# LSTM Introduction

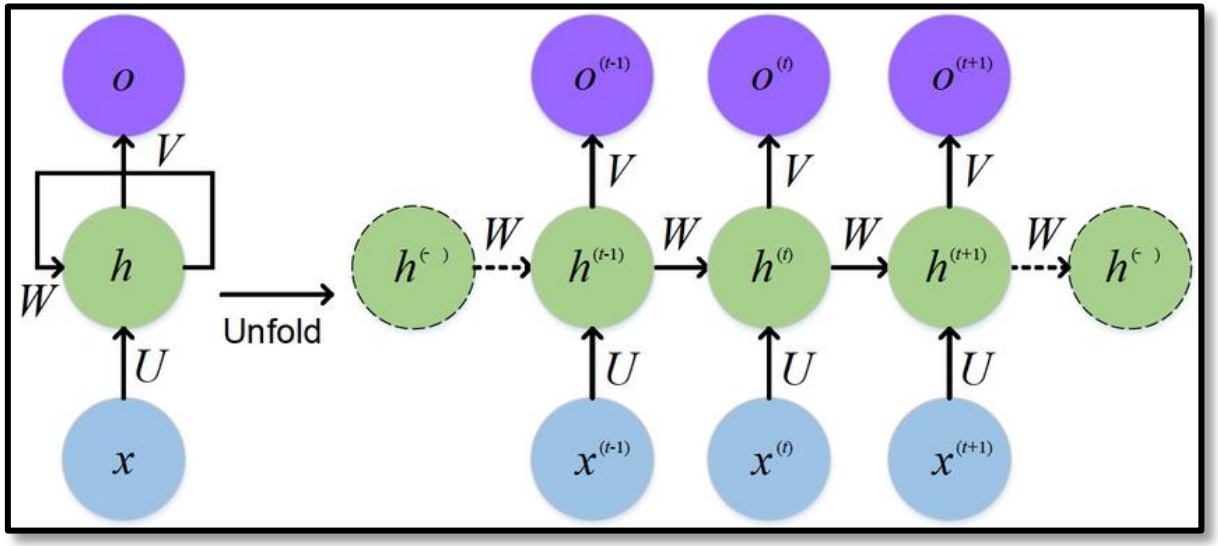
# LSTM Introduction

## RNN Recap: A Quick Overview

In previous presentations, we studied Recurrent Neural Networks (RNNs). These types of models are used when we have sequential data as our input. The cells in these architectures are designed to capture the relationships between different elements of the sequence.

We also examined the feedforward process and the backpropagation through time process in these networks. We understood that the output of each time step depends on the hidden state of the previous time step.

Additionally, we went through a step-by-step process of preparing the data and building an RNN model using a practical example.




Source: www.researchgate.net

# LSTM Introduction

## Limitations of RNNs

One of the challenges when training deep neural networks is the vanishing gradient problem. This occurs because we calculate the gradient with respect to the weights and biases of the first layers through numerous multiplications with numbers smaller than 1. The number of multiplications has a linear relationship with the number of layers in the network.

In RNNs, due to the backpropagation through time process, the number of multiplications also has a linear relationship with the number of cells in the layer. This means that these networks are more susceptible to vanishing and exploding gradients.

The vanishing gradient problem arises when the gradients become extremely small, making it difficult for the network to learn effectively, especially in the earlier layers.

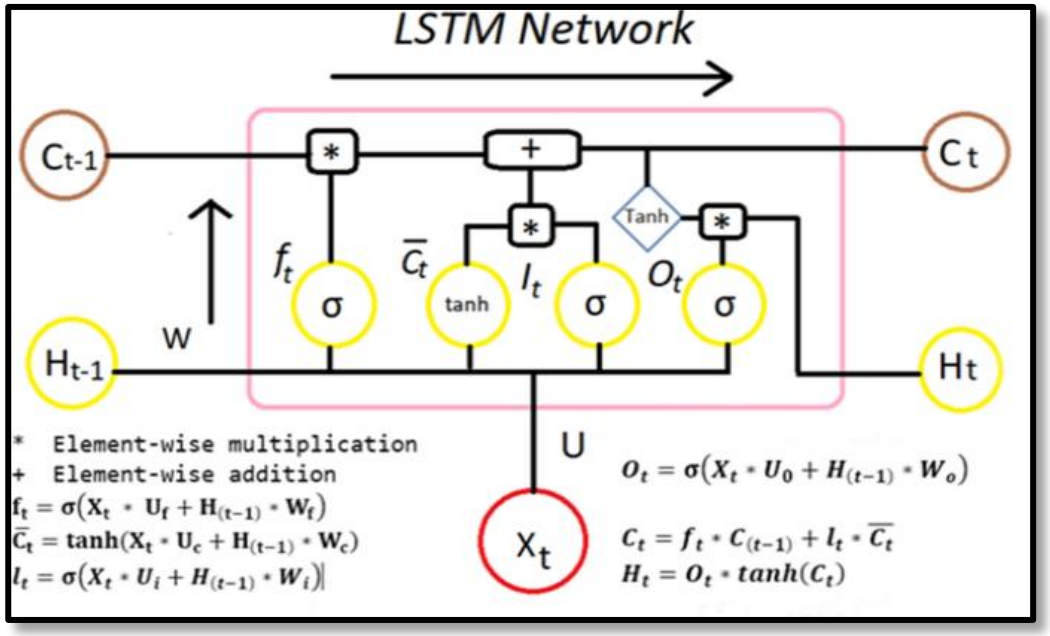
Another issue related to the vanishing gradient problem is that RNNs struggle to capture long-term relationships between elements effectively. This limitation occurs because the gradients diminish progressively after passing through each cell. As a result, two cells that are distant from each other in the sequence do not have a significant impact on each other.

# LSTM Introduction

## LSTM

A new architecture of cells was designed to address the limitations of simple RNNs. This new architecture aims to overcome both the incapacity to capture long-term dependencies and the vanishing gradient problem.

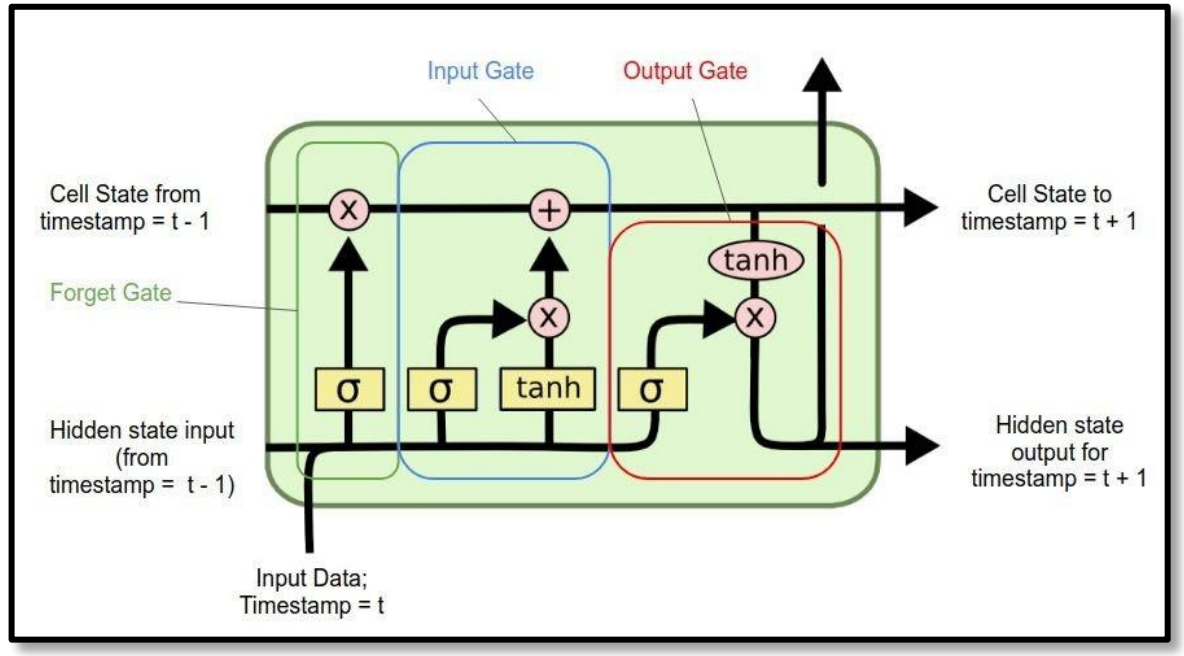
Long Short-Term Memory (LSTM) is a modified RNN architecture that specifically tackles the issues of vanishing and exploding gradients. It is particularly effective for training over long sequences while retaining memory.




Source: www.researchgate.net

## LSTM Architecture

In LSTMs, a set of gates is utilized to control when information enters and exits memory. These gates are the core of the LSTM architecture. In an LSTM, we use the previous cell's state, output, and the new input to generate the current cell's state and output.
The forget gate helps the LSTM selectively retain or discard information from the previous state, allowing it to maintain long-term dependencies. The input gate controls what new information from the current input and previous output should be added to the current state. Finally, the output gate generates the output based on the updated state, considering both the current input and the retained information from the previous state.




Source: www.medium.com

## Forget Gate

The first gate in an LSTM cell is the forget gate. In this gate, we generate a matrix that determines what information to keep and what to discard.

To do this, we first compute the weighted sum of the new input and the previous output. Specifically, we multiply the new input by a set of weights and add it to the multiplication of the previous output by another set of weights. Next, we add a set of biases and apply the sigmoid activation function. This process yields a matrix with the same shape as the previous cell state. Due to the properties of the sigmoid activation function, all the values in this new matrix will be between 0 and 1. We then perform an element-wise multiplication between this new matrix and the previous cell state to generate the updated cell state.

If the corresponding value in the matrix is 0, it indicates that the associated information will be completely forgotten after the multiplication.

Conversely, if the corresponding value is 1, it signifies that the information should be completely retained.

This selective process is why it is called the forget gate; it allows the LSTM to control which information is discarded and which is preserved.

## Input Gate

The second gate in an LSTM cell is the input gate. This gate is responsible for determining which new information should be added to the cell state.

To accomplish this, we first compute the weighted sum of the new input and the previous output, similar to the forget gate. After this, we add a set of biases and apply the sigmoid activation function. This results in a matrix that indicates which values will be updated in the cell state.

In parallel, we also create a candidate cell state vector by applying the hyperbolic tangent (tanh) activation function to the same weighted sum of the new input and previous output. This candidate vector represents the potential new information that could be added to the cell state.

Next, we perform an element-wise multiplication between the output of the sigmoid function (from the input gate) and the candidate cell state. This operation selectively filters the candidate values, allowing only the relevant information to be added to the cell state.

If the corresponding value in the input gate matrix is 0, it shows that the associated new information will not be added to the cell state. Conversely, if the value is 1, it signifies that the information should be fully incorporated.

## Output Gate

The third and final gate in an LSTM cell is the output gate. This gate determines what information from the current input and previous state will be used to generate the output for the current time step.

Similar to the forget and input gates, we first compute the weighted sum of the new input and the previous output. We multiply the new input by a set of weights and add it to the multiplication of the previous output by another set of weights. After adding a set of biases, we apply the sigmoid activation function to produce a matrix that indicates which values from the cell state will be used to generate the output.

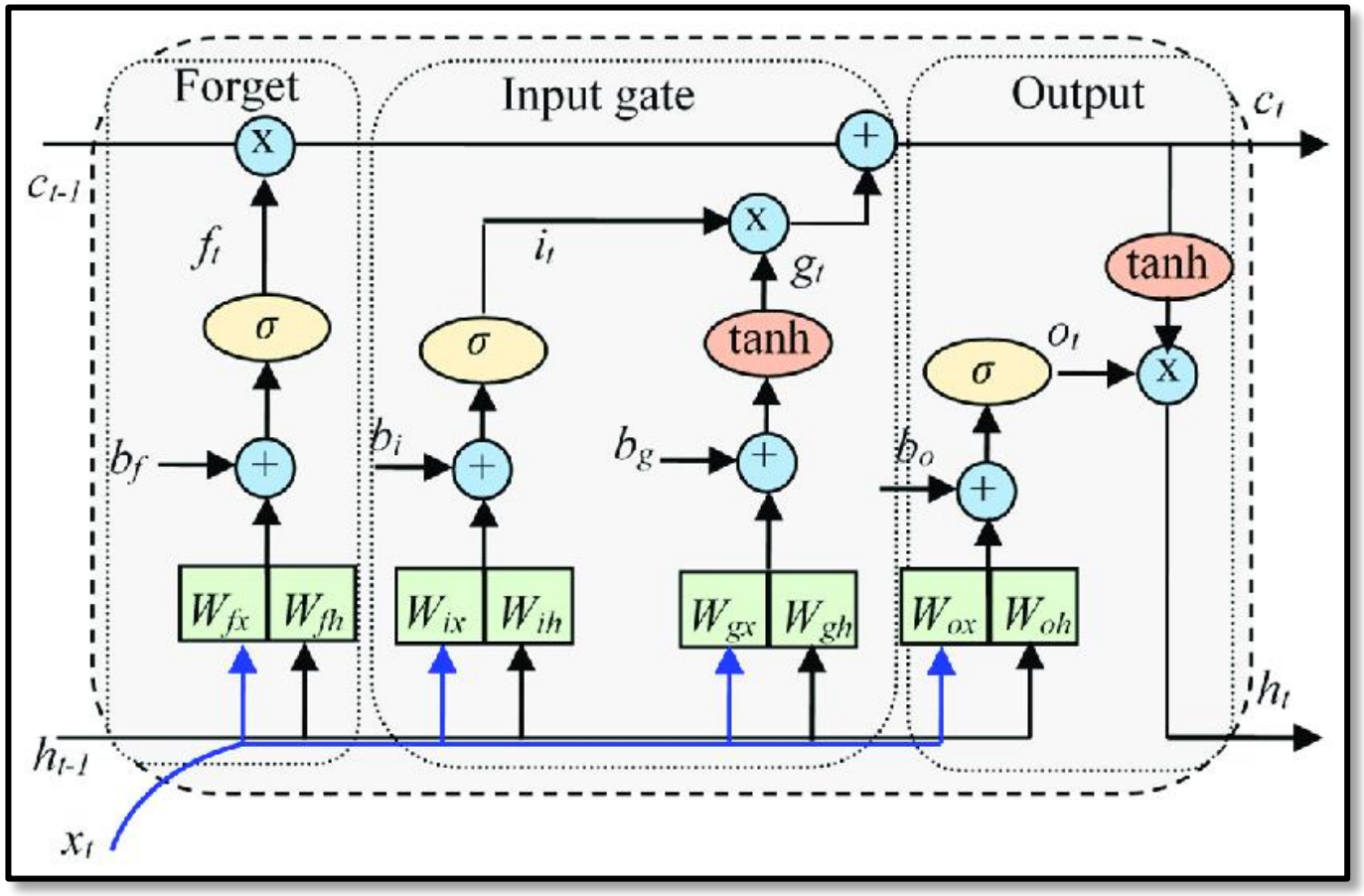
In parallel, we apply the hyperbolic tangent (tanh) activation function to the current cell state. This step scales the cell state values to be between -1 and 1, ensuring that the output is also within this range.

Next, we perform an element-wise multiplication between the output of the sigmoid function and the scaled cell state. This operation selectively filters the cell state values, allowing only the relevant information to contribute to the output.

Finally, the resulting output is passed to the next time step as the previous output and is also used as the output for the current time step.

## LSTM Cell

In this figure, we can see how all the gates work together to form the LSTM cell.
The forget gate is located on the left and determines which parts of the cell state should be discarded and which should be retained.
The input gate is positioned in the middle of the cell. Here, we generate a candidate input using the tanh activation function. The input gate then selects which candidate values to add to the current cell state based on the output of the sigmoid function.
Finally, the output gate is shown on the right. This gate generates the output from the cell state and selects which values to use based on the output of the sigmoid function.

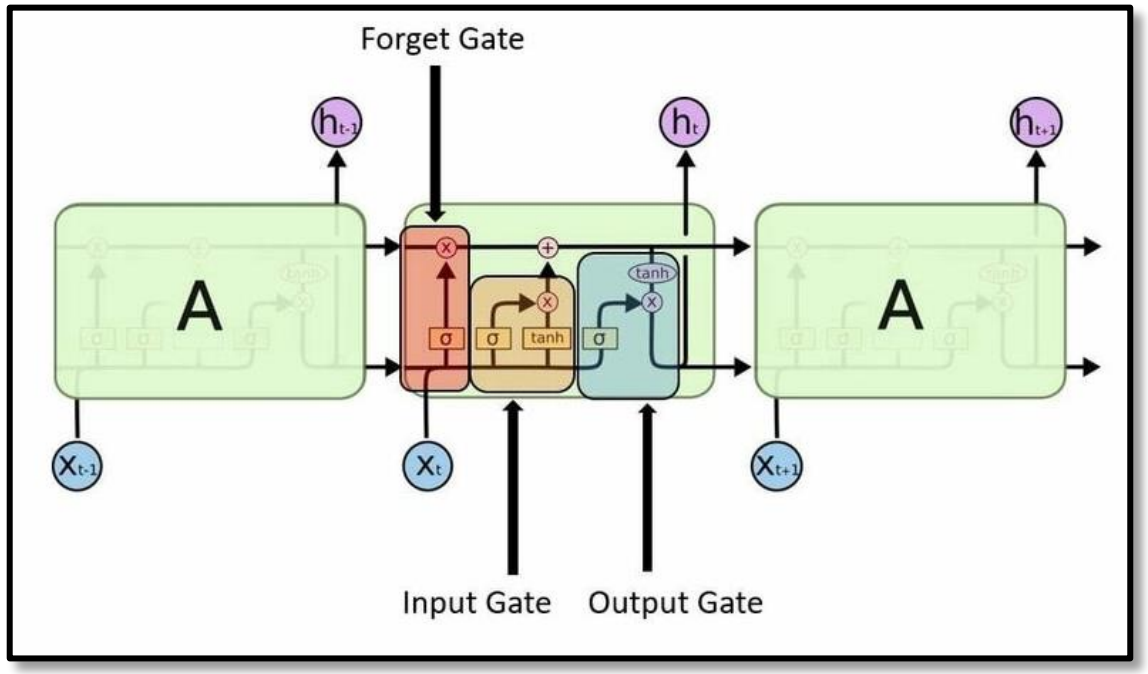



Source: www.researchgate.net

## LSTM Layer

We can use several LSTM cells to form an LSTM layer. Similar to RNNs, the ability to pass the current cell's state to the next cell enables the model to learn the relationships among different elements in the sequence.

What differentiates LSTMs from traditional RNNs is the presence of gates in the architecture. These gates help prevent the vanishing gradient problem by reducing the decay effect of the gradient during backpropagation through time. As a result, LSTMs are better equipped to capture long-term dependencies in the data.




Source: www.researchgate.net

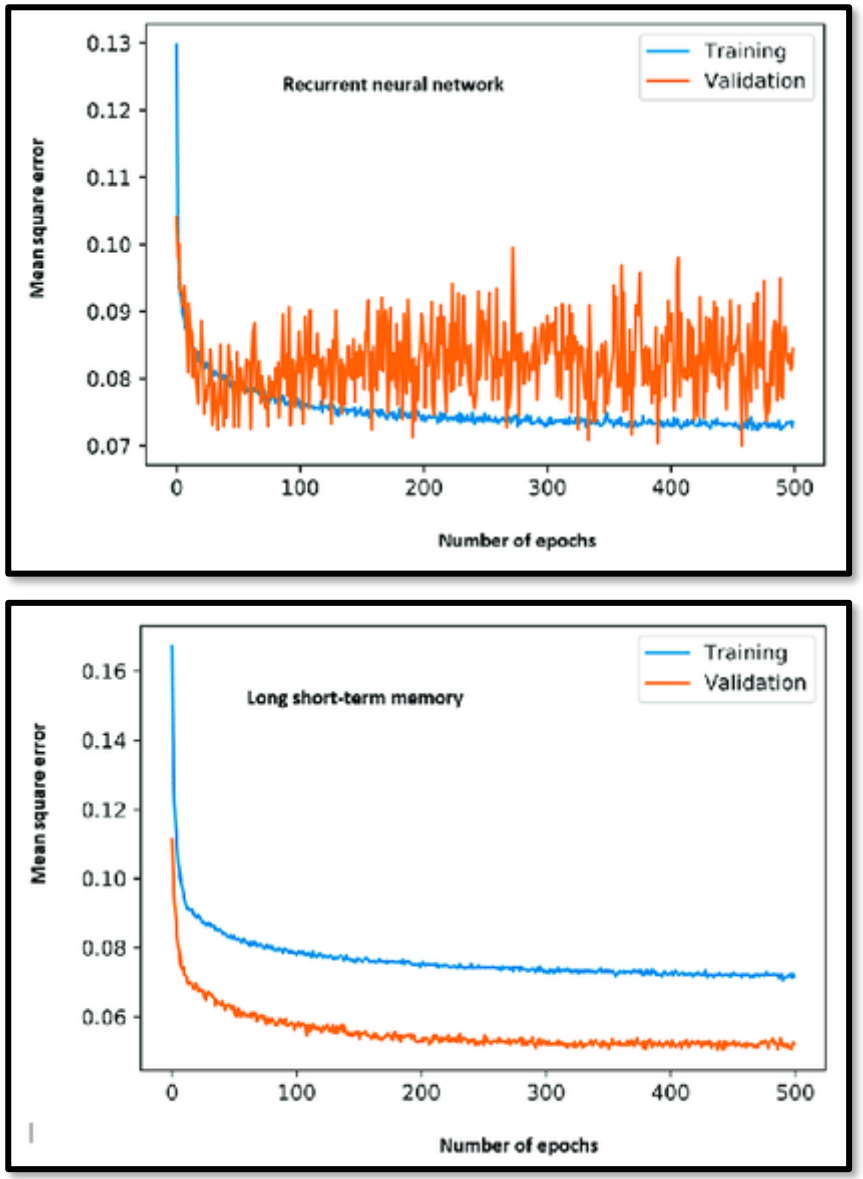
# LSTM Introduction

## Training Process

Through the use of different gates, LSTMs typically exhibit a smoother learning curve compared to traditional RNNs. While these models have a higher number of parameters, which may result in a longer time to reach an acceptable loss, once they achieve that stage, their learning curve becomes very smooth and experiences minimal fluctuations.

In contrast, the accuracy of traditional RNNs often fluctuates significantly on certain datasets. This instability can hinder the model's ability to learn effectively over time.

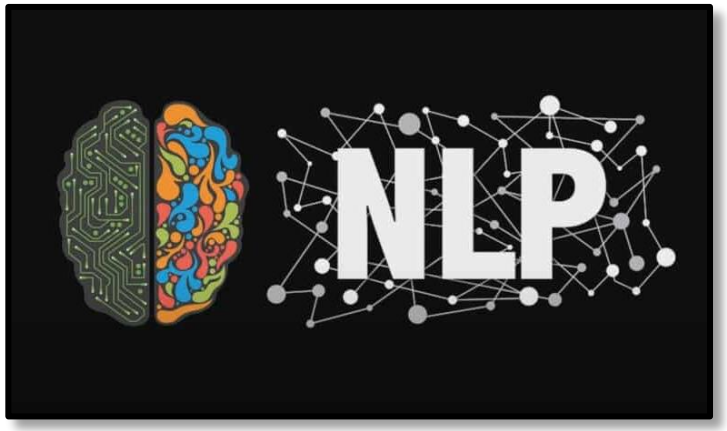
We can clearly observe this difference in the accompanying image, which illustrates the learning curves of LSTMs and traditional RNNs.




Source: www.researchgate.net

## LSTM Applications

The ability of LSTMs to capture long-term dependencies more effectively than traditional RNNs makes them the ideal choice for datasets with these characteristics.
LSTMs excel in tasks such as language modeling, machine translation, and sentiment analysis, where understanding context over long sequences is crucial. They are also widely used in various time series forecasting applications, such as stock price prediction, where capturing long-term patterns is essential for accurate forecasting.



# LSTM Introduction

## LSTM Limitations

Like any other architecture, LSTMs have their own limitations.

**Overfitting**: Without sufficient data, LSTMs can overfit, failing to generalize well to new, unseen data.

**Modeling Long-Range Dependencies**: Although LSTMs are designed to capture long-term dependencies, they can still struggle with modeling very long-range dependencies in the data.

**Training Challenges**: Due to their sequential nature, LSTMs are difficult to parallelize during training. This limitation restricts the speed-up that can be achieved by utilizing multiple GPUs or other parallel hardware, potentially leading to longer training times compared to other architectures that allow for greater parallelization.