## Bidirectional Recurrent Neural Networks

In the landscape of neural networks designed for sequential data, standard Recurrent Neural Networks (RNNs) process information in a single, forward direction. This architectural choice limits their ability to incorporate future context when making a prediction at a given time step. To overcome this limitation, Bidirectional Recurrent Neural Networks (BRNNs) were introduced, providing a more comprehensive understanding of the sequence by processing it in both forward and backward directions.

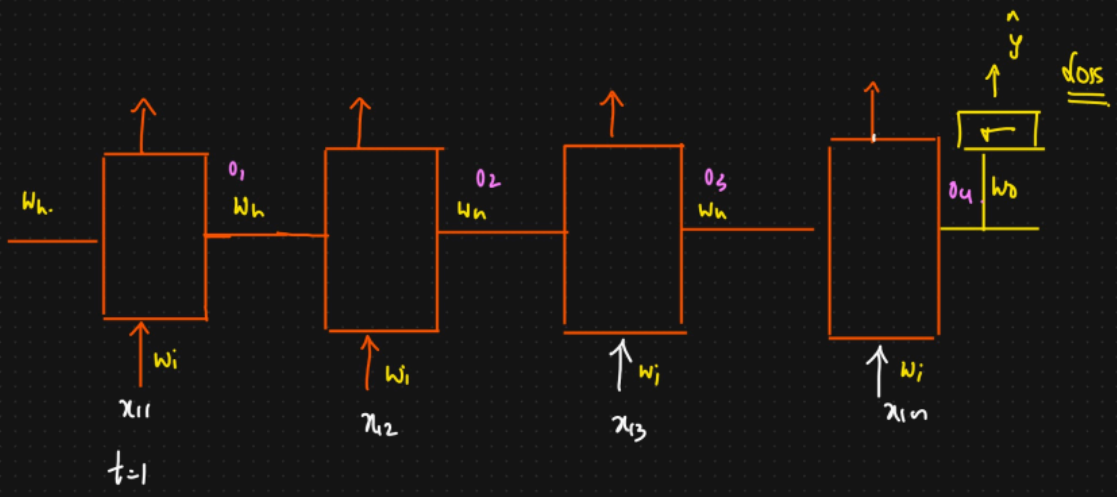
## The Shortcoming of Unidirectional RNNs

A standard RNN, including more advanced variants like LSTMs and GRUs, processes a sequence chronologically. At any given time step *t*, the hidden state is a function of the input at that step and the hidden state from the previous step, *t-1*. This means the network's understanding at a particular point is based solely on past and present information.

However, in many tasks, particularly in Natural Language Processing (NLP), the context of a word is not just determined by the words that precede it but also by those that follow. For example, in the sentence "He said, 'Teddy is a great leader.'", to understand that "Teddy" refers to a person (like Teddy Roosevelt) and not a toy, the subsequent words "is a great leader" are crucial. A unidirectional RNN would struggle with such ambiguity as it wouldn't have access to this future context.

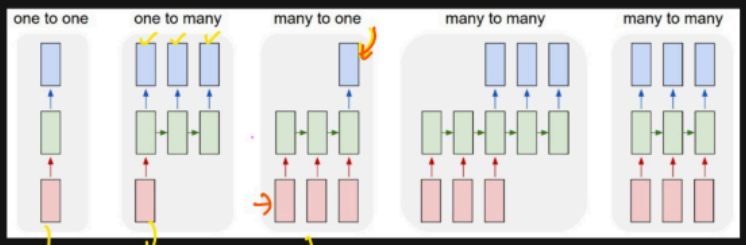
# Bidirectional RNN

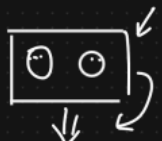
1. Simple RNN → Practical Implement → Embedding Layers
2. LSTM, GRU Variants RNN → Practical Implement
3. Bidirectional RNN / LSTM RNN




## Types of RNN

1. One to Many RNN
2. Many to One RNN
3. Many to Many RNN.
4. One to One RNN



There is a dog and cat

Eg:

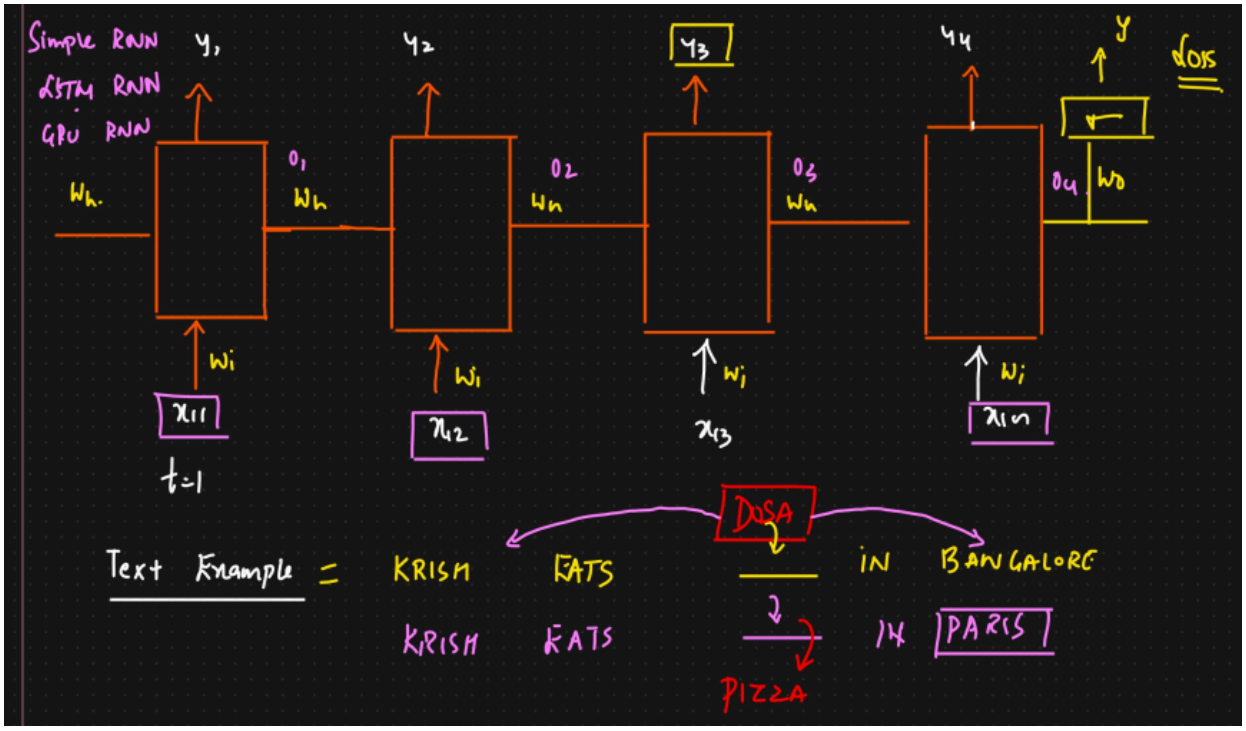
One i/p
One O/p

One i/p
O/p Many

Eg: Image Captioning

Image scanning
I/p Many
O/p One.
A cat eating food
O eating

Language Translation
Many I/p
Many O/p.




## The Architecture of a Bidirectional RNN

A Bidirectional RNN addresses this by employing two separate RNN layers that process the input sequence in opposite directions. Here's a breakdown of its architecture:

1. **Forward Layer:** This is a standard RNN layer that processes the input sequence from the beginning to the end (from time step *t=1* to *t=T*). At each time step *t*, it produces a forward hidden state, denoted as $\overrightarrow{h_t}$, which captures information from the past.

$$\overrightarrow{h_t} = f(W_{\overrightarrow{h}x} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h_{t-1}} + b_{\overrightarrow{h}})$$

2. **Backward Layer:** This second RNN layer processes the input sequence in reverse, from the end to the beginning (from time step *t=T* to *t=1*). It produces a backward hidden state, denoted as $\overleftarrow{h_t}$, which encapsulates information from the future.

$$\overleftarrow{h_t} = f(W_{\overleftarrow{h}x} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h_{t+1}} + b_{\overleftarrow{h}})$$

3. **Output Layer:** At each time step *t*, the final output, $y_t$, is a function of both the forward and backward hidden states at that specific time step. This is typically achieved by concatenating the two hidden states, but other operations like summation or averaging can also be used.

$$y_t = g(W_y[\overrightarrow{h_t}, \overleftarrow{h_t}] + b_y)$$

Here:

- xt is the input at time step *t*.
- W represents the weight matrices.
- b represents the bias vectors.
- f and g are activation functions.

This structure allows the network, at any point in the sequence, to have a representation that includes both preceding and succeeding information.

It's important to note that the underlying recurrent units in a BRNN can be simple RNN cells, LSTMs, or GRUs. Using LSTMs or GRUs is common to mitigate the vanishing gradient problem and effectively capture long-range dependencies in both directions.

## How Bidirectional RNNs Work: A Step-by-Step Example

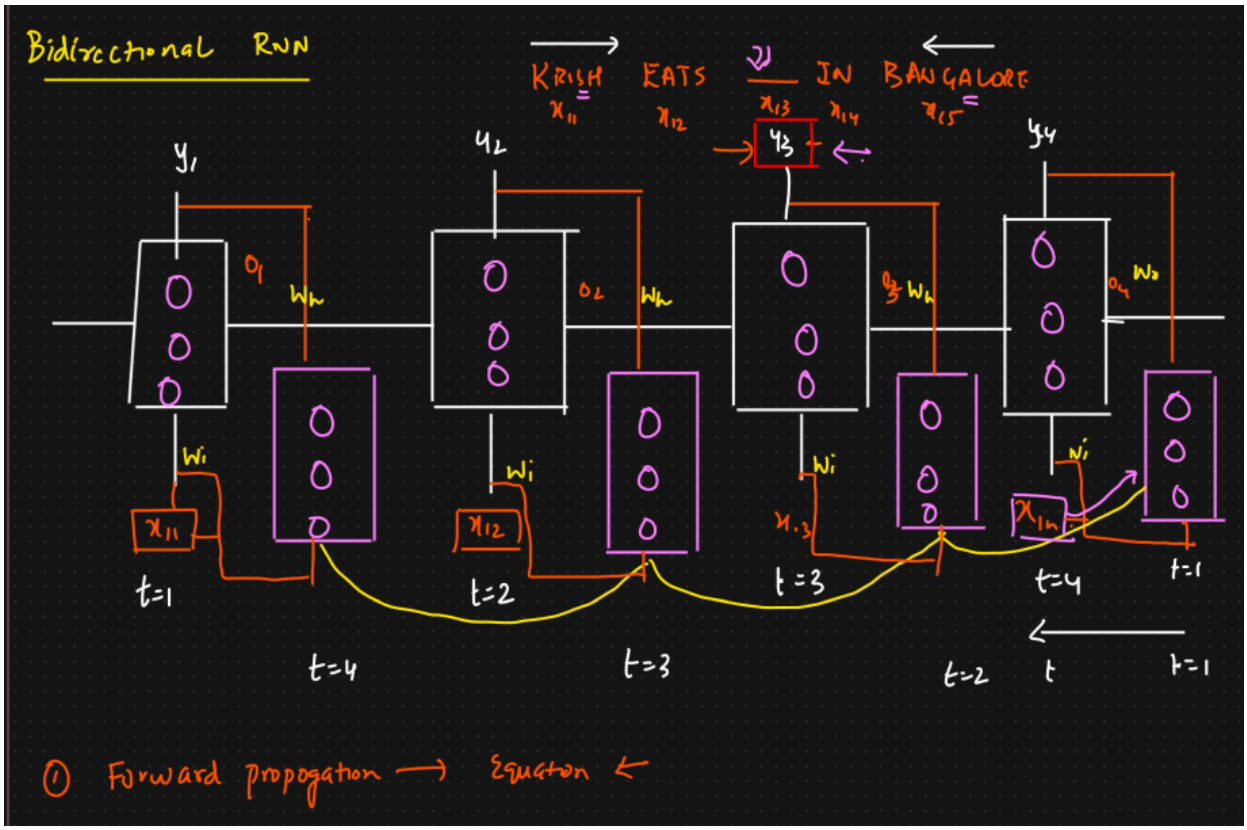
Let's consider the sentence "The cat sat on the mat." and how a BRNN would process the word "sat."

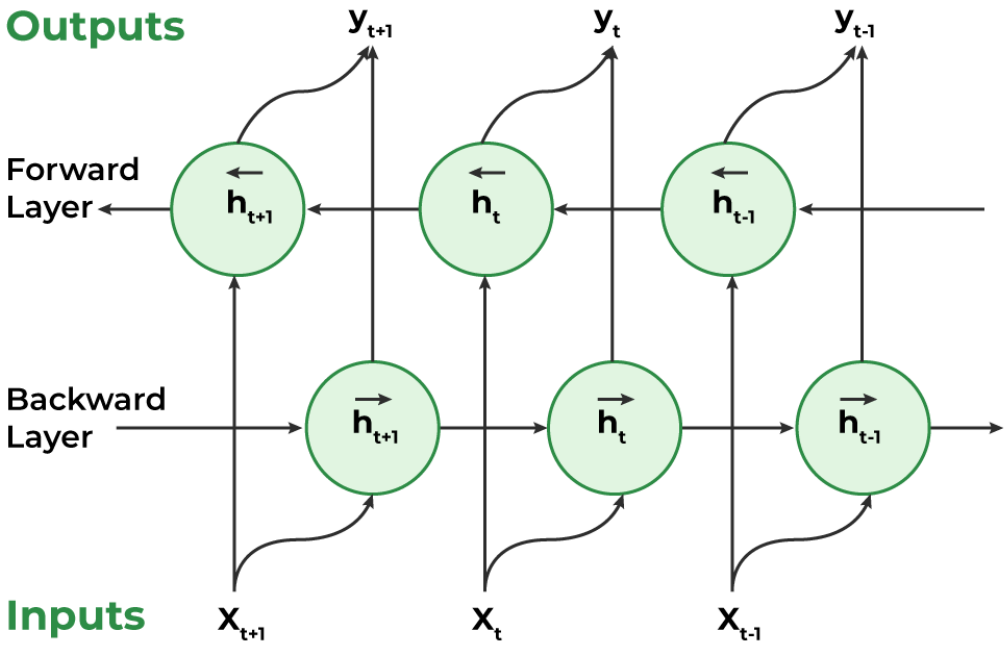
1. **Forward Pass:** The forward RNN processes the sentence from left to right. By the time it reaches "sat," it has already processed "The" and "cat." Its hidden state at "sat" will

contain contextual information from these preceding words.

2. **Backward Pass:** The backward RNN processes the sentence from right to left. It starts with "mat.", then "the," and then "on." When it reaches "sat," its hidden state contains context from the words that follow it.

3. **Combining Context:** At the "sat" time step, the BRNN combines the hidden state from the forward pass (context from "The cat") and the hidden state from the backward pass (context from "on the mat."). This combined representation gives a much richer understanding of the word "sat" in the full context of the sentence.






## Advantages of Bidirectional RNNs

- **Improved Contextual Understanding:** By considering both past and future information, BRNNs can create more accurate and context-aware representations of sequential data.
- **Enhanced Performance:** This deeper understanding often leads to significantly better performance on various NLP and sequence-to-sequence tasks compared to their unidirectional counterparts.
- **Flexibility:** They can be used with any type of recurrent unit (vanilla RNN, LSTM, GRU), allowing for a balance between performance and computational complexity.

## Limitations of Bidirectional RNNs

- **Unsuitability for Real-Time Applications:** The most significant drawback of BRNNs is that they require the entire input sequence to be available before they can make a prediction for any time step. This is because the backward pass needs to start from the end of the sequence. Consequently, they are not suitable for real-time applications like live speech recognition or stock price prediction where the future is unknown.
- **Increased Computational Complexity:** Since a BRNN has two separate RNN layers, it has roughly double the number of parameters as a unidirectional RNN, leading to increased training time and a higher demand for computational resources.
- **Potentially Slower Inference:** The need to process the sequence in two passes can make inference slower than with a unidirectional model.

## Applications of Bidirectional RNNs

Despite their limitations, BRNNs have proven to be highly effective in a wide range of applications where the entire input sequence is accessible, including:

- **Natural Language Processing (NLP):**
    - **Named Entity Recognition (NER):** Identifying entities like names, locations, and organizations in a text.
    - **Sentiment Analysis:** Determining the sentiment of a piece of text.
    - **Machine Translation:** Especially in the encoder part of an encoder-decoder architecture, to create a rich representation of the source sentence.
    - **Part-of-Speech Tagging:** Assigning grammatical categories to words.
- **Speech Recognition:** Post-processing recorded speech to improve transcription accuracy.
- **Bioinformatics:** Analyzing biological sequences like DNA and proteins.
- **Handwriting Recognition:** Recognizing characters in the context of the entire word or line.

In conclusion, Bidirectional RNNs are a powerful extension of the standard RNN model. By breaking the constraint of processing information in a single direction, they provide a more holistic view of sequential data, leading to state-of-the-art results in many non-real-time applications.