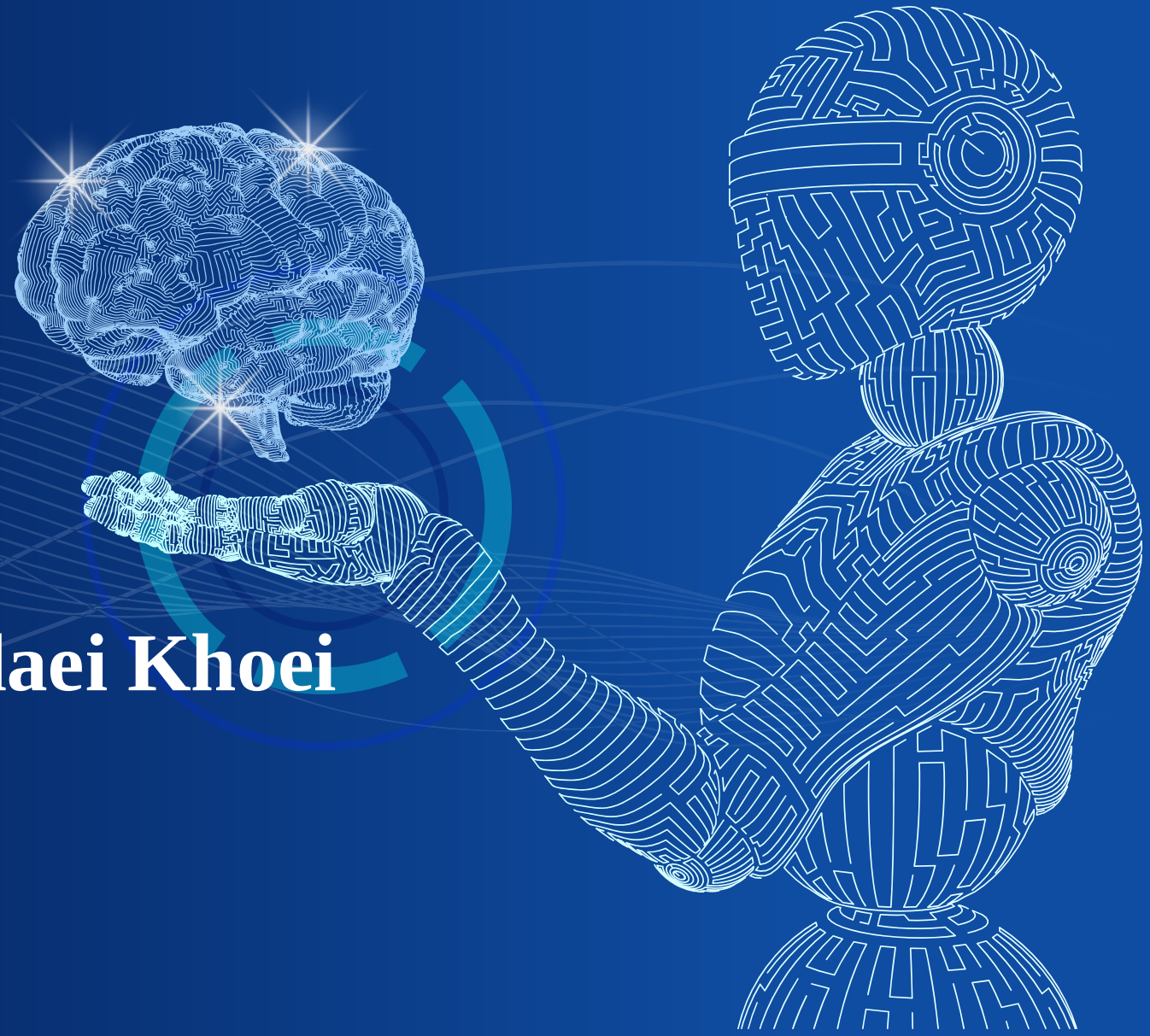
CS 7150: Deep Learning

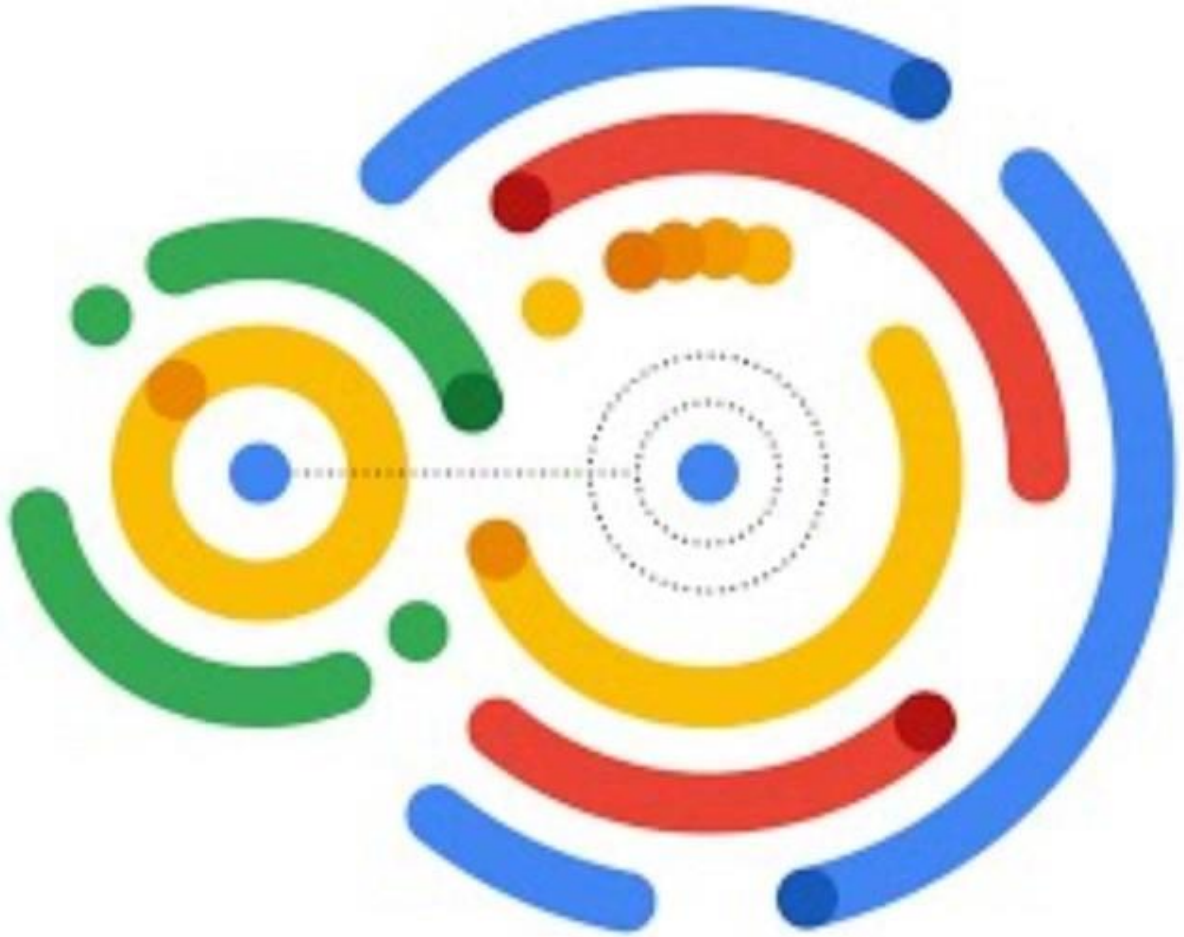


Presented by: Dr. Tala Talaei Khoei

Attention Mechanism

Google Cloud

Attention Mechanism: Overview

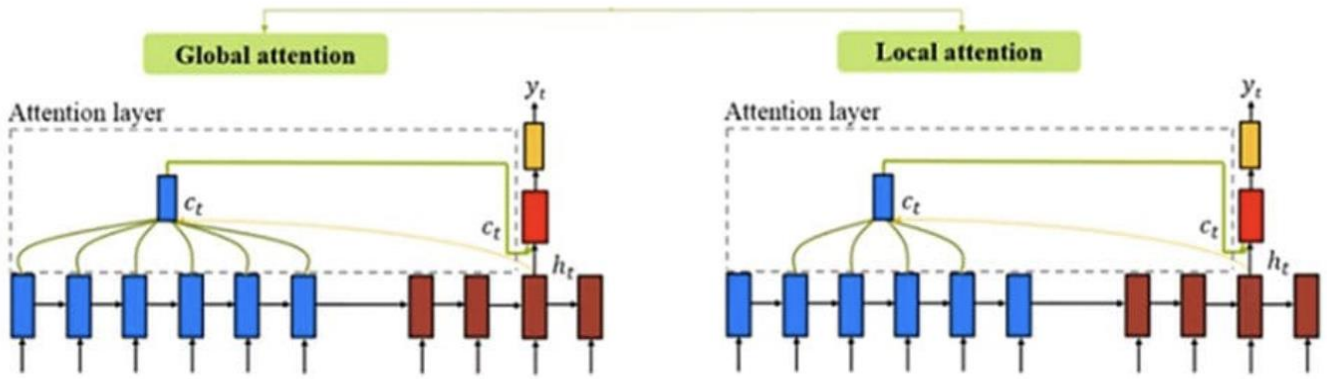


Attention Mechanism

- An attention mechanism is an Encoder-Decoder kind of neural network architecture that allows the model to focus on specific sections of the input while executing a task. It dynamically assigns weights to different elements in the input, indicating their relative importance or relevance.

- By incorporating attention, the model can selectively attend to and process the most relevant information, capturing dependencies and relationships within the data. This mechanism is particularly valuable in tasks involving sequential or structured data, such as natural language processing or computer vision, as it enables the model to effectively handle long-range dependencies and improve performance by selectively attending to important features or contexts.

- Depending on how many source states contribute while deriving the attention vector (α), there can be two types of attention mechanisms:
  - **Global Attention**: When attention is placed on all source states. In global attention, we require as many weights as the source sentence length.
  - **Local Attention**: When attention is placed on a few source states.



The key idea behind attention mechanisms is to enable the model to focus on specific parts of the input sequence that are most relevant for making predictions. Rather than treating all input elements equally, attention allows the model to assign different weights or importance to different elements, depending on their relevance.

To gain a better understanding of how attention works, let's visualize an example. Consider a time-series dataset containing daily stock prices over several years. We want to predict the stock price for the next day. By applying attention mechanisms, the model can learn to focus on specific patterns or trends in the historical prices that are likely to impact the future price.

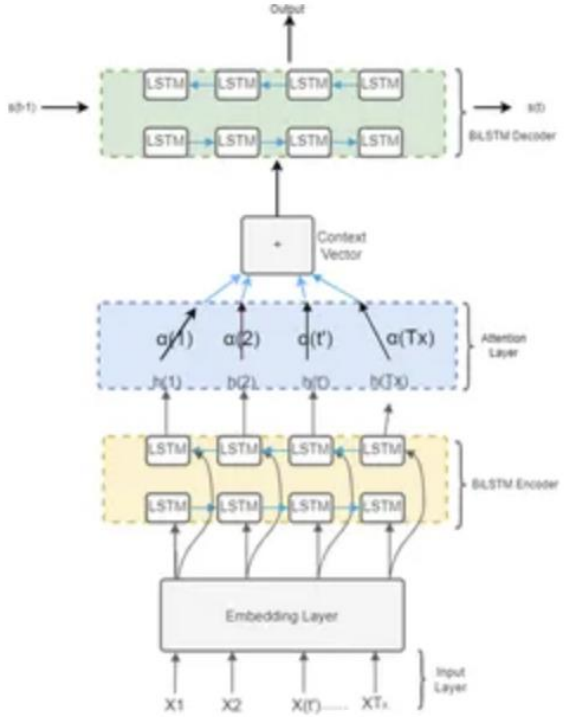
In the visualization provided, each time step is depicted as a small square, and the attention weight assigned to that specific time step is indicated by the size of the square. We can observe that the attention mechanism assigns higher weights to the recent prices, indicating their increased relevance for predicting the future price.

Attention-Based Time-Series Forecasting Models

One popular approach is to combine attention with recurrent neural networks (RNNs), which are widely used for sequence modeling.

Encoder-Decoder Architecture

The encoder-decoder architecture consists of two main components: the encoder and the decoder. Let's denote the historical input sequence as *X = [X1, X2, …, XT]*, where *Xi* represents the input at time step *i*.

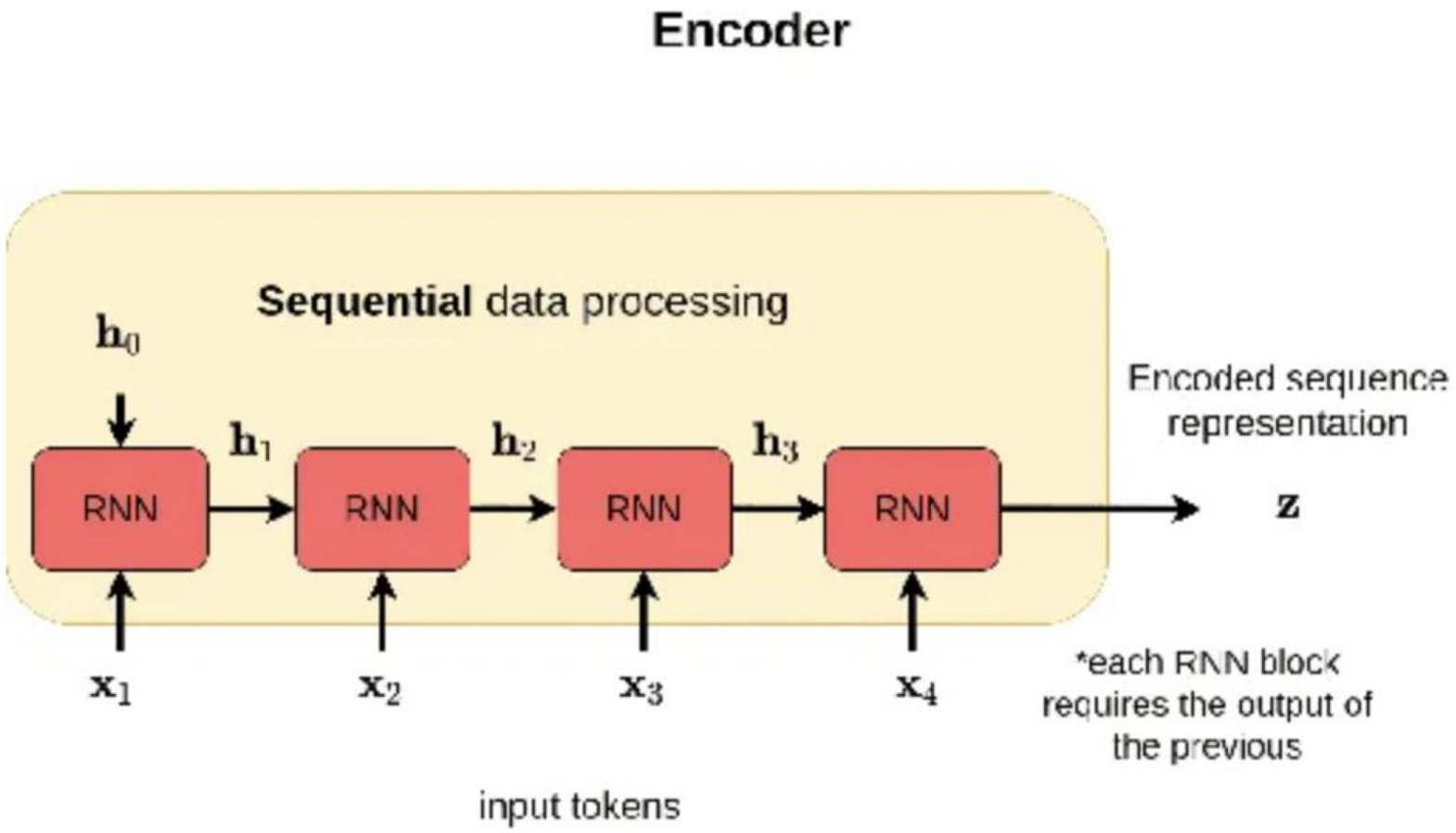


Encoder

The encoder processes the input sequence X and captures the underlying patterns and dependencies. In this architecture, the encoder is typically implemented using an LSTM (Long Short-Term Memory) layer. It takes the input sequence *X* and produces a sequence of hidden states *H = [H1, H2, …, HT]*. Each hidden state *Hi* represents the encoded representation of the input at time step *i*.

H, _ = LSTM(X)

Here, *H* represents the sequence of hidden states obtained from the LSTM layer, and "_" denotes the output of the LSTM layer that we don't need in this case.



Decoder

The decoder generates the forecasted values based on the attention-weighted encoding and the previous predictions.

- **LSTM Layer:**

The decoder takes the previous predicted value (*prev_pred*) and the context vector (*Context*) obtained from the attention mechanism as input. It processes this input using an LSTM layer to generate the decoder hidden state (*dec_hidden*):

dec_hidden, _ = LSTM([prev_pred, Context])

Here, *dec_hidden* represents the decoder hidden state, and "_" represents the output of the LSTM layer that we don't need.

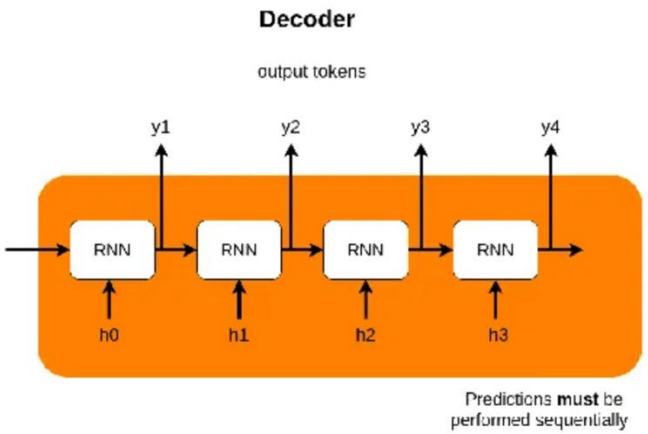
- **Output Layer:**

The decoder hidden state (*dec_hidden*) is passed through an output layer to produce the predicted value (pred) for the current time step:

pred = OutputLayer(dec_hidden)

The OutputLayer applies appropriate transformations and activations to map the decoder hidden state to the predicted value.

By combining the encoder and decoder components, the encoder-decoder architecture with attention allows the model to capture dependencies in the input sequence and generate accurate forecasts by considering the attention-weighted encoding and previous predictions.



Advantages of Attention Mechanisms in Time-Series Forecasting

Incorporating attention mechanisms into time-series forecasting models offers several advantages:

1. Capturing Long-Term Dependencies

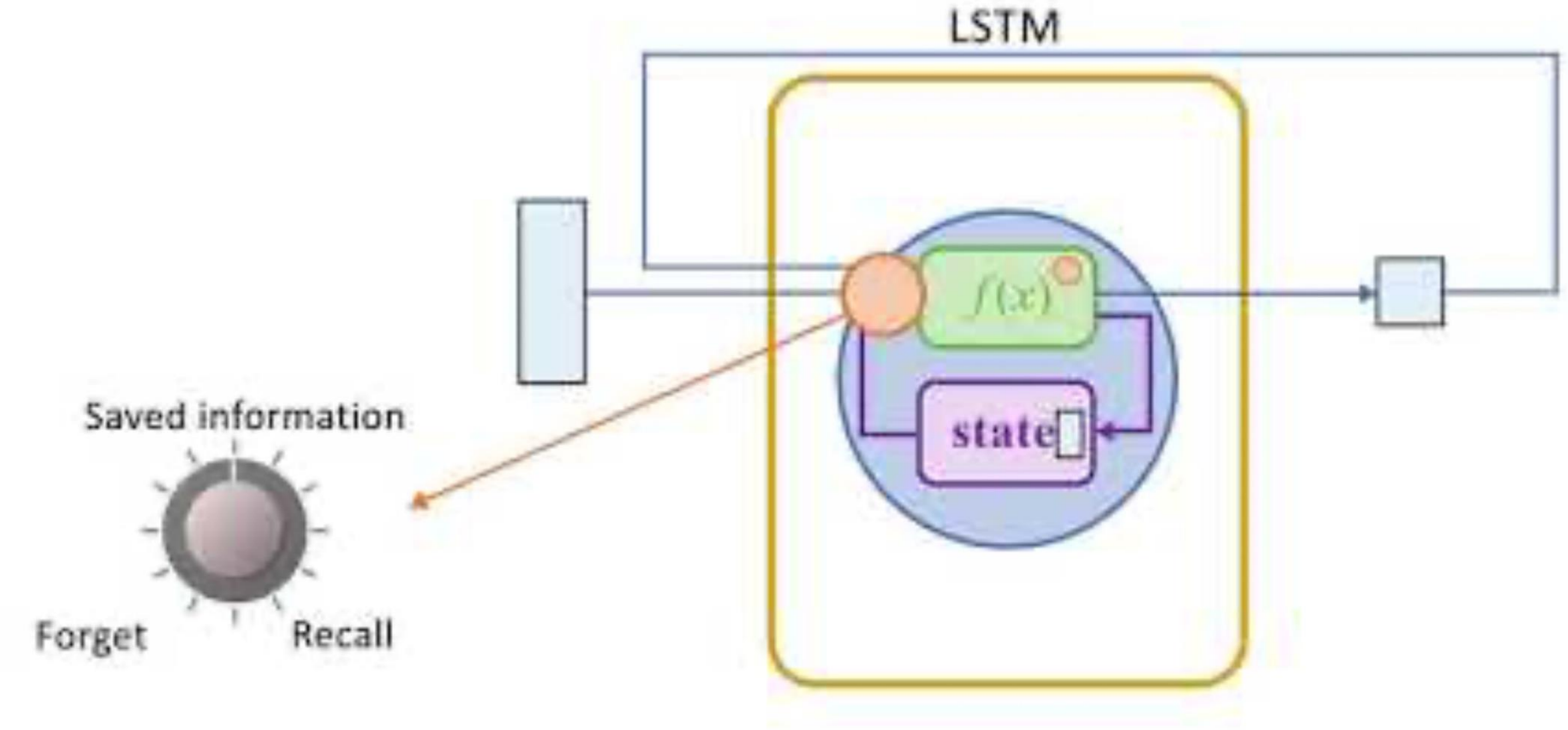
Attention mechanisms allow the model to capture long-term dependencies in time-series data. Traditional models like ARIMA have limited memory and struggle to capture complex patterns that span across distant time steps. Attention mechanisms provide the ability to focus on relevant information at any time step, regardless of its temporal distance from the current step.

2. Handling Irregular Patterns

Time-series data often contains irregular patterns, such as sudden spikes or drops, seasonality, or trend shifts. Attention mechanisms excel at identifying and capturing these irregularities by assigning higher weights to the corresponding time steps. This flexibility enables the model to adapt to changing patterns and make accurate predictions.

3. Interpretable Forecasts

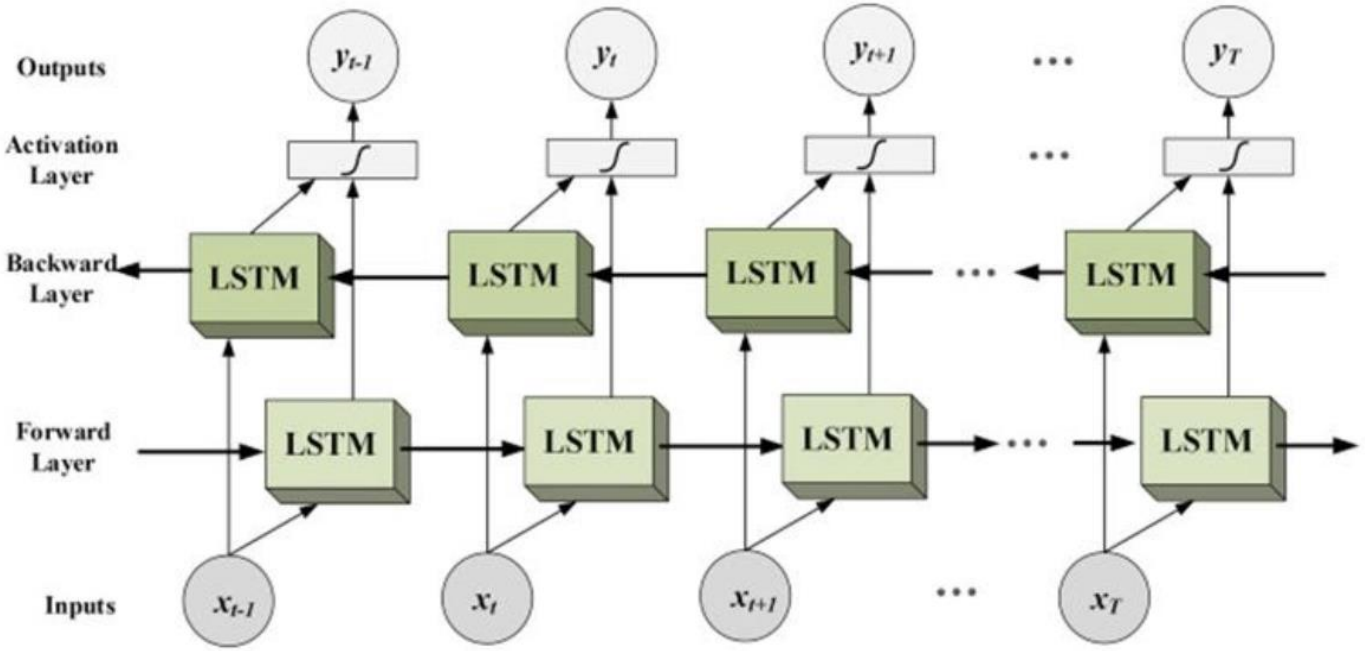
Attention mechanisms provide interpretability to time-series forecasting models. By visualizing the attention weights, users can understand which parts of the historical data are most influential in making predictions. This interpretability helps in gaining insights into the driving factors behind the forecasts, making it easier to validate and trust the model’s predictions.


LSTM
f(x)
state
Saved information
Forget
Recall

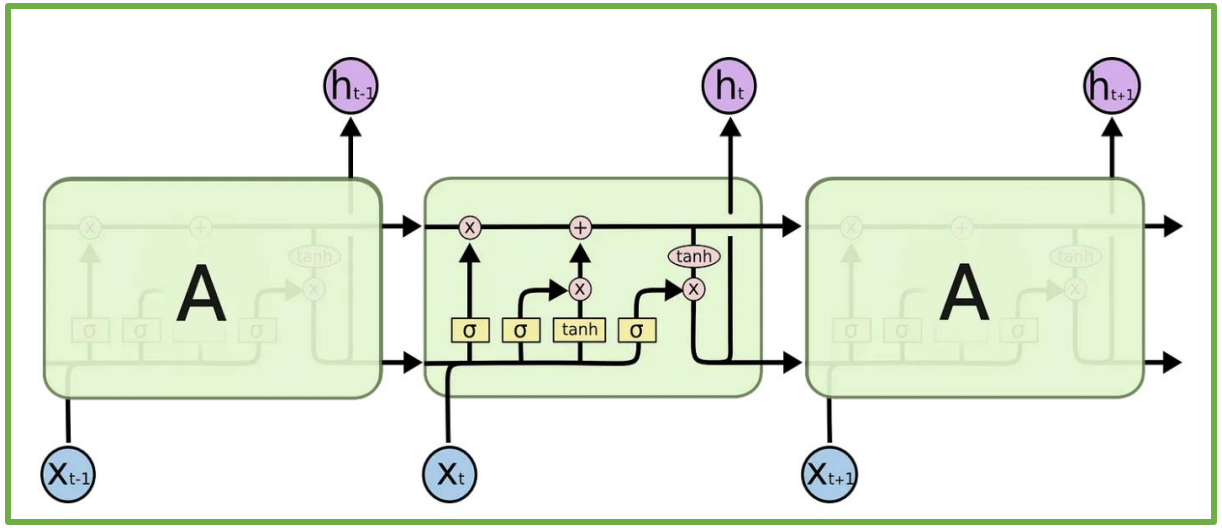
Bidirectional LSTM

- Bi-LSTM (Bidirectional Long Short-Term Memory) is a type of recurrent neural network (RNN) that processes sequential data in both forward and backward directions. It combines the power of LSTM with bidirectional processing, allowing the model to capture both past and future context of the input sequence.

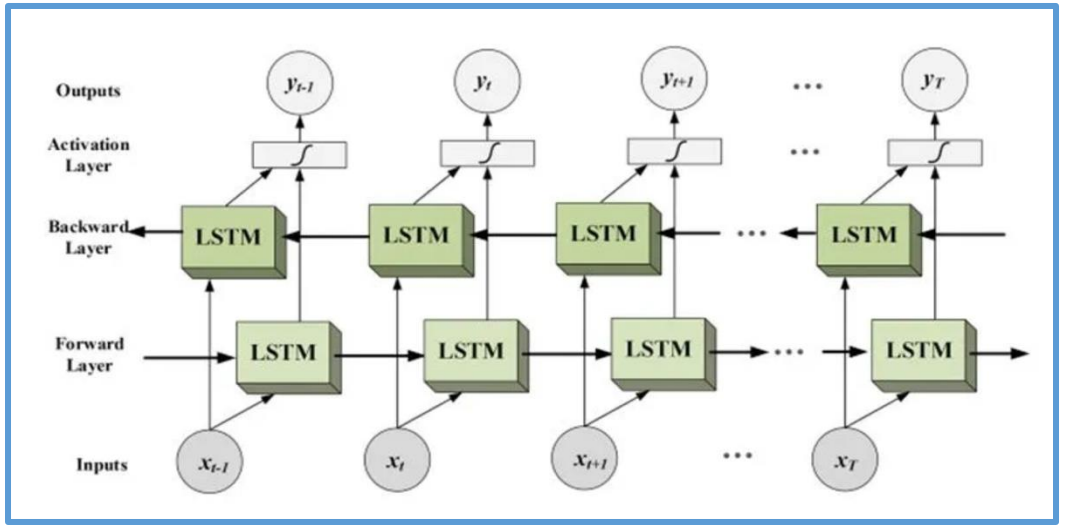
- Bidirectional long-short term memory(Bi-LSTM) is the process of making any neural network o have the sequence information in both directions backwards (future to past) or forward(past to future). In bidirectional, our input flows in two directions, making a Bi-LSTM different from the regular LSTM. With the regular LSTM, we can make input flow in one direction, either backwards or forward. However, in bi-directional, we can make the input flow in both directions to preserve the future and the past information.



LSTM


h_{t-1}
h_t
h_{t+1}
A
A
σ
σ
tanh
σ
tanh
X_{t-1}
X_t
X_{t+1}

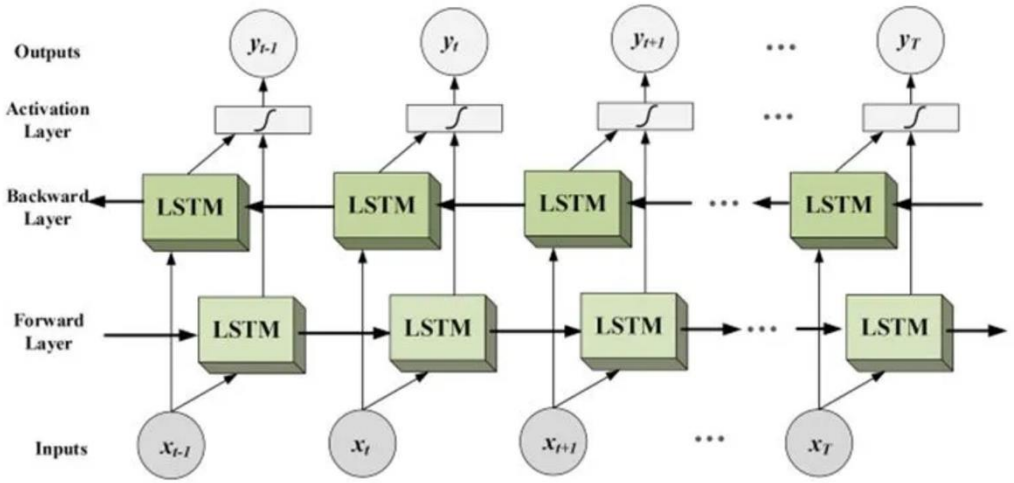
Bi-LSTM


Outputs
Activation Layer
Backward Layer
Forward Layer
Inputs
y_{t-1}
y_t
y_{t+1}
y_T
LSTM
x_{t-1}
x_t
x_{t+1}
x_T

Bi-LSTM

- Now, when we are dealing with long sequences of data and the model is required to learn relationship between future and past word as well. we need to send data in that manner. To solve this problem bidirectional network was introduced. We can use bidirectional network with LSTM.
- In bidirectional LSTM we give the input from both the directions from right to left and from left to right . So, the question is how the data is combined in output if we are having 2 inputs.

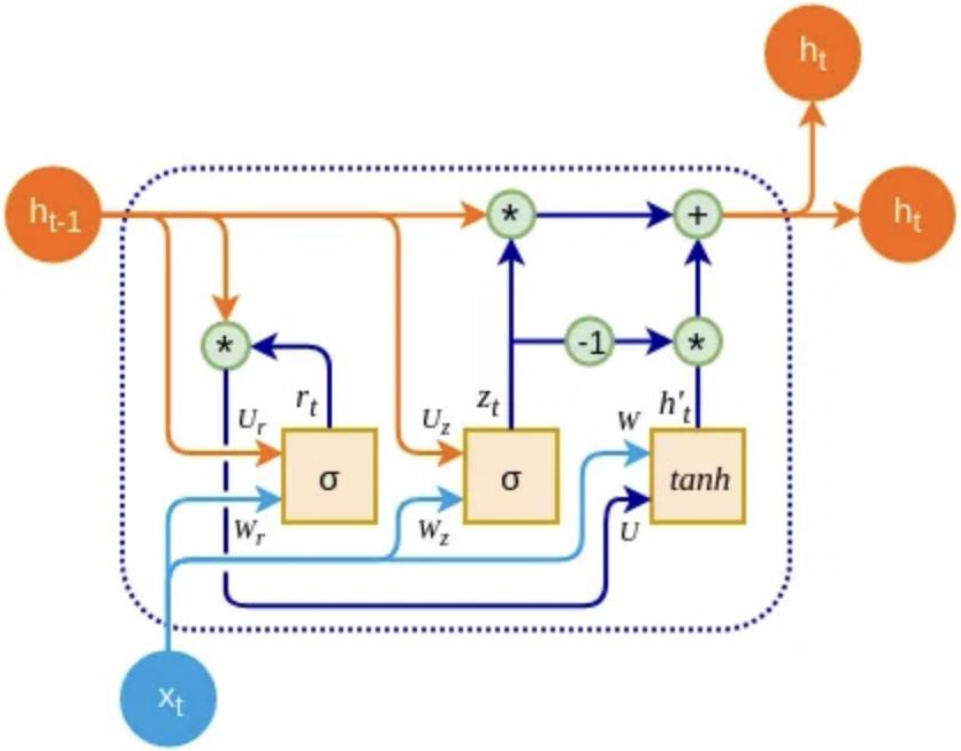
- Generally, in normal LSTM network we take output directly as shown in first figure but in bidirectional LSTM network output of forward and backward layer at each stage is given to activation layer which is a neural network and output of this activation layer is considered. This output contains the information or relation of past and future word also.



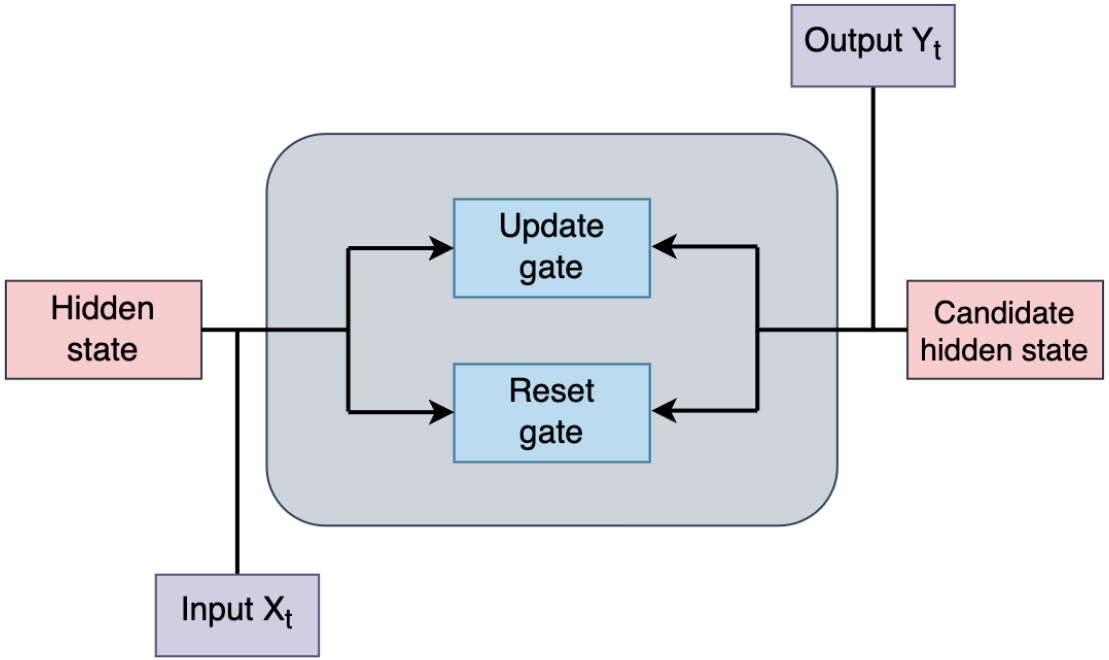
Please refer to Codes on Canvas

Gated Recurrent Unit

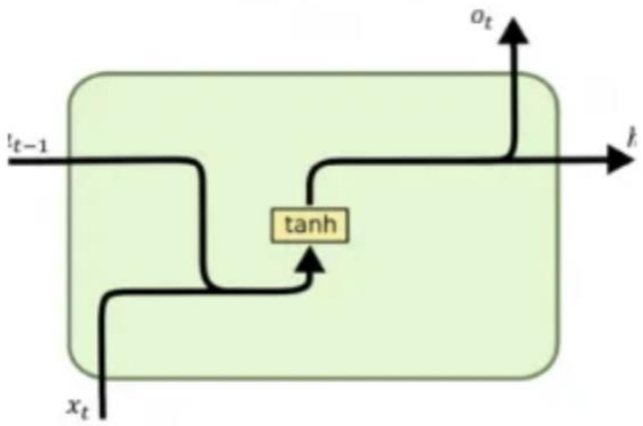
- Introduced by Cho, et al. in 2014, GRU (Gated Recurrent Unit) aims to solve the **vanishing gradient problem** which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results.
- GRU stands for Gated Recurrent Unit, which is a type of recurrent neural network (RNN) architecture that is similar to LSTM (Long Short-Term Memory).



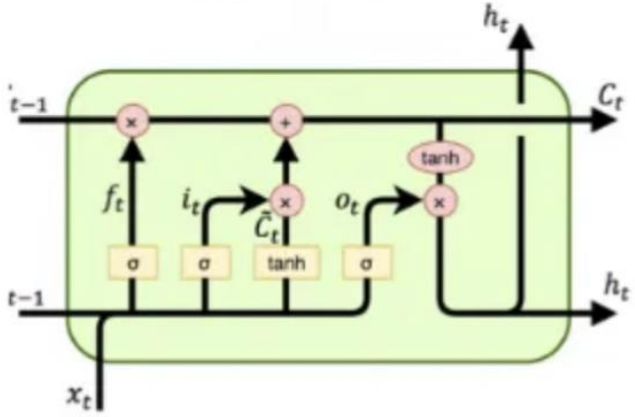
- Like LSTM, GRU is designed to model sequential data by allowing information to be selectively remembered or forgotten over time. However, GRU has a simpler architecture than LSTM, with fewer parameters, which can make it easier to train and more computationally efficient.
- The main difference between GRU and LSTM is the way they handle the memory cell state. In LSTM, the memory cell state is maintained separately from the hidden state and is updated using three gates: the input gate, output gate, and forget gate. In GRU, the memory cell state is replaced with a “candidate activation vector,” which is updated using two gates: the reset gate and update gate.
- The reset gate determines how much of the previous hidden state to forget, while the update gate determines how much of the candidate activation vector to incorporate into the new hidden state.
- Overall, GRU is a popular alternative to LSTM for modeling sequential data, especially in cases where computational resources are limited or where a simpler architecture is desired.



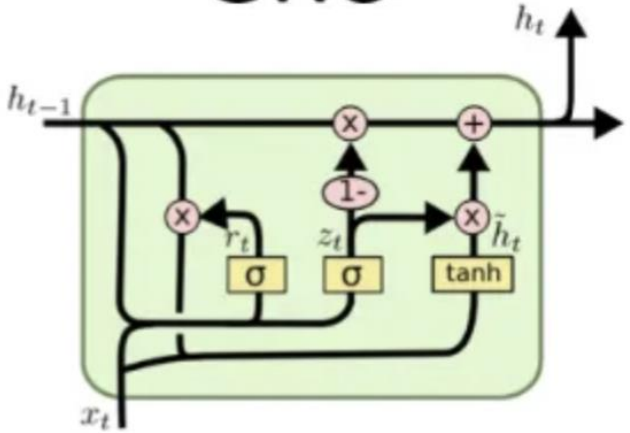
RNN


o_t
h_{t-1}
tanh
x_t

LSTM

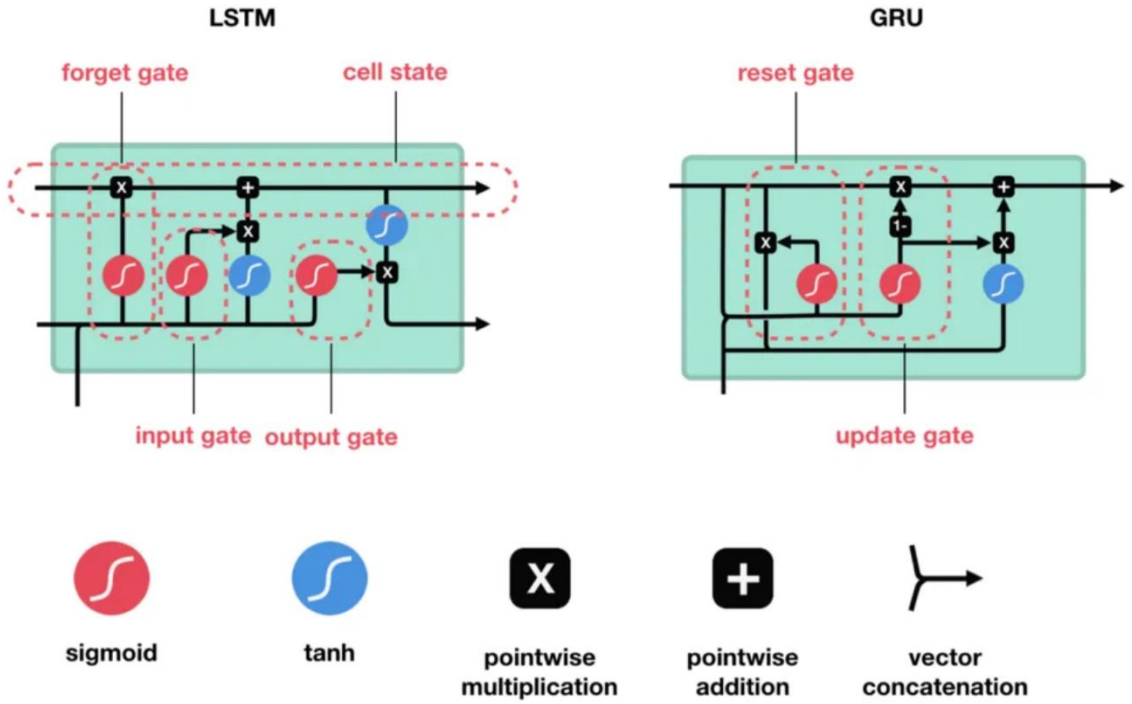

h_t
C_t
C_{t-1}
f_t
i_t
$\tilde{C}_t$
o_t
tanh
σ
σ
tanh
σ
h_{t-1}
h_t
x_t

GRU


h_t
h_{t-1}
1-
r_t
z_t
$\tilde{h}_t$
σ
σ
tanh
x_t

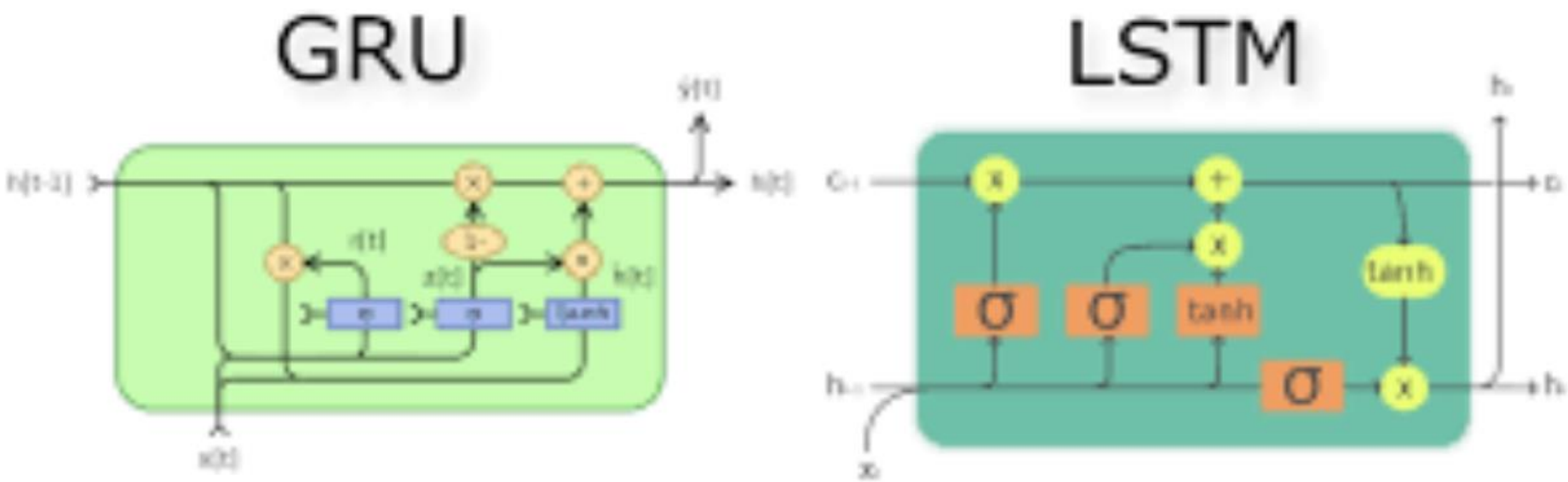
LSTM vs. GRU

- LSTMs and GRUs essentially function just like RNNs, but they're capable of learning long-term dependencies using mechanisms called "gates." These gates are different tensor operations that can learn what information to add or remove to the hidden state. Because of this ability, short-term memory is less of an issue for them.
- The GRU is simpler and enjoys the advantage of greater ease of implementation(fewer tensor operations) and efficiency. It might generalize slightly better with less data because of a smaller parameter footprint, although the LSTM would be preferable with an increased amount of data.
- LSTM directly controls the amount of information changed in the hidden state.



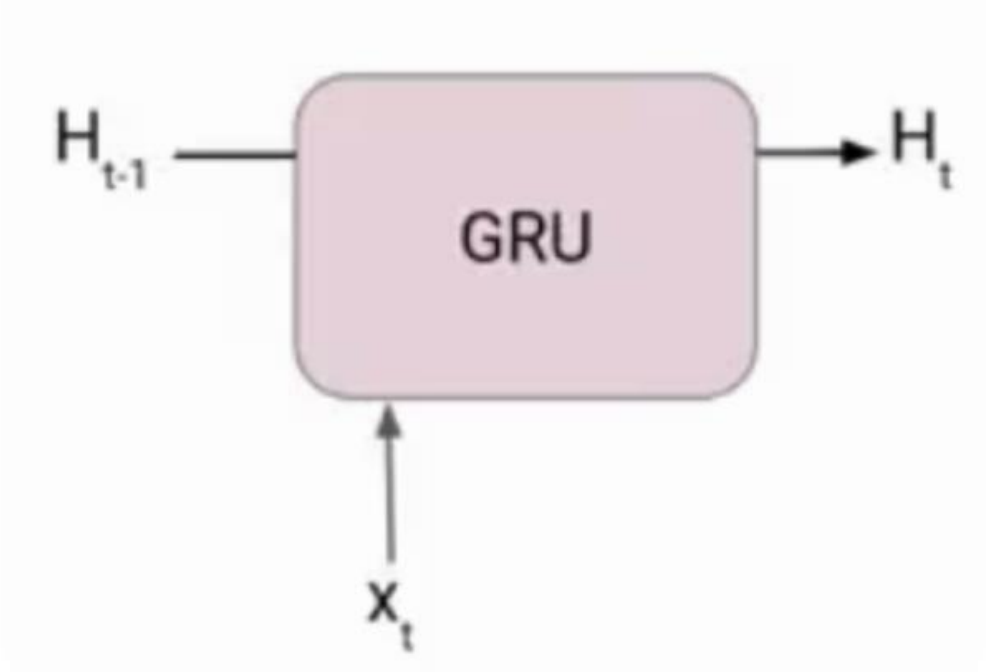
Core Differences between GRU and LSTM:

- The GRU has two gates, and LSTM has three gates.
- GRU does not possess any internal memory, they don't have an output gate that is present in LSTM.
- One feature of the LSTM unit that is missing from the GRU is the controlled exposure of the memory content.
- In the LSTM unit, the amount of the memory content that is seen, or used by other units in the network is controlled by the output gate.
- On the other hand, the GRU exposes its full content without any control.
- The LSTM has been more extensively tested than the GRU, simply because it is an older architecture and enjoys widespread popularity. As a result, it is generally seen as a safer option, particularly when working with longer sequences and larger data sets.



Architecture of Gated Recurrent Unit

- At each timestamp t, it takes an input Xt and the hidden state Ht-1 from the previous timestamp t-1. Later it outputs a new hidden state Ht which again passed to the next timestamp.
- Now there are primarily two gates in a GRU as opposed to three gates in an LSTM cell. The first gate is the Reset gate and the other one is the update gate.



Reset Gate (Short term memory)

- The Reset Gate is responsible for the short-term memory of the network i.e the hidden state (Ht). Here is the equation of the Reset gate.

$$r_t = \sigma(x_t * U_r + H_{t-1} * W_r)$$

- If you remember from the LSTM gate equation it is very similar to that. The value of **rt** will range from 0 to 1 because of the sigmoid function. Here Ur and Wr are weight matrices for the reset gate.

Update Gate (Long Term memory)

Similarly, we have an Update gate for long-term memory and the equation of the gate is shown below.

$$u_t = \sigma(x_t * U_u + H_{t-1} * W_u)$$

The only difference is of weight metrics i.e Uu and Wu.

GRU Architecture

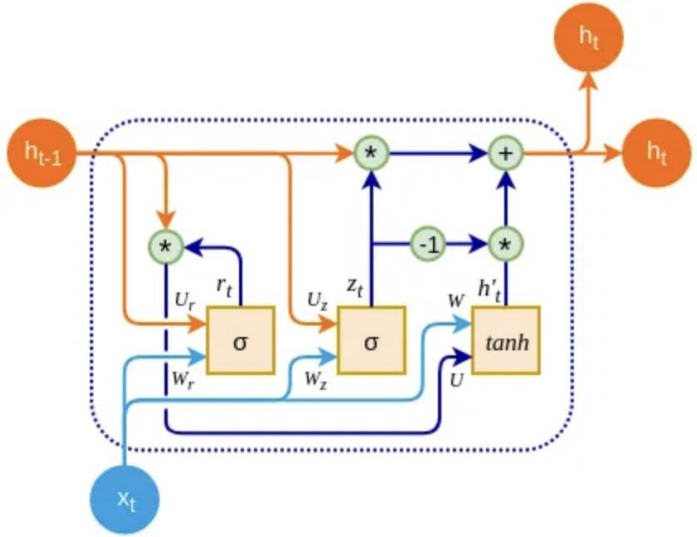
The GRU architecture consists of the following components:

- **Input layer:** The input layer takes in sequential data, such as a sequence of words or a time series of values, and feeds it into the GRU.
- **Hidden layer:** The hidden layer is where the recurrent computation occurs. At each time step, the hidden state is updated based on the current input and the previous hidden state. The hidden state is a vector of numbers that represents the network's "memory" of the previous inputs.
- **Reset gate:** The reset gate determines how much of the previous hidden state to forget. It takes as input the previous hidden state and the current input, and produces a vector of numbers between 0 and 1 that controls the degree to which the previous hidden state is "reset" at the current time step.
- **Update gate:** The update gate determines how much of the candidate activation vector to incorporate into the new hidden state. It takes as input the previous hidden state and the current input, and produces a vector of numbers between 0 and 1 that controls the degree to which the candidate activation vector is incorporated into the new hidden state.
- **Candidate activation vector:** The candidate activation vector is a modified version of the previous hidden state that is "reset" by the reset gate and combined with the current input. It is computed using a tanh activation function that squashes its output between -1 and 1.
- **Output layer:** The output layer takes the final hidden state as input and produces the network's output. This could be a single number, a sequence of numbers, or a probability distribution over classes, depending on the task at hand.

How GRU Works?

- Like other recurrent neural network architectures, GRU processes sequential data one element at a time, updating its hidden state based on the current input and the previous hidden state. At each time step, the GRU computes a “candidate activation vector” that combines information from the input and the previous hidden state. This candidate vector is then used to update the hidden state for the next time step.

- The candidate activation vector is computed using two gates: the reset gate and the update gate. The reset gate determines how much of the previous hidden state to forget, while the update gate determines how much of the candidate activation vector to incorporate into the new hidden state.

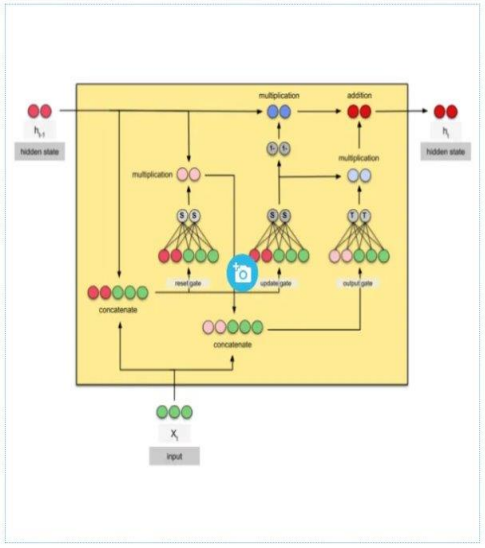


Please refer to Codes on Canvas

Bi-GRU

The main difference between a GRU and a Bi-GRU is that a Bi-GRU has two separate hidden states, one for each direction, and it concatenates the final hidden states from both directions before making its final prediction. This allows the bi-GRU to capture information from both the past and the future of the input sequence, whereas a regular GRU only has access to information from the past.
In practice, Bi-GRUs are often used for tasks such as natural language processing, where the model needs to understand the context of a word in a sentence in order to make accurate predictions.

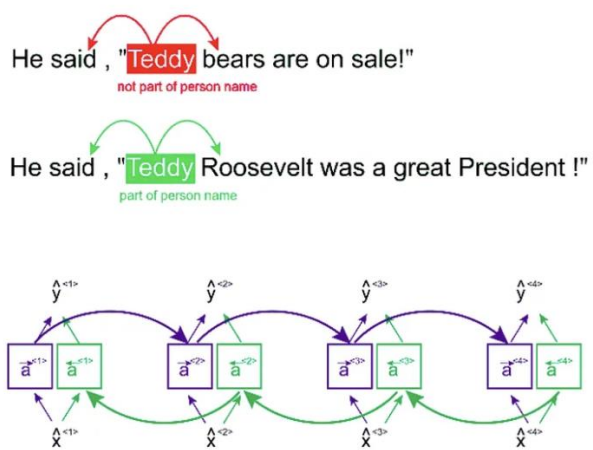
GRU



- The update gate acts similar to the forget and input gate of an LSTM
- The update gate decides what information to throw away and what new information to add
- The reset gate decides how much past information should be retained
- Fewer tensor ops and speedier to train than LSTMs

Bi-GRU

The need for Bi-directional GRUs



- Bi-directional GRUs are just putting two independent GRUs together
- The input sequence is fed in forward order for one GRU, and in reverse order for the other
- The outputs of the two networks are usually concatenated at each time step
- Preserving information from both past and future helps understand context better

Please refer to Codes on Canvas