follows:

$$\begin{aligned}
\overline{h}_t^{(f)} &= \tanh(W_{xh}^{(f)}\overline{x}_t + W_{hh}^{(f)}\overline{h}_{t-1}^{(f)}) \\
\overline{h}_t^{(b)} &= \tanh(W_{xh}^{(b)}\overline{x}_t + W_{hh}^{(b)}\overline{h}_{t+1}^{(b)}) \\
\overline{y}_t &= W_{hy}^{(f)}\overline{h}_t^{(f)} + W_{hy}^{(b)}\overline{h}_t^{(b)}
\end{aligned}$$

- It is easy to see that the bidirectional equations are simple generalizations of the conditions used in a single direction.
- It is assumed that there are a total of T time-stamps in the neural network shown above, where T is the length of the sequence.
- One question is about the forward input at the boundary conditions corresponding to t = 1 and the backward input at t = T, which are not defined.
- In such cases, one can use a default constant value of 0.5 in each case, although one can also make the determination of these values as a part of the learning process.
- An immediate observation about the hidden states in the forward and backwards direction is that they do not interact with one another at all.
- Therefore, one could first run the sequence in the forward direction to compute the hidden states in the forward direction, and then run the sequence in the backwards direction to compute the hidden states in the backwards direction. At this point, the output states are computed from the hidden states in the two directions
- After the outputs have been computed, the backpropagation algorithm is applied to compute the partial derivatives with respect to various parameters.

One can summarize the steps as follows:

1. Compute forward and backwards hidden states in independent and separate passes.

2. Compute output states from backwards and forward hidden states.

3. Compute partial derivatives of loss with respect to output states and each copy of the output parameters.

4. Compute partial derivatives of loss with respect to forward states and backwards states independently using backpropagation. Use these computations to evaluate partial derivatives with respect to each copy of the forwards and backwards parameters.

5. Aggregate partial derivatives over shared parameters. Bidirectional recurrent neural networks are appropriate for applications in which the predictions are not causal based on a historical window.