Module 5

Bi-directional recurrent neural networks (Bi-RNNs) are artificial neural networks that process input data in both the forward and backward directions. They are often used in natural language processing tasks, such as language translation, text classification, and named entity recognition. In addition, they can capture contextual dependencies in the input data by considering past and future contexts. Bi-RNNs consist of two separate RNNs that process the input data in opposite directions, and the outputs of these RNNs are combined to produce the final output.

What is Bidirectional RNN?

A bi-directional recurrent neural network (Bi-RNN) is a type of recurrent neural network (RNN) that processes input data in both forward and backward directions. The goal of a Bi-RNN is to capture the contextual dependencies in the input data by processing it in both directions, which can be useful in various natural language processing (NLP) tasks.

In a Bi-RNN, the input data is passed through two separate RNNs: one processes the data in the forward direction, while the other processes it in the reverse direction. The outputs of these two RNNs are then combined in some way to produce the final output.

One common way to combine the outputs of the forward and reverse RNNs is to concatenate them. Still, other methods, such as element-wise addition or multiplication, can also be used. The choice of combination method can depend on the specific task and the desired properties of the final output.

Need for Bi-directional RNNs

- A uni-directional recurrent neural network (RNN) processes input sequences in a single direction, either from left to right or right to left.
- This means the network can only use information from earlier time steps when making predictions at later time steps.
- This can be limiting, as the network may not capture important contextual information relevant to the output prediction.
- For example, in natural language processing tasks, a uni-directional RNN may not accurately predict the next word in a sentence if the previous words provide important context for the current word.

Consider an example where we could use the recurrent network to predict the masked word in a sentence.

1. Apple is my favorite _____.
2. Apple is my favourite _____, and I work there.
3. Apple is my favorite _____, and I am going to buy one.

In the first sentence, the answer could be fruit, company, or phone. But it can not be a fruit in the second and third sentences.

Module 5

A Recurrent Neural Network that can only process the inputs from left to right may not accurately predict the right answer for sentences discussed above.

To perform well on natural language tasks, the model must be able to process the sequence in both directions.

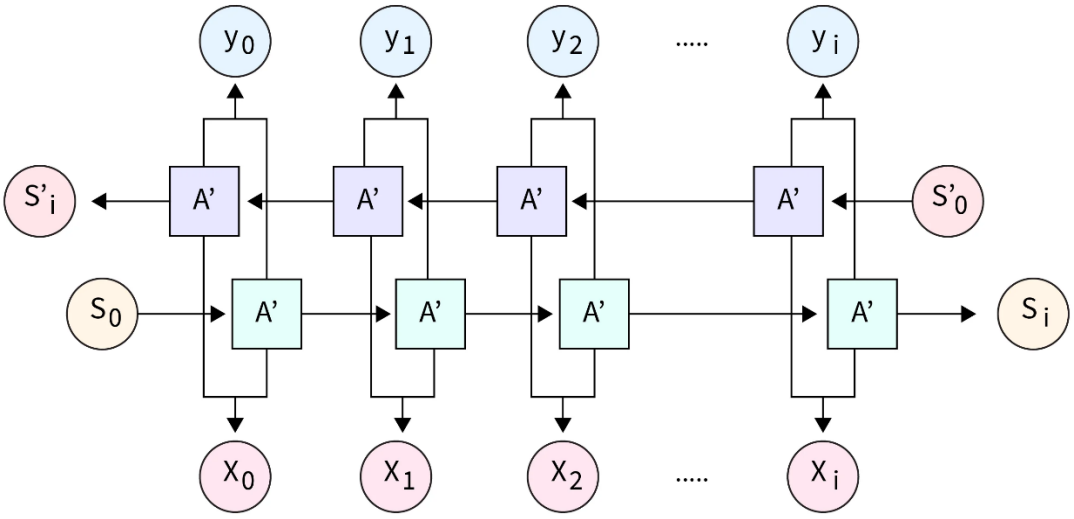
Bi-directional RNNs

- A bidirectional recurrent neural network (RNN) is a type of recurrent neural network (RNN) that processes input sequences in both forward and backward directions.
- This allows the RNN to capture information from the input sequence that may be relevant to the output prediction. Still, the same could be lost in a traditional RNN that only processes the input sequence in one direction.
- This allows the network to consider information from the past and future when making predictions rather than just relying on the input data at the current time step.
- This can be useful for tasks such as language processing, where understanding the context of a word or phrase can be important for making accurate predictions.
- In general, bidirectional RNNs can help improve a model's performance on various sequence-based tasks.

This means that the network has **two separate RNNs**:

1. One that processes the input sequence from left to right
2. Another one that processes the input sequence from right to left.

These two RNNs are typically called forward and backward RNNs, respectively.



Module 5

During the forward pass of the RNN, the forward RNN processes the input sequence in the usual way by taking the input at each time step and using it to update the hidden state. The updated hidden state is then used to predict the output.

Backpropagation through time (BPTT) is a widely used algorithm for training recurrent neural networks (RNNs). It is a variant of the backpropagation algorithm specifically designed to handle the temporal nature of RNNs, where the output at each time step depends on the inputs and outputs at previous time steps.

In the case of a bidirectional RNN, BPTT involves two separate Backpropagation passes: one for the forward RNN and one for the backward RNN. During the forward pass, the forward RNN processes the input sequence in the usual way and makes predictions for the output sequence. These predictions are then compared to the target output sequence, and the error is backpropagated through the network to update the weights of the forward RNN.

The backward RNN processes the input sequence in reverse order during the backward pass and predicts the output sequence. These predictions are then compared to the target output sequence in reverse order, and the error is backpropagated through the network to update the weights of the backward RNN.

Once both passes are complete, the weights of the forward and backward RNNs are updated based on the errors computed during the forward and backward passes, respectively. This process is repeated for multiple iterations until the model converges and the predictions of the bidirectional RNN are accurate.

This allows the bidirectional RNN to consider information from past and future time steps when making predictions, which can significantly improve the model's accuracy.

Merge Modes in Bidirectional RNN

In a bidirectional recurrent neural network (RNN), two separate RNNs process the input data in opposite directions (forward and backward). The output of these two RNNs is then combined, or "merged," in some way to produce the final output of the model.

There are several ways in which the outputs of the forward and backward RNNs can be merged, depending on the specific needs of the model and the task it is being used for. Some common merge modes include:

1. **Concatenation:** In this mode, the outputs of the forward and backward RNNs are concatenated together, resulting in a single output tensor that is twice as long as the original input.
2. **Sum:** In this mode, the outputs of the forward and backward RNNs are added together element-wise, resulting in a single output tensor that has the same shape as the original input.
3. **Average:** In this mode, the outputs of the forward and backward RNNs are averaged element-wise, resulting in a single output tensor that has the same shape as the original input.
4. **Maximum:** In this mode, the maximum value of the forward and backward outputs is taken at each time step, resulting in a single output tensor with the same shape as the original input.

Module 5

Which merge mode to use will depend on the specific needs of the model and the task it is being used for. Concatenation is generally a good default choice and works well in many cases, but other merge modes may be more appropriate for certain tasks.