

# NLP4: RNN,LSTM, GRU and Bidirectional LSTM



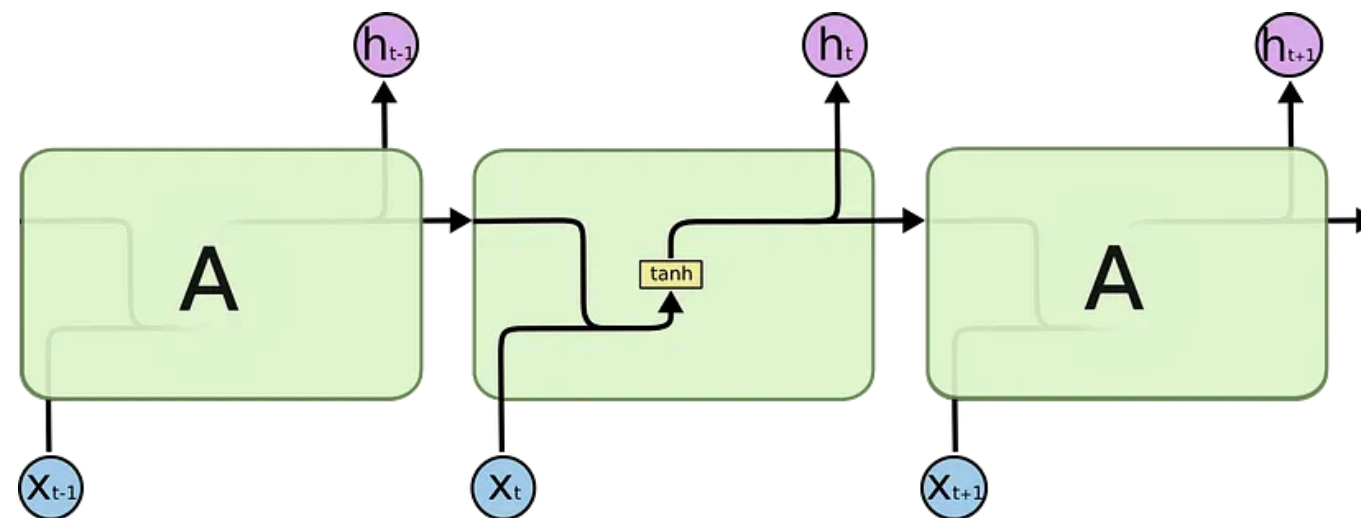
Saket Kumar

12 min read · Just now

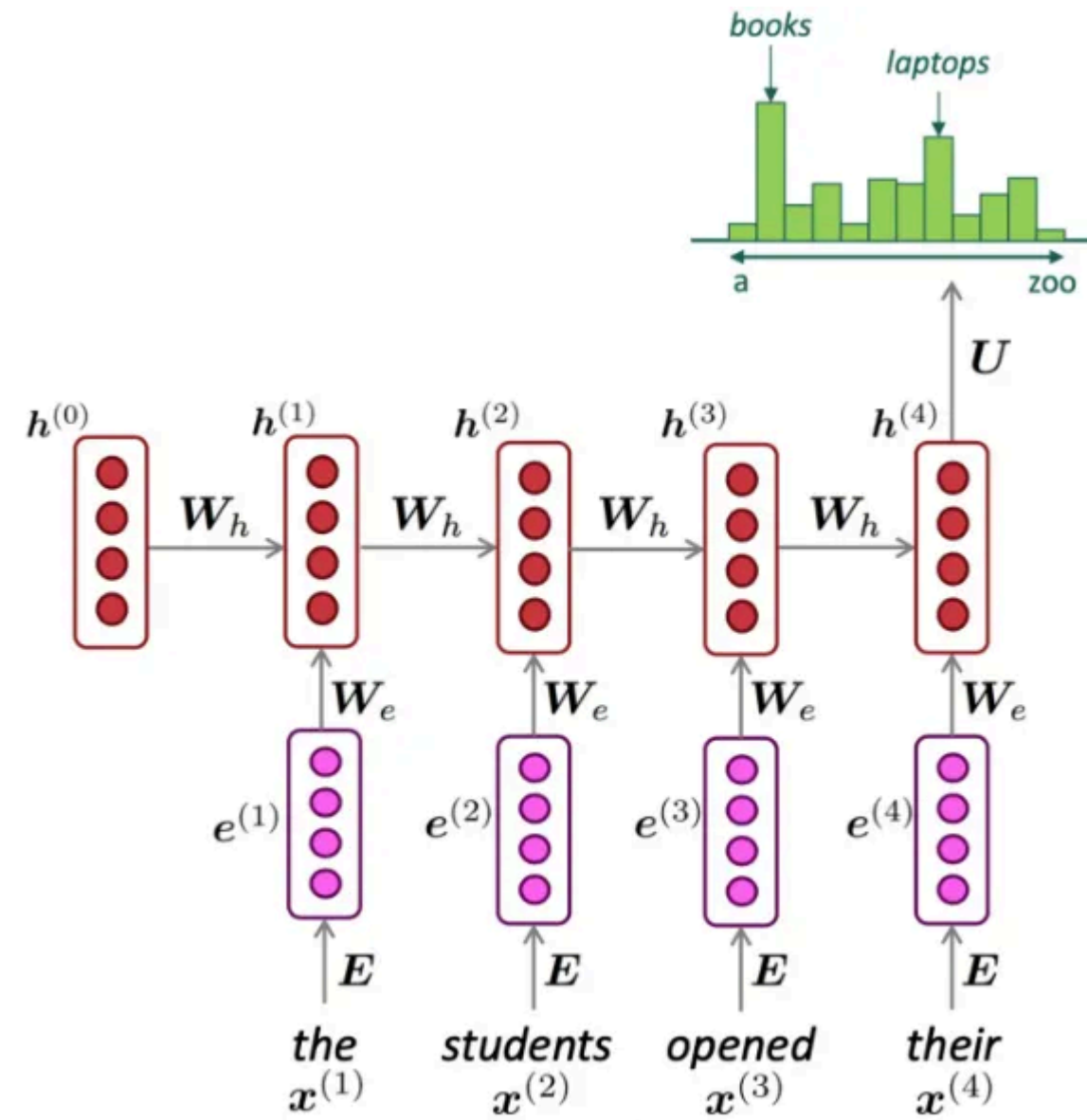


## RNN

Recurrent Neural Networks (RNNs) are a type of artificial neural network specifically designed to work with sequential data, such as time series, text, or audio. Unlike traditional neural networks, which assume that inputs are independent of each other, RNNs have a built-in mechanism to retain and utilize information from previous inputs in the sequence, making them ideal for tasks where context or order matters.



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



<https://aiml.com/briefly-describe-the-architecture-of-a-recurrent-neural-network-rnn/>

## Key Concepts of RNNs

### Sequential Data and Temporal Dependencies:

- RNNs are particularly well-suited for sequential data, where the order of the data points matters. For example, in natural language processing (NLP), the meaning of a word often depends on the words that come before it.

### Hidden State:

- RNNs maintain a hidden state that gets updated as the network processes each element of the sequence. This hidden state acts as a memory, capturing information about the previous inputs.

### Recurrent Connections:

- In an RNN, the output from the previous step is fed back into the network as input for the next step, along with the current input. This recurrent connection allows the network to retain information over time.

### Mathematical Formulation:

- At each time step  $t$ , the RNN takes an input  $x_t$  and the previous hidden state  $h_{t-1}$  to compute the new hidden state  $h_t$ :

$$h_t = \sigma(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h)$$

where:

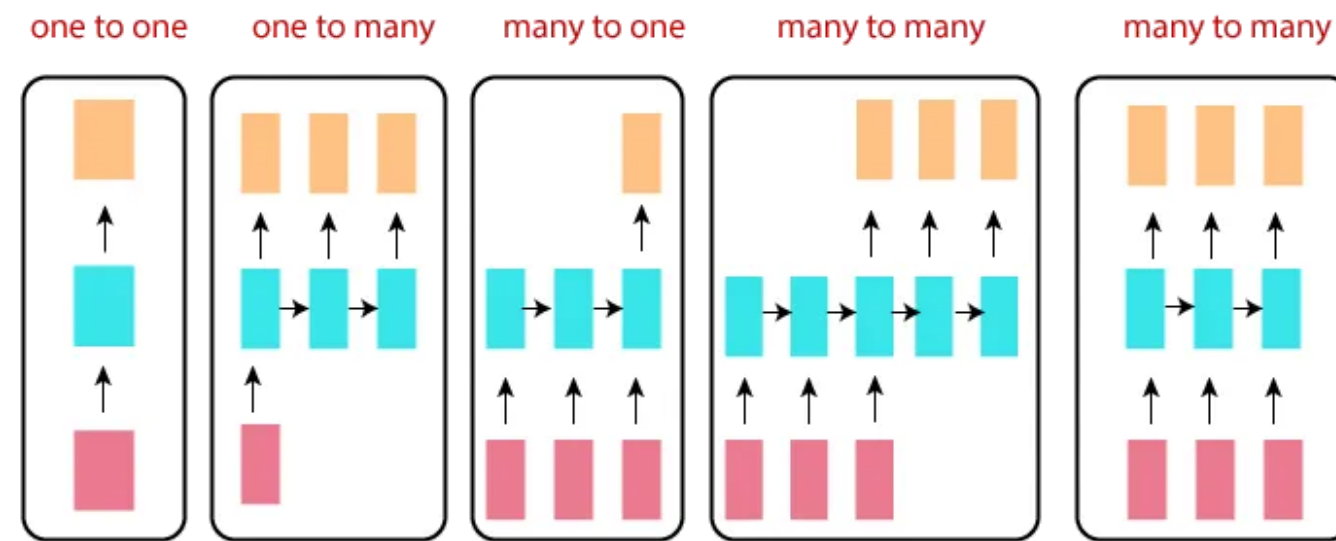
- $W_h$  is the weight matrix for the input.
- $U_h$  is the weight matrix for the hidden state.
- $b_h$  is the bias term.
- $\sigma$  is a non-linear activation function, such as tanh or ReLU.
- The output  $y_t$  can be computed as:

$$y_t = \sigma(W_y \cdot h_t + b_y)$$

where:

- $W_y$  is the weight matrix for the output.
- $b_y$  is the bias term for the output.

### Types of RNN Architectures



<https://www.javatpoint.com/tensorflow-types-of-rnn>

### One-to-One:

- A standard neural network where each input corresponds to a single output. This architecture is not usually referred to as an RNN but is the simplest form.

### One-to-Many:

- This architecture takes a single input and generates a sequence of outputs. It's useful for tasks like image captioning, where a single image (input) generates a sequence of words (output).

### Many-to-One:

- This architecture takes a sequence of inputs and produces a single output. It's commonly used in sentiment analysis, where the entire sequence of words in a sentence (input) results in a sentiment score (output).

### Many-to-Many:

- This can either be a sequence-to-sequence architecture, where both the input and output are sequences of the same length (e.g., video classification), or a sequence transduction architecture, where the input and output sequences have different lengths (e.g., machine translation).

## Challenges of RNN's

### Vanishing and Exploding Gradients:

- During training, the gradients used for back propagation can either become very small (vanishing) or very large (exploding), making it difficult to train the network effectively. This problem is especially prevalent in long sequences.

### Long-Term Dependencies:

- RNNs struggle to retain information from inputs that are far apart in the sequence. This issue limits the network's ability to capture long-term dependencies.

## Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) designed to overcome the limitations of traditional RNNs, particularly in dealing with long-range dependencies in sequential data. LSTM networks are particularly well-suited for tasks involving time series prediction, language modeling, and other sequence-related problems.

### 1. Understanding the Problem with Standard RNNs

Before diving into LSTMs, it's essential to understand why they were developed in the first place. Traditional RNNs work by passing the hidden state from one time step to the next, which theoretically allows them to retain information from previous time steps. However, they suffer from two major problems:

- **Vanishing Gradient Problem:** When training RNNs, gradients used to update the model's weights can become very small, especially when dealing with long sequences. This leads to minimal updates and poor learning of long-term dependencies.
- **Exploding Gradient Problem:** Conversely, gradients can also become too large, causing unstable updates and making the model difficult to train.

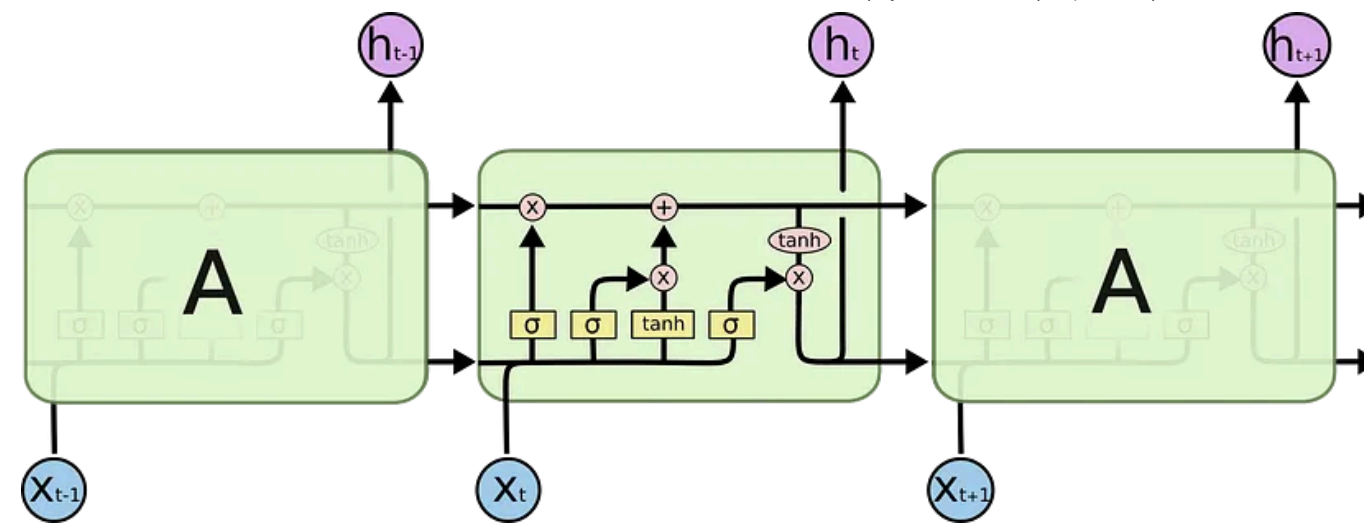
LSTM networks address these problems by introducing a memory cell structure that can maintain its state over long periods.

## 2. LSTM Architecture

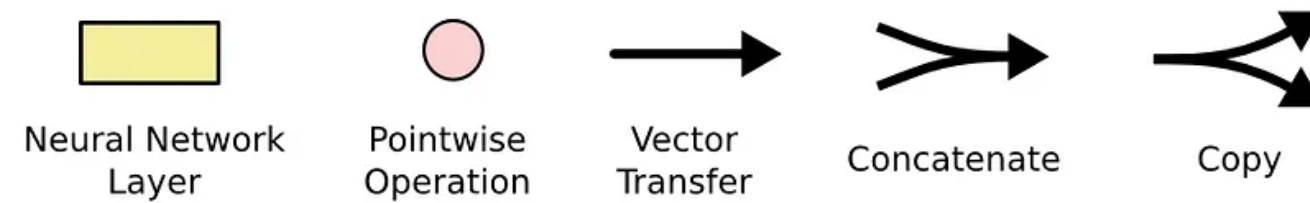
An LSTM unit is composed of several key components:

- **Cell State (  $c_t$  ):** This is the memory of the network, which can be thought of as a conveyor belt that runs through the entire sequence. It allows information to flow unchanged, with only some minor linear interactions, which helps in maintaining long-term dependencies.
- **Gates:** LSTMs have three gates that regulate the flow of information into and out of the cell state:
  1. **Forget Gate (  $f_t$  ):** Decides what information to throw away from the cell state.
  2. **Input Gate (  $i_t$  ):** Decides which information to add to the cell state.
  3. **Output Gate (  $o_t$  ):** Decides what information from the cell state to output.

These gates are implemented using sigmoid functions, which output values between 0 and 1, where 0 means “completely forget” and 1 means “completely remember.”



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



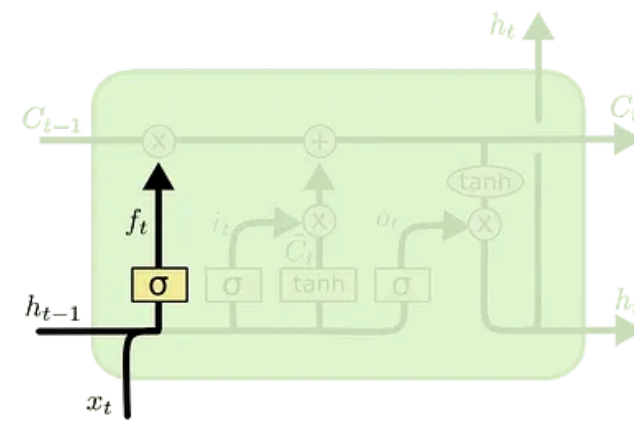
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

### 3. Step-by-Step Working of an LSTM Cell

Let's break down the LSTM operation at a single time step  $t$ :

#### 1. Forget Gate ( $f_t$ )

The forget gate determines how much of the previous cell state  $c_{t-1}$  should be carried forward. It takes the previous hidden state  $h_{t-1}$  and the current input  $x_t$ , passes them through a sigmoid activation function, and outputs a number between 0 and 1 for each value in the cell state.



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

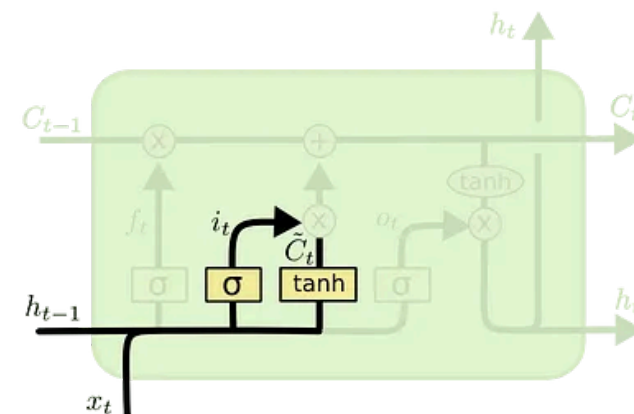
Where:

- $w_f$  is the weight matrix for the forget gate.
- $b_f$  is the bias term.
- $\sigma$  is the sigmoid function.

## 2. Input Gate ( $i_t$ ) & Candidate Memory Cell ( $\tilde{C}_t$ )

The input gate decides how much of the new input should influence the current cell state. It also uses a sigmoid function:

Simultaneously, a candidate memory cell is created using a tanh activation function, which will add new information to the cell state:



$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Where:

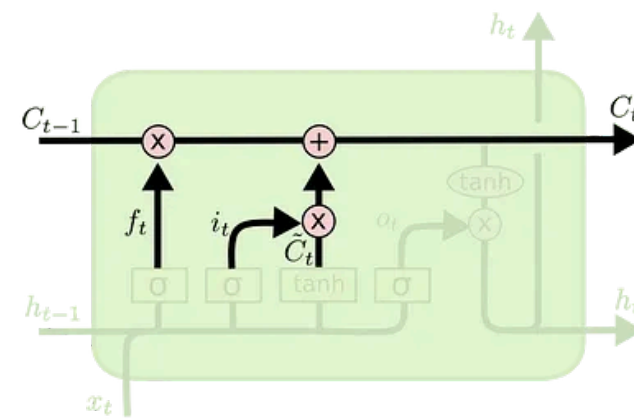


- $W_i$  and  $W_C$  are weight matrices for the input gate and candidate memory cell, respectively.
- $b_i$  and  $b_C$  are bias terms.

### 3. Update Cell State ( $C_t$ )

The current cell state  $C_t$  is updated by combining the old cell state (scaled by the forget gate) and the candidate memory cell (scaled by the input gate):

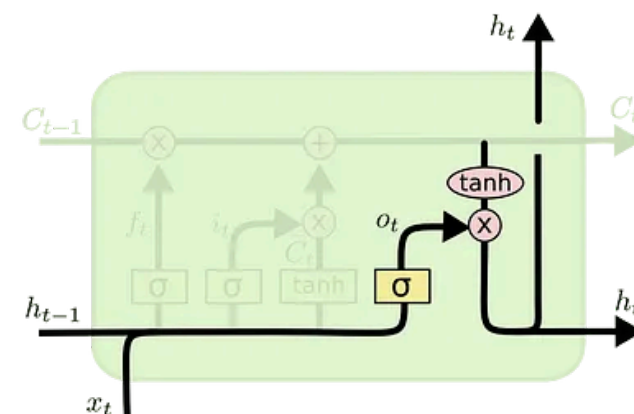
This allows the cell to retain information over long periods while also incorporating new, relevant information.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

### 4. Output Gate ( $o_t$ ) & Hidden State ( $h_t$ )

Finally, the output gate decides what the next hidden state  $h_t$  should be. The hidden state is a filtered version of the cell state, determined by the output gate:



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Where:

- $W_o$  is the weight matrix for the output gate.
- $b_o$  is the bias term.

#### 4. Advantages of LSTM

- **Long-Term Dependency Handling:** The unique structure of LSTM allows it to learn and remember over long sequences, effectively addressing the vanishing gradient problem.
- **Selective Memory:** LSTM can selectively forget or remember information, making it more efficient in processing sequential data.
- **Flexibility:** LSTMs can be used in various sequence-based tasks such as time series prediction, language modeling, and sequence-to-sequence learning.

#### 5. LSTM in Action

Let's consider a practical example: predicting the next word in a sentence. Given a sequence of words, an LSTM can learn patterns over the sequence, remembering important context (like the subject of a sentence) over long periods, and using that context to predict the next word.

For instance, in the sentence “The cat sat on the \_\_\_”, an LSTM might predict “mat” as the next word, based on the patterns it learned from previous sentences in its training data.

#### Bidirectional LSTM

Bidirectional Long Short-Term Memory (BiLSTM) networks are an extension of the traditional LSTM networks. They are designed to capture information from both past (backward direction) and future (forward direction) contexts,

which makes them particularly effective in tasks where context from both directions is important, such as language modeling, speech recognition, and machine translation.

## 1. Traditional LSTM Recap:

- **LSTM Overview:** Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to overcome the vanishing gradient problem, making them capable of learning long-term dependencies.
- **Key Components:**
- **Cell State (  $c_t$  ):** The memory of the network that carries information across time steps.
- **Forget Gate (  $f_t$  ):** Decides which information from the cell state should be discarded.
- **Input Gate (  $i_t$  ):** Determines which new information should be added to the cell state.
- **Output Gate (  $o_t$  ):** Controls what part of the cell state should be output.

In a traditional LSTM, the information flows only in one direction — from past to future. This means the LSTM processes the sequence in a forward manner.

## 2. Bidirectional LSTM:

- **Concept:** Bidirectional LSTMs (BiLSTMs) address the limitation of unidirectional LSTMs by processing the input data in both forward and backward directions. This allows the network to have two different LSTMs — one for each direction — working together.
- **Architecture:**

- **Forward LSTM:** Processes the sequence from the beginning to the end (left to right).
- **Backward LSTM:** Processes the sequence from the end to the beginning (right to left).
- **Output:** At each time step, the outputs of both LSTMs are concatenated or combined to form the final output for that time step.

### 3. Why Use Bidirectional LSTM?

- **Enhanced Contextual Understanding:** In many sequential tasks, the meaning of a word, event, or data point can depend on both the preceding and succeeding elements. For example, in the sentence “He said it was **not** good,” the word “not” changes the meaning of “good.” A unidirectional LSTM might struggle to fully capture this dependency, whereas a BiLSTM can consider context from both directions.
- **Improved Performance:** By incorporating information from both past and future contexts, BiLSTMs can achieve better performance in tasks like named entity recognition (NER), part-of-speech (POS) tagging, and text classification.

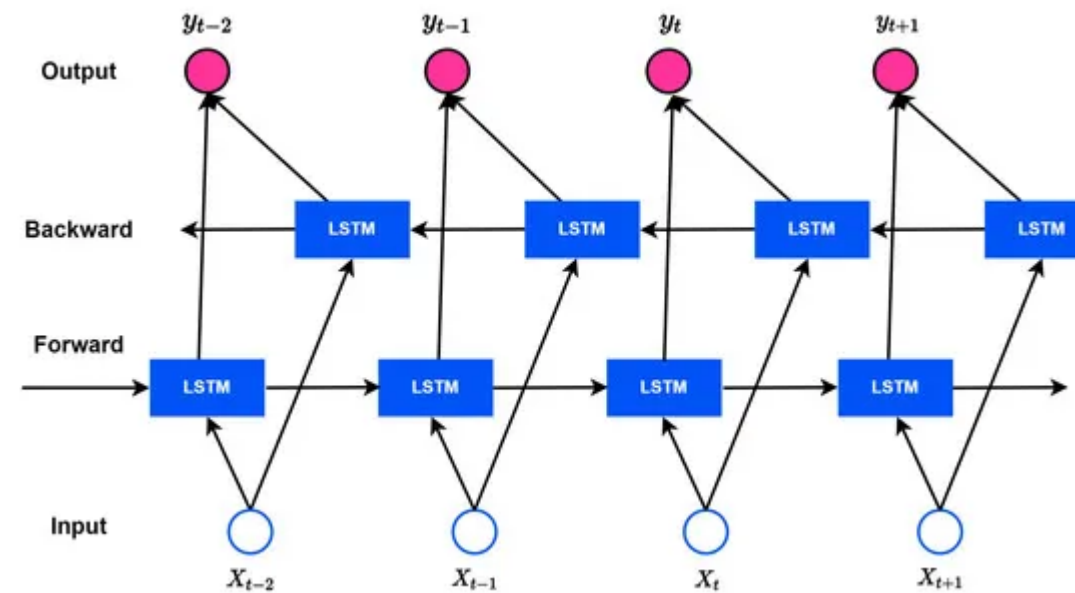
### 4. Intuition with Example:

Imagine you’re reading a sentence word by word. If you only remember the words you’ve already read (unidirectional LSTM), you might miss out on the meaning that would become clearer if you could also consider the words that come after. However, with a BiLSTM, you simultaneously consider words from both directions, giving you a more comprehensive understanding.

#### Example Sentence:

“He did not win the race because he was injured.”

- **Forward LSTM:** Processes from “He” to “injured.” At the word “injured,” it only knows the preceding words.
- **Backward LSTM:** Processes from “injured” to “He.” At the word “injured,” it knows all the subsequent words.
- **Combined:** At each word, the network has access to information from both directions, allowing it to understand the full context.



<https://www.mdpi.com/2076-3417/12/13/6390>

## 5. Mathematical Formulation:

Given a sequence of inputs  $\{x_1, x_2, \dots, x_T\}$ , a BiLSTM computes:

- **Forward LSTM:**

$$\vec{h}_t = \text{LSTM}_{\text{forward}}(x_t, \vec{h}_{t-1})$$

- **Backward LSTM:**

$$\overleftarrow{h}_t = \text{LSTM}_{\text{backward}}(x_t, \overleftarrow{h}_{t+1})$$

- **Combined Output:**

$$h_t = \text{concat}(\vec{h}_t, \overleftarrow{h}_t)$$

Where  $h_t$  is the final output at time step  $t$  that combines information from both directions.

## 6. Practical Considerations:

- **Memory and Computation:** BiLSTMs require more computational resources and memory than unidirectional LSTMs because they involve two LSTMs running in parallel.
- **Applications:** BiLSTMs are widely used in NLP tasks like sentiment analysis, machine translation, and speech recognition, where understanding context in both directions is crucial.

## 7. Visualization:

Imagine a BiLSTM as two parallel LSTM networks, with one processing inputs from start to end and the other from end to start. At each time step, the outputs of these two networks are merged, providing a richer representation of the input sequence.

## Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is another type of Recurrent Neural Network (RNN) architecture, similar to Long Short-Term Memory (LSTM), but with a simplified structure. GRUs were introduced to address the vanishing

gradient problem while being computationally less expensive than LSTMs. GRUs have gained popularity because of their effectiveness and simpler design.

## 1. Understanding the Motivation Behind GRUs

Like LSTMs, GRUs were developed to improve the learning of long-term dependencies in sequential data. Traditional RNNs struggle with the vanishing gradient problem, which makes it difficult for them to learn relationships over long sequences. While LSTMs tackle this issue effectively, they are complex and have many parameters, making them computationally intensive. GRUs aim to strike a balance between complexity and performance.

## 2. GRU Architecture

A GRU consists of two main gates:

1. **Update Gate ( $Z_t$ )**: Determines how much of the past information needs to be passed along to the future.
2. **Reset Gate ( $R_t$ )**: Decides how much of the past information to forget.

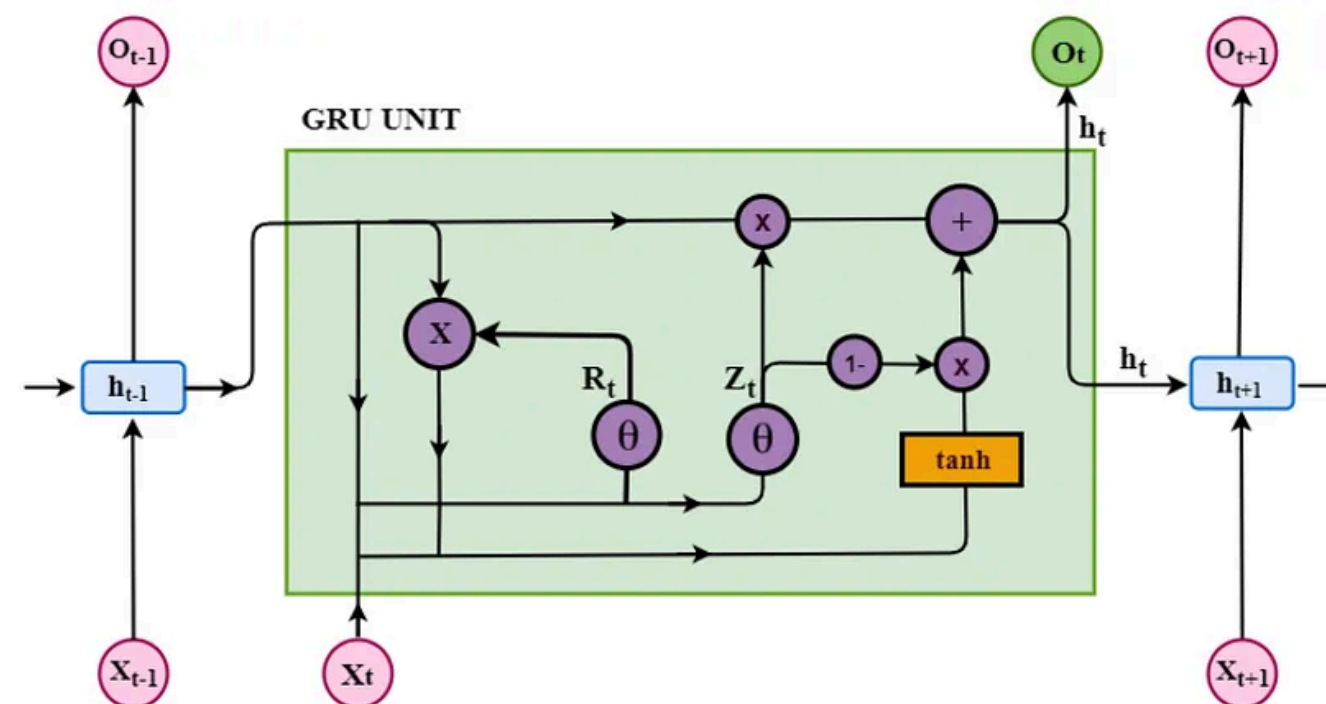
Example:

“I enjoy playing basketball, but my sister prefers reading mystery novels, which are full of suspense.”

- **Update Gate ( $Z_t$ )**: If the conversation continues with something like “so we often spend our weekends doing different activities,” the Update Gate will retain the information about basketball and mystery novels to maintain the flow of the conversation.

- **Reset Gate ( $R_t$ ):** If the conversation shifts to something unrelated, like “but today we are planning a family trip,” the Reset Gate will decide to discard the earlier details about basketball and mystery novels, as they are not relevant to the new topic about the family trip.

These gates control the flow of information in and out of the hidden state, similar to how LSTM gates control the cell state.



[https://www.researchgate.net/figure/The-Architecture-of-basic-Gated-Recurrent-Unit-GRU\\_fig1\\_343002752](https://www.researchgate.net/figure/The-Architecture-of-basic-Gated-Recurrent-Unit-GRU_fig1_343002752)

### 3. Step-by-Step Working of a GRU Cell

Let's break down the operations within a GRU cell at a single time step  $t$ :

#### 1. Update Gate ( $Z_t$ )

The update gate controls the extent to which the previous hidden state  $h_{t-1}$  should be carried forward to the current hidden state  $h_t$ . It decides how much of the past information should be kept and how much new information should be added.



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

Where:

- $W_z$  is the weight matrix for the update gate.
- $b_z$  is the bias term.
- $\sigma$  is the sigmoid function, which outputs values between 0 and 1.

The update gate can be thought of as a soft switch that controls how much of the previous hidden state is carried forward.

## 2. Reset Gate ( $r_t$ )

The reset gate controls how much of the previous hidden state to forget. It allows the model to reset the hidden state when new, important information arrives.

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

Where:

- $W_r$  is the weight matrix for the reset gate.
- $b_r$  is the bias term.

The reset gate enables the GRU to drop irrelevant information from the past, which can be useful when the current input is significantly different from the previous ones.

## 3. Current Memory Content ( $\tilde{h}_t$ )

Once the reset gate has determined which parts of the previous hidden state to keep, the GRU computes the candidate hidden state (or current memory content), which incorporates the current input  $x_t$  and the previous hidden state  $h_{t-1}$  that has been filtered by the reset gate.

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t] + b)$$

Where:

- $w$  is the weight matrix for the candidate hidden state.
- $b$  is the bias term.
- $\tanh$  is the hyperbolic tangent activation function, which outputs values between -1 and 1.

This candidate hidden state represents the new information that could be added to the final hidden state  $h_t$ .

#### 4. Final Hidden State ( $h_t$ )

The final hidden state  $h_t$  is a combination of the previous hidden state  $h_{t-1}$  and the candidate hidden state  $\tilde{h}_t$ . The update gate  $z_t$  determines how much of each to keep:

$$h_t = z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t$$

This equation ensures that the GRU can both retain information from the past and incorporate new information as needed. If  $z_t$  is close to 1, the previous hidden state  $h_{t-1}$  is mostly retained. If  $z_t$  is close to 0, the model primarily relies on the newly computed candidate hidden state  $\tilde{h}_t$ .

## 4. Advantages of GRU

- **Simplified Architecture:** GRUs have fewer gates and parameters compared to LSTMs, making them computationally less expensive and easier to train.
- **Effective Performance:** Despite their simpler structure, GRUs often perform as well as LSTMs, particularly on tasks with less complex dependencies.
- **Reduced Overfitting:** The fewer parameters in GRUs can help reduce the risk of overfitting, especially when working with smaller datasets.

## 5. GRU vs. LSTM

While both GRUs and LSTMs are designed to address the same problem of long-term dependencies in sequential data, they differ in their approach:

- **Complexity:** LSTMs have more gates (forget, input, and output gates) and a separate cell state, making them more complex than GRUs. GRUs merge the forget and input gates into a single update gate and do not maintain a separate cell state, which simplifies their structure.
- **Training Time:** GRUs, being simpler, usually train faster and require less computational power.
- **Performance:** GRUs can perform comparably to LSTMs in many tasks, but there are scenarios where LSTMs might have an edge due to their ability to model more complex dependencies.

## 6. GRU in Action

Consider a practical example: predicting the next word in a sentence, similar to the LSTM example. Given a sequence of words, a GRU can learn to retain the relevant context from previous words and discard irrelevant information. For instance, in the sentence “The sun rises in the \_\_\_”, a GRU

might predict “east” as the next word based on the patterns it has learned from its training data.



Written by Saket Kumar

Edit profile

10 Followers

Seasoned mechanical engineer looking to get into Data Scientist role.


Recommended from Medium



 Eric Sentell  in Age of Awareness

Teaching During the “Rise of AI” and the “End of Reading”



 Hazel Paradise

How I Create Passive Income With No Money

many ways to start a passive income today