Understanding LSTM

Minor in AI - IIT ROPAR

17th March, 2025

LSTM – A Journey to Overcome RNN Limitations

Imagine you're reading a novel, and the story builds slowly, with subtle hints dropped in the early chapters that pay off only in the final pages. As you progress, you need to remember key details, like a character's backstory or a mysterious event that happened many chapters ago. Without those details, the story would lose its depth and meaning, and you'd struggle to understand the unfolding plot.

Now, picture a machine trying to read this same novel and predict what comes next based on its previous chapters. This task, known as sequence prediction, requires the machine to remember previous words to make an accurate prediction. But there's a challenge: what if the sentence is long, and the machine needs to remember things that were said far earlier in the sentence?

In the early days, we used a simple approach called **Recurrent Neural Networks (RNNs)** to solve this problem. RNNs were designed with the idea of feeding the output from the previous step back into the model as input, thus giving the network some memory. But as you tried to train these RNNs with longer sentences, something strange happened: the machine began to forget. It wasn't remembering what was important, and instead, it was getting stuck on irrelevant details. This was known as the **vanishing gradient problem**.

The machine couldn't seem to remember long-term dependencies effectively, no matter how deep or complex the sequence was. It was as if the machine could only focus on the immediate past, completely ignoring the earlier parts of the sequence.

Thus began the quest to improve RNNs. Researchers sought ways to allow the machine to maintain long-term memory and remember important details over many time steps. This quest led to the development of more advanced models: **LSTMs** (Long Short-Term Memory units). Each of these models addressed the problems faced by RNNs, one step at a time, leading to more powerful and accurate models.

Now, let's break down the inner workings of LSTMs and understand how they revolutionized sequence modeling.

Introduction to LSTM (Long Short-Term Memory)

LSTMs are a type of recurrent neural network (RNN) that can capture long-term dependencies in sequential data. Unlike regular RNNs, which can forget important information over time, LSTMs use a special architecture that helps them remember crucial information over long periods and make better predictions in sequential tasks.

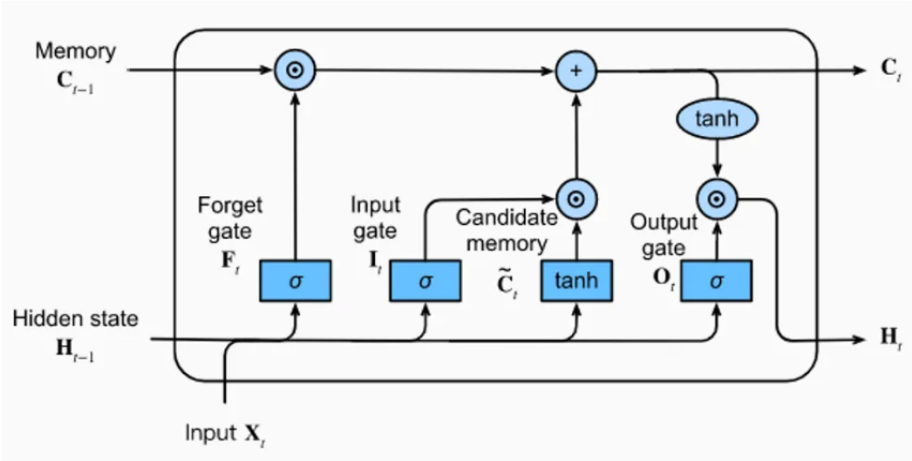
LSTM networks consist of **memory cells**, which store the information to be remembered, and **gates**, which control how information flows through the network. These gates determine which information is kept and which is discarded, making LSTM a powerful model for sequence learning tasks.

LSTM Architecture

An LSTM cell has three key components:

- **Forget Gate**
- **Input Gate**
- **Output Gate**

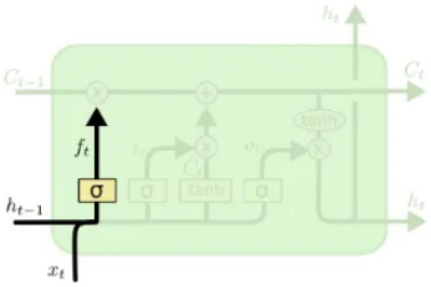
Each of these gates plays a crucial role in deciding what information should be remembered, updated, or discarded at each time step. Let's go step by step through each of these components.



Forget Gate: Deciding What to Forget

The first step in the LSTM update flow is the forget gate. This gate decides how much of the information from the previous time step's cell state (i.e., memory) should be carried over to the next step. In sequential tasks, not all past information is useful for making predictions at the current time step. Therefore, the forget gate filters out irrelevant information and keeps only what is necessary.

The forget gate looks at both the previous hidden state and the current input to make its decision. If it outputs a value close to 0, it means that most of the previous memory should be discarded, effectively "forgetting" it. On the other hand, if it outputs a value close to 1, it means that the previous memory should be retained almost entirely, ensuring that the past information is kept for the future steps.



Mathematical Representation:

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

Where:

- f_t is the output of the forget gate (a value between 0 and 1).
- σ is the sigmoid activation function, which outputs values between 0 and 1, determining how much of the past memory should be kept.
- W_f is the weight matrix for the forget gate.
- h_{t-1} is the previous hidden state.
- x_t is the input at the current time step.
- b_f is the bias term.

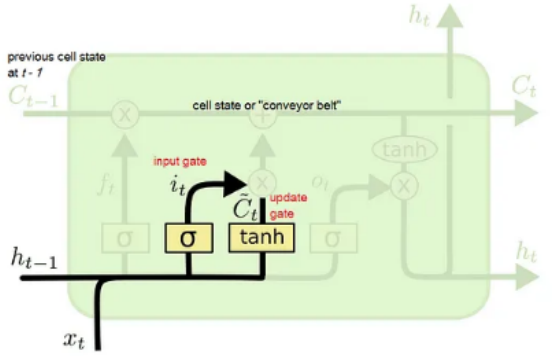
Interpretation:

- A value of $f_t = 0$ means **forget everything** from the previous memory.
- A value of $f_t = 1$ means **retain everything** from the previous memory.

Input Gate: Deciding What New Information to Add

The input gate controls how much new information from the current input should be added to the memory cell. It works in two stages:

- First, it decides how much of the incoming information should be passed through.
- Then, it creates a candidate memory $\tilde{C}_t$, which is a new memory content that could potentially be added to the memory cell.



Mathematical Representation:

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

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

Where:

- i_t is the input gate output.
- $\tilde{C}_t$ is the candidate cell state (the new information).
- W_i, W_C are the weight matrices for the input and candidate cell state.
- b_i, b_C are the bias terms.
- tanh is the hyperbolic tangent activation function.

Interpretation:

- The input gate i_t determines which parts of the new information should be used to update the memory cell.
- The candidate memory $\tilde{C}_t$ is the new information that could be added to the cell state.

Cell State Update: Combining Forget and Input Gates

Once the forget and input gates have made their respective decisions, the cell state is updated. The cell state is the long-term memory of the LSTM, and it gets modified by combining both the forgotten memory from the previous time step and the newly added memory from the current time step.

The forget gate helps determine which parts of the old memory should be kept, and the input gate decides how much of the new information should be incorporated. These two pieces of information are then merged to form the updated cell state, which is now ready to influence the LSTM's future operations. This combination ensures that the LSTM maintains a dynamic memory, adapting to both past and new information as the sequence progresses.

Mathematical Representation:

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

Where:

- C_{t-1} is the cell state from the previous time step.
- f_t is the forget gate's output.
- i_t is the input gate's output.
- $\tilde{C}_t$ is the candidate memory.

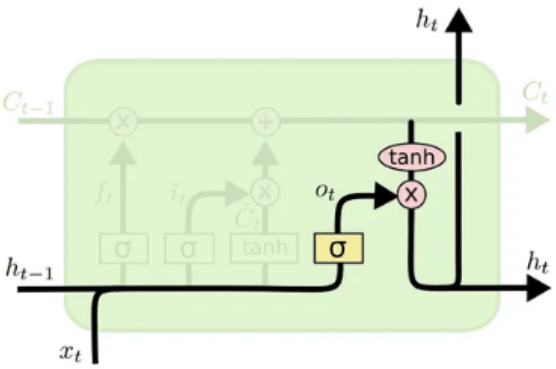
Interpretation:

- The cell state C_t is updated by forgetting some parts of the old memory (controlled by f_t) and adding some new information (controlled by i_t).

Output Gate: Deciding What to Output

Finally, the output gate determines how much of the current cell state should be exposed as the output of the LSTM at the current time step. This output, also known as the hidden state, is passed on to the next time step or the next layer in the network.

The output gate looks at both the current input and the previous hidden state to make its decision. Based on this, it selectively chooses how much of the updated cell state should be visible as the output. This output is then passed to the next layer or used for making predictions, depending on the task.



Mathematical Representation:

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

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

Where:

- o_t is the output gate.
- h_t is the hidden state output at the current time step, which is used as the output of the LSTM.
- C_t is the updated cell state.
- W_o is the weight matrix for the output gate.
- b_o is the bias term.

Interpretation:

- The output gate o_t controls which parts of the cell state C_t will be visible as the output.
- The hidden state h_t is the output of the LSTM cell, and it is passed to the next time step or the next layer in the network.

Full LSTM Update Flow

Let's summarize the entire LSTM cell operation at each time step t:

1. The forget gate f_t decides how much of the previous cell state C_{t-1} should be remembered.
2. The input gate i_t decides how much of the new information should be added to the memory.
3. The cell state C_t is updated by combining the forgotten memory and the new memory.
4. The output gate o_t decides how much of the cell state C_t should be output as the hidden state h_t.

This process allows LSTMs to maintain and update long-term memory across time steps, solving the vanishing gradient problem faced by regular RNNs.

Advantages of LSTMs

- **Memory Over Long Sequences:** LSTMs can remember information over long sequences, making them suitable for tasks where long-term dependencies matter, such as language modeling or time-series forecasting.
- **Efficient Gradient Flow:** The use of gates allows LSTMs to control the gradient flow, preventing the vanishing or exploding gradient problems, which are common in traditional RNNs.
- **Flexible Information Control:** The three gates (forget, input, and output) provide the model with fine-grained control over what information to remember and what to forget, making it more adaptable and powerful.
- **Wide Applicability:** LSTMs have been successfully applied to a wide range of tasks, including machine translation, speech recognition, and sentiment analysis, due to their ability to model sequences effectively.

Bidirectional RNNs

While Long Short-Term Memory (LSTM) networks are very powerful at learning from past information and retaining important patterns, there are certain scenarios in which simply looking at past information is not sufficient to make accurate predictions. In many real-world applications, understanding both the past and the future context is crucial to making well-informed decisions. For instance, in language modeling tasks, such as Named Entity Recognition (NER) or speech recognition, the context surrounding a word, both from preceding and succeeding words, plays a vital role in understanding its meaning.

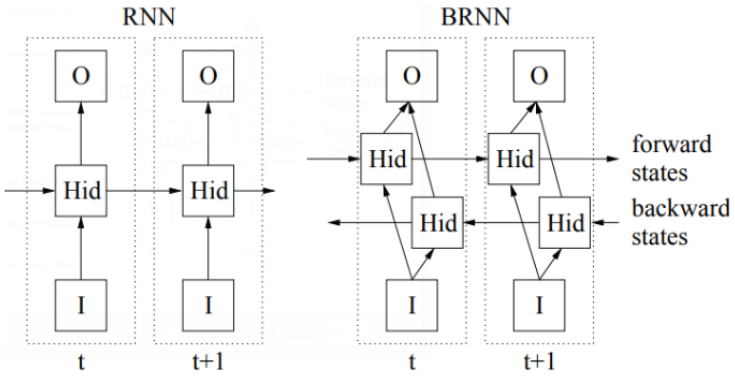
This is where Bidirectional Recurrent Neural Networks (BRNNs) come into play. In a Bidirectional RNN, the model is designed to process the input sequence in both directions: from the past (forward direction) and from the future (backward direction). By doing so, the network is able to capture richer contextual information, which makes it more adept at making predictions where understanding the full sequence, both backward and forward, is necessary.

In a Bidirectional RNN, two separate RNN layers are used:

- One layer processes the sequence in the forward direction, meaning it processes the sequence from the start to the end (i.e., from x_1 to x_T).
- The second layer processes the same sequence in the reverse direction, from the end to the start (i.e., from x_T to x_1).

By combining the outputs of both layers at each time step, the model has access to information from both the past (previous time steps) and the future (subsequent time steps). This allows the model to make predictions based not only on the previous context but also by utilizing the upcoming context, which can significantly improve the model's performance, especially in tasks that require a comprehensive understanding of sequences.

For example, in Named Entity Recognition (NER), understanding the word "Apple" in a sentence depends not only on the words before it (such as "company" or "fruit") but also on the words that follow



it (such as "Inc." or "fruit"). In such cases, Bidirectional RNNs allow the model to make better-informed decisions, as they can take both the preceding and succeeding words into account.

Mathematical Representation:

Let's say we have a sequence of inputs $x_1, x_2, \ldots, x_T$. A standard RNN processes the input x_t at each time step t, while a bidirectional RNN processes the input in both directions. For each time step t, we have two states:

$$h_t^{\text{forward}} = f_{\text{forward}}(x_t)$$
$$h_t^{\text{backward}} = f_{\text{backward}}(x_{T-t})$$

Where:

- f_{forward} is the function for the forward pass of the RNN.
- f_{backward} is the function for the backward pass of the RNN.

The final output for time step t is the combination of these two:

$$h_t = [h_t^{\text{forward}}, h_t^{\text{backward}}]$$

This way, the model captures information from both past and future, enriching its understanding of the sequence.

Key Takeaways

- **LSTMs Address RNN Limitations:** LSTMs effectively solve the vanishing gradient problem of traditional RNNs, enabling the capture of long-term dependencies in sequential data.
- **Three Critical Gates in LSTM:** LSTM cells utilize three essential gates—forget, input, and output—that control the flow of information, allowing the model to decide what to remember, forget, and output at each step.
- **Efficient Memory Management:** The LSTM memory cell updates by combining retained memory with newly learned information, allowing for the long-term retention of crucial sequence data.
- **Benefits of LSTMs:** LSTMs maintain memory over extended sequences, mitigate gradient-related issues, and offer fine-grained control over information retention, making them ideal for complex, long-range tasks.
- **Bidirectional RNNs Enhance Context Understanding:** By processing sequences in both forward and backward directions, Bidirectional RNNs improve context comprehension, making them particularly effective in tasks like Named Entity Recognition (NER).
- **Broad Applicability in Sequence Modeling:** LSTMs and Bidirectional RNNs are widely used in real-world applications such as machine translation, sentiment analysis, and speech recognition, where understanding long-range dependencies and future context is essential.