NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

EE6405

Natural Language Processing

Dr. Simon Liu
NTU Electrical and Electronic Engineering

Neural Language Models

Sequential Data

- Sequential data is organised in a specific order, often with a time-based or chronological sequence.
- The order in which the data points occur is essential for understanding the data's meaning.
- In sequential data, each data point is likely correlated to both the earlier and later data points in the sequence.

Textual Data:

- The meaning of a text often depends on the order of words and the grammatical rules that govern their arrangement.
- Understanding the text often requires knowledge of what came before and what follows.
- The meaning of a word or phrase can change based on the context provided by the surrounding text.

Sequential Data

- Traditional models, such as simple linear regression or basic feedforward neural networks, often struggle with handling sequential data for several reasons:

Lack of Memory

Traditional models do not possess the inherent ability to remember or capture long-term dependencies within the sequence.

Order Insensitivity

Traditional models treat data as unordered, meaning they do not inherently understand the significance of the order of data points in a sequence.

Variable-Length Limitation

Traditional models are designed to work with fixed-length input. However, sequential data often comes in variable lengths.

Sequential Data

- To process sequential data effectively, we need models that can account for the interconnectedness of sequential data.
- Models can handle sequential data through various techniques and architectures designed to capture the temporal dependencies and patterns within the data.
- Some of these models include:

Recurrent Neural Networks (RNNs)

Long Short-Term Memory (LSTM) Networks

Transformers
(To be covered next week)

Gated Recurrent Units (GRUs)

Bi-directional Networks

RNNs

- Recurrent Neural Networks (RNNs) are a type of neural network designed for tasks involving sequences or time series data.
- Unlike traditional feedforward neural networks, RNNs have connections that loop back on themselves, allowing them to maintain memory of previous inputs.
- RNNs can model dependencies over time, making them great for tasks like language translation, speech recognition, and predicting future values in a time series.

Recurrent Neural Networks

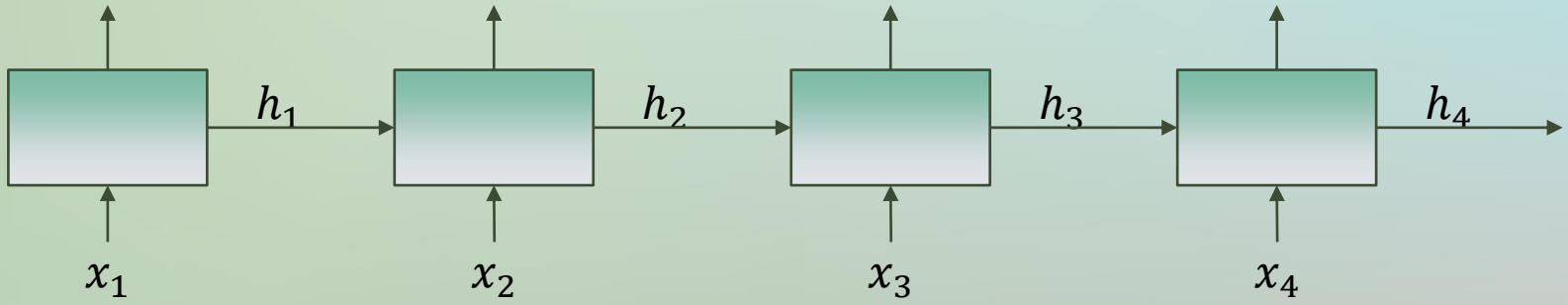
- Each RNN unit computes a new hidden state using the previous state and a new input.

$$h_t = g(x_t, h_{t-1})$$

- Each RNN unit (optionally) makes an output using the current hidden state.

$$y_t = f(h_t)$$

- Hidden states $h_t \in R^D$ are continuous vectors
  - Can represent very rich information
  - Possibly the entire history from the beginning
- Parameters are shared (tied) across all RNN units (unlike feedforward NNs).



Recurrent Neural Networks

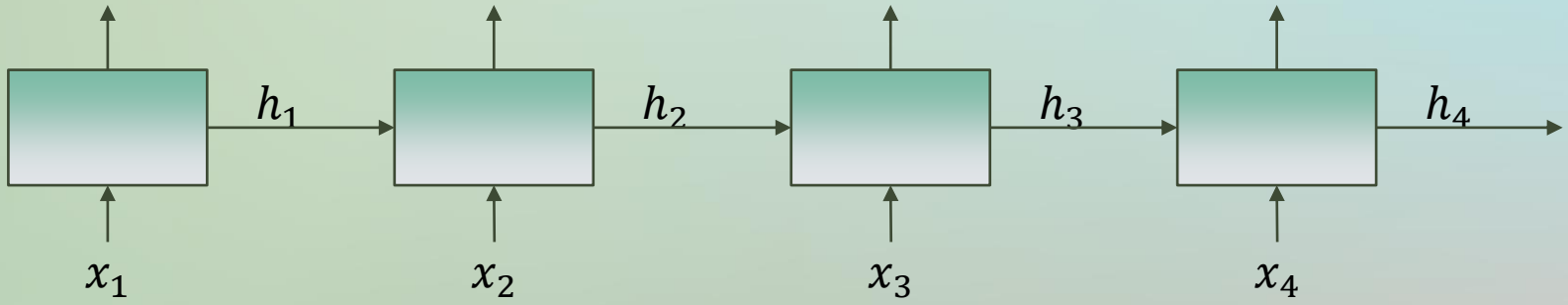
Generic RNNs

- $h_t = g(x_t, h_{t-1})$
- $y_t = f(h_t)$

Vanilla RNNs

- $h_t = tanh(Ux_t + Wh_{t-1} + b)$
- $y_t = softmax(Vh_t)$

Vanilla RNN is the simplest form of a recurrent neural network.



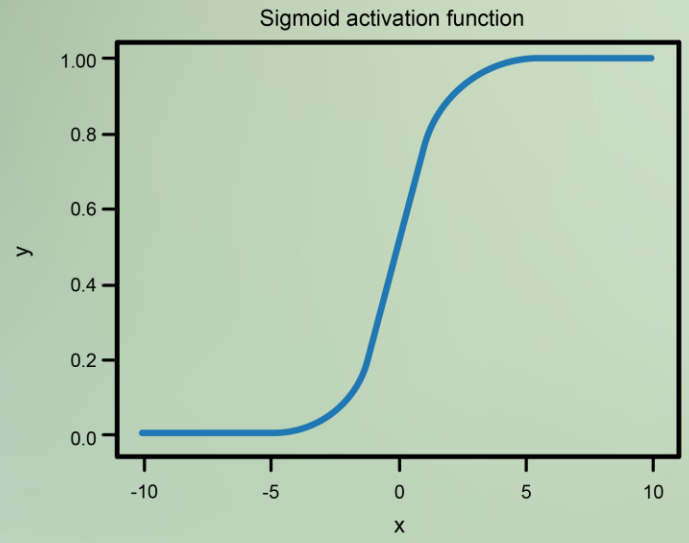
Recurrent Neural Networks (Activation Functions)

Sigmoid

- Often used for gates, and output layer for classification
- Pros: Non-linear, smooth gradient
- Cons: Not zero-centered, vanishing gradients

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Recurrent Neural Networks (Activation Functions)

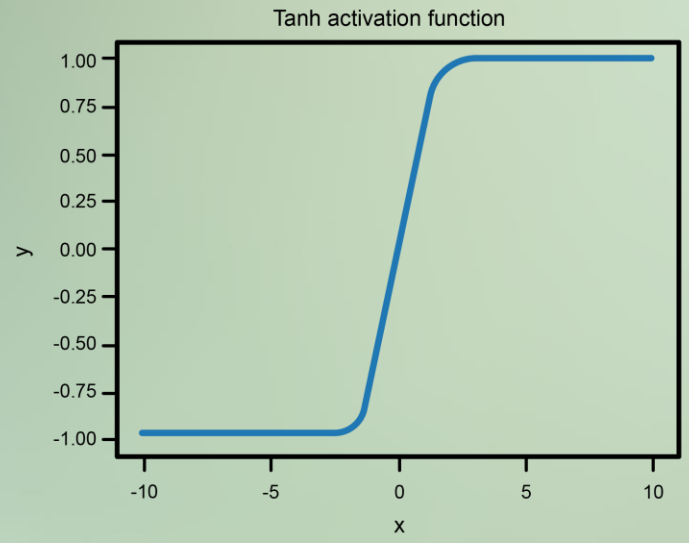
Tanh

- Used for hidden states & cells in RNNs, LSTMs.
- Pros: Zero-centred, often converges faster than sigmoid.
- Cons: Also suffer from vanishing gradients.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$tanh'(x) = 1 - \tanh^{2}(x)$$

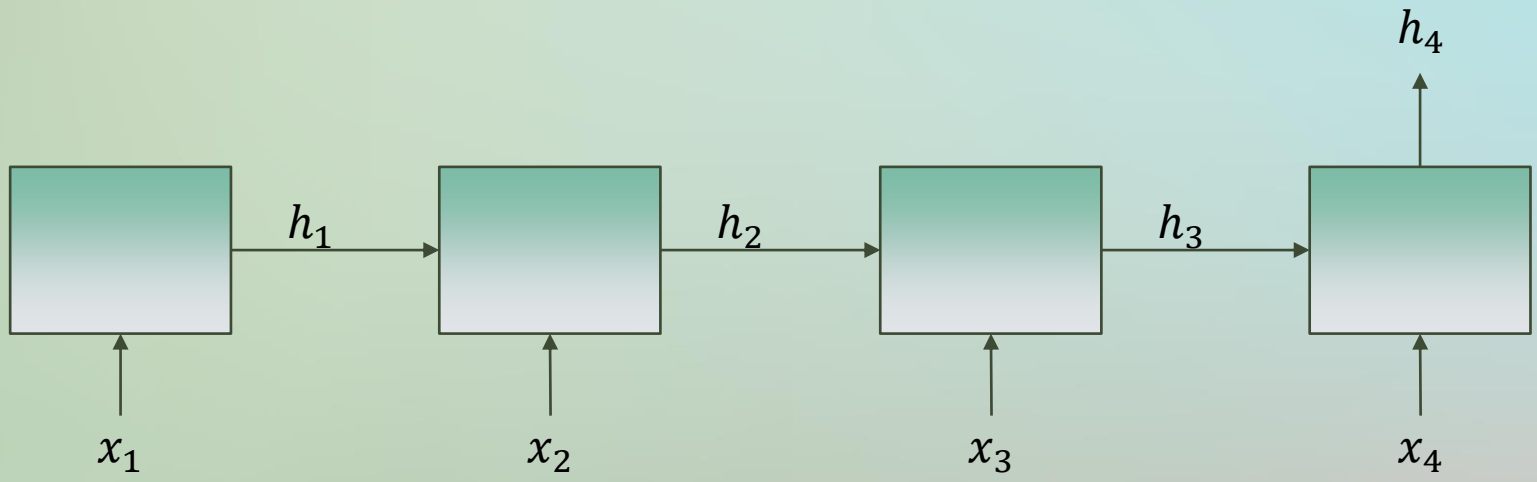
$$\tanh(x) = 2\sigma(2x) - 1$$



Recurrent Neural Networks – Classifier

- Follows a sequence-to-one architecture.
- Input: A sequence.
- Output: One Label (A classification).
- Use Case: Sentiment Analysis.

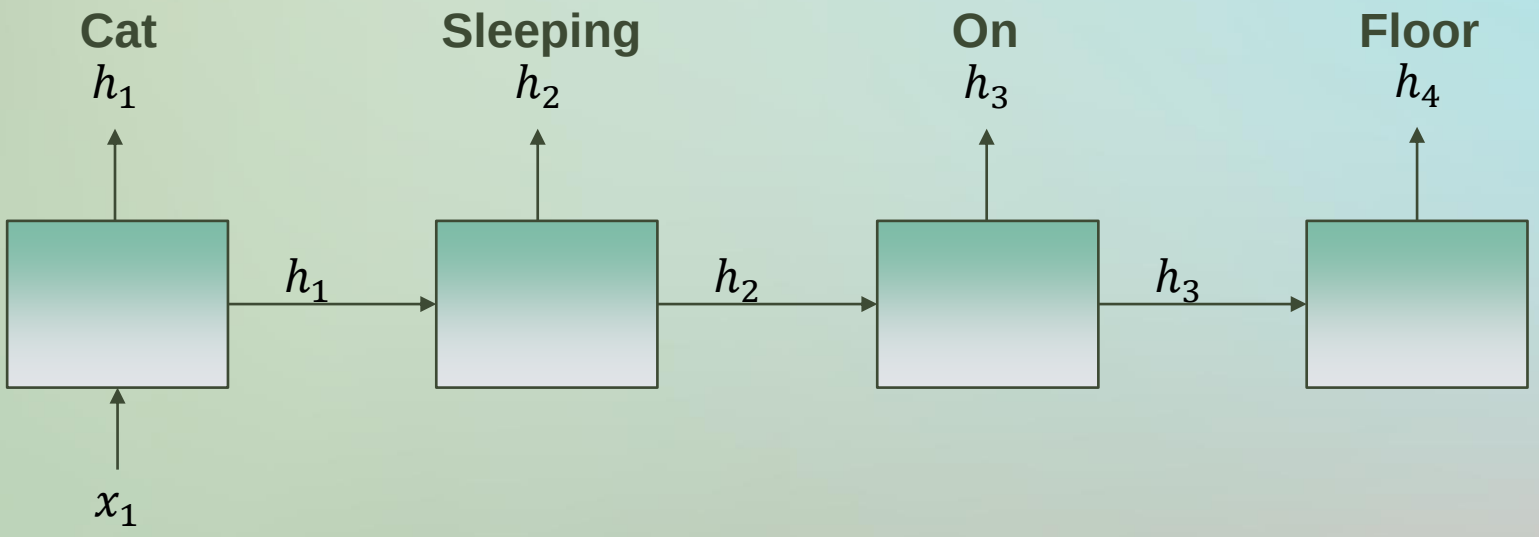
$$h_t = g(x_t, h_{t-1})$$
$$y = f(h_n)$$



Recurrent Neural Networks – One to Seq

- Follows a one-to-sequence architecture
- Input: One item
- Output: A sequence
- Use Case: Image Captions

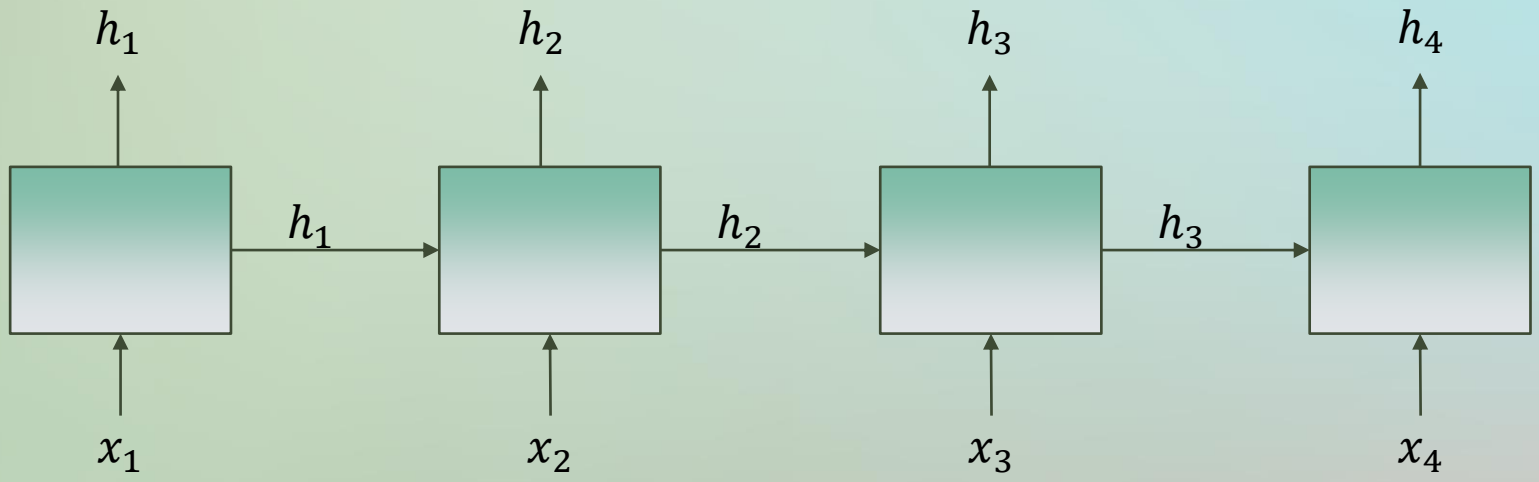
$$h_t = g(x_t, h_{t-1})$$
$$y_t = f(h_t)$$



Recurrent Neural Networks – Seq to Seq

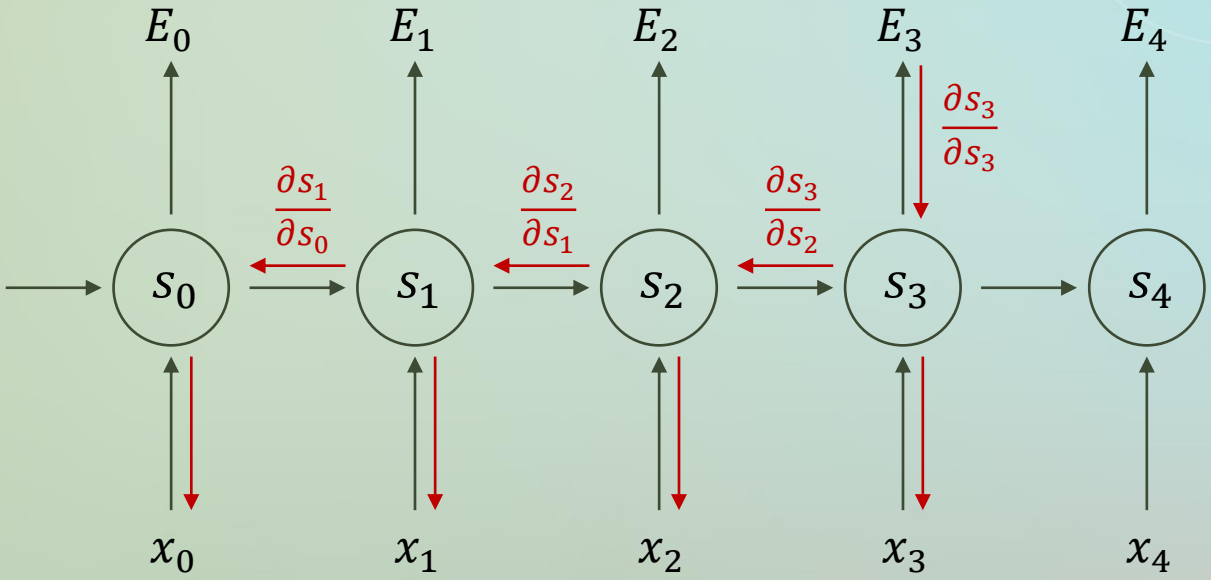
- Follows a sequence-to-sequence architecture
- Input: A sequence
- Output: A sequence
- Use Case: POS tagging, Named Entity Recognition

$$h_t = g(x_t, h_{t-1})$$
$$y_t = f(h_t)$$



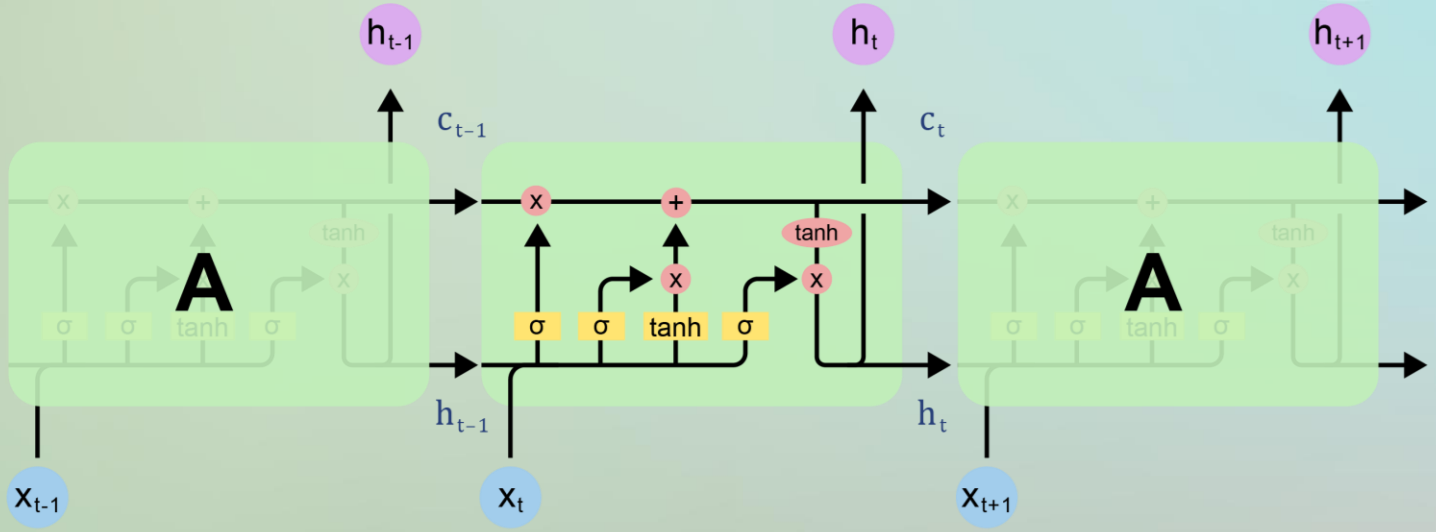
Recurrent Neural Networks – Limitations

- **Vanishing Gradients:** RNNs can struggle with long sequences because gradients can become too small, making it hard to learn from distant past information.
- **Exploding Gradients:** Conversely, gradients can become too large, causing unstable training.
- **Memory Limitations:** RNNs have limited short-term memory and may not remember relevant information from very early in a sequence due to Vanishing Gradients.

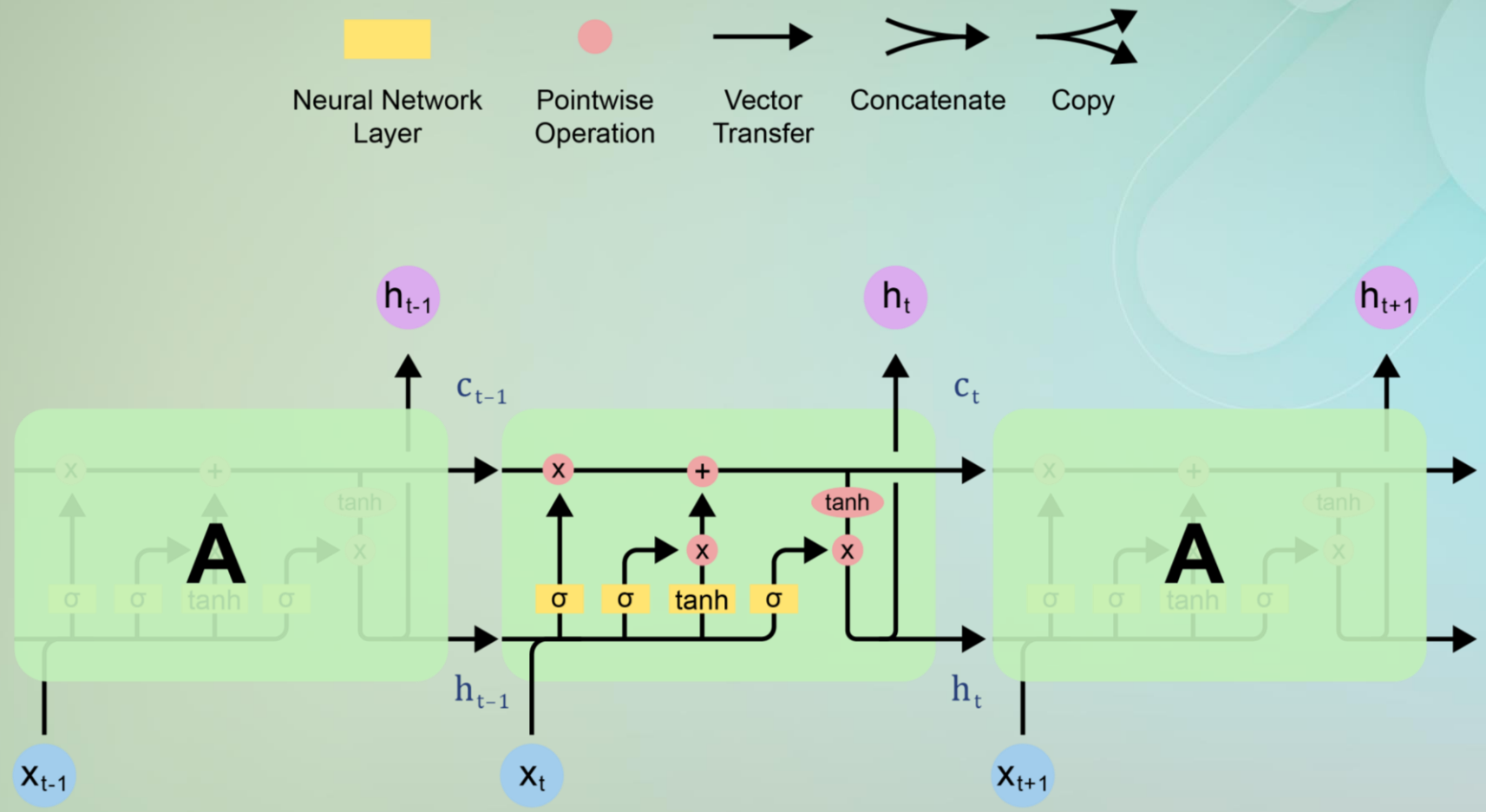


Long Short-Term Networks

- LSTM, short for Long Short-Term Memory, is a type of RNN architecture.
- Designed to address the vanishing gradient problem in RNNs.
- LSTMs incorporate unique gating mechanisms, including forget, input, and output gates, which allow them to regulate the flow of information and avoid the vanishing gradient problem common in standard RNNs.
- RNNs can struggle with long sequences, where information from early time steps may become difficult to capture. LSTMs overcome this limitation.

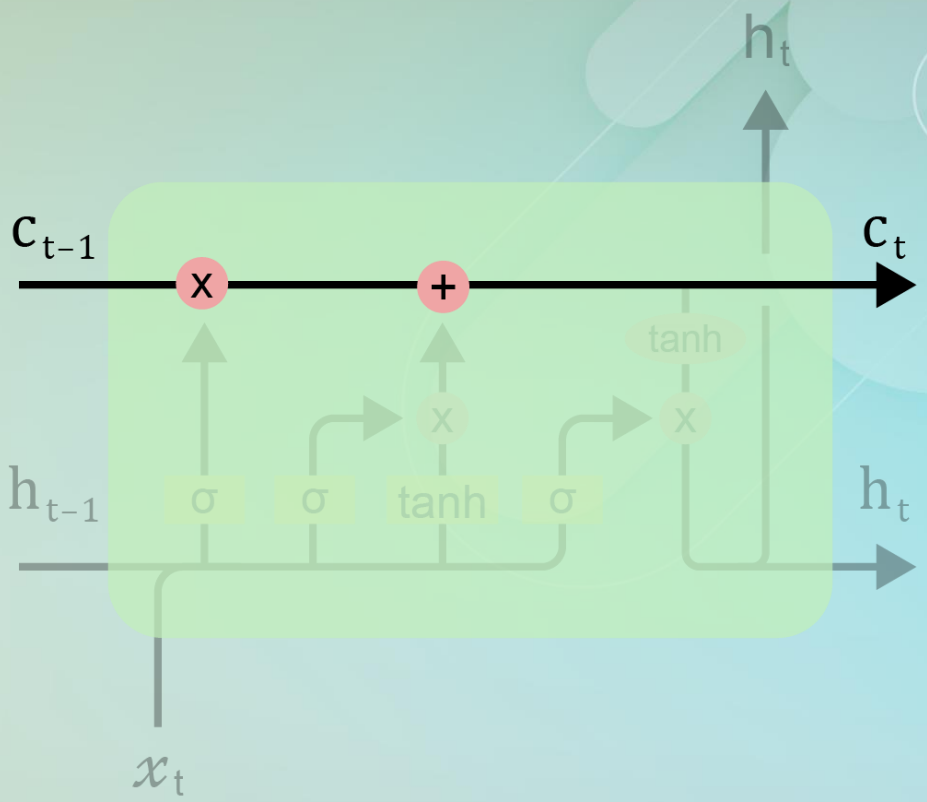


Long Short-Term Networks



LSTM – Cell States

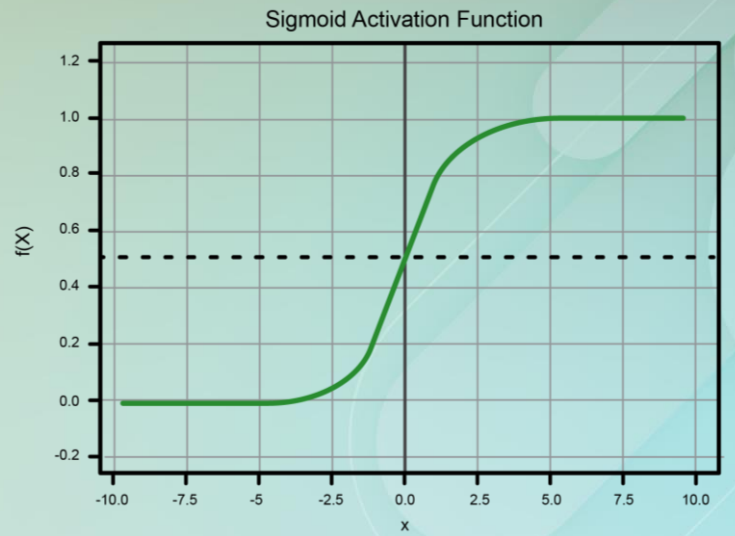
- Cell States represent long term memory.
- The cell state is modulated by three gates (forget, input, and output), which selectively add, retain, or remove information, thereby preventing the gradients from vanishing during long sequences.
- Cell states enable LSTMs to capture and retain long-term dependencies.

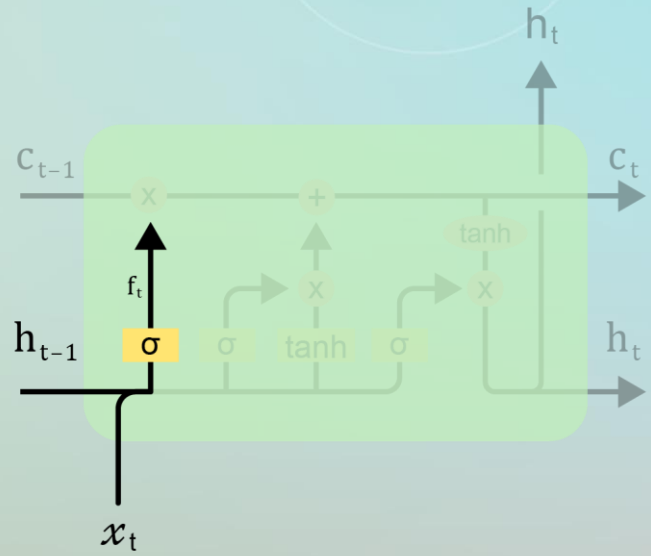


LSTM – Forget Gates

- Decides what long term information should be kept in the cell state.
- Information from previous hidden state and current input passed through sigmoid function.
- Output closer to 0 means forget, close to 1 means keep.
- Equation given by:

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





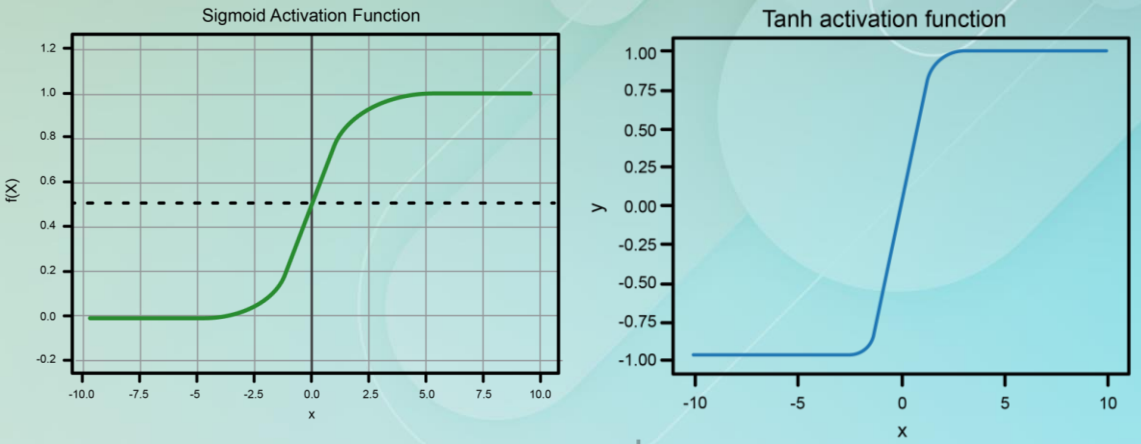
LSTM – Input Gates

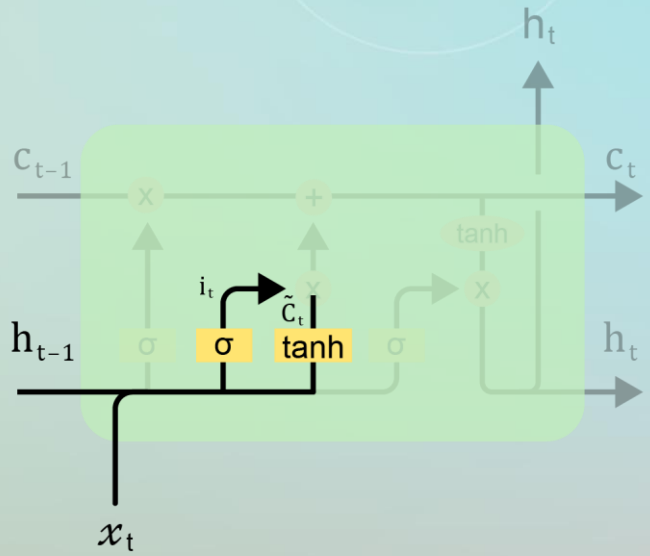
- Decides what new information to add to the cell state.
- First, a sigmoid layer decides which value to update and how much to update.
- Next, a tanh layer scales the input between 1 and -1.
- Combine them to create an update to the state of the long term memory.
- The input gate formula is given by:

$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$

- The equation for temporary new cell content is given by:

$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$

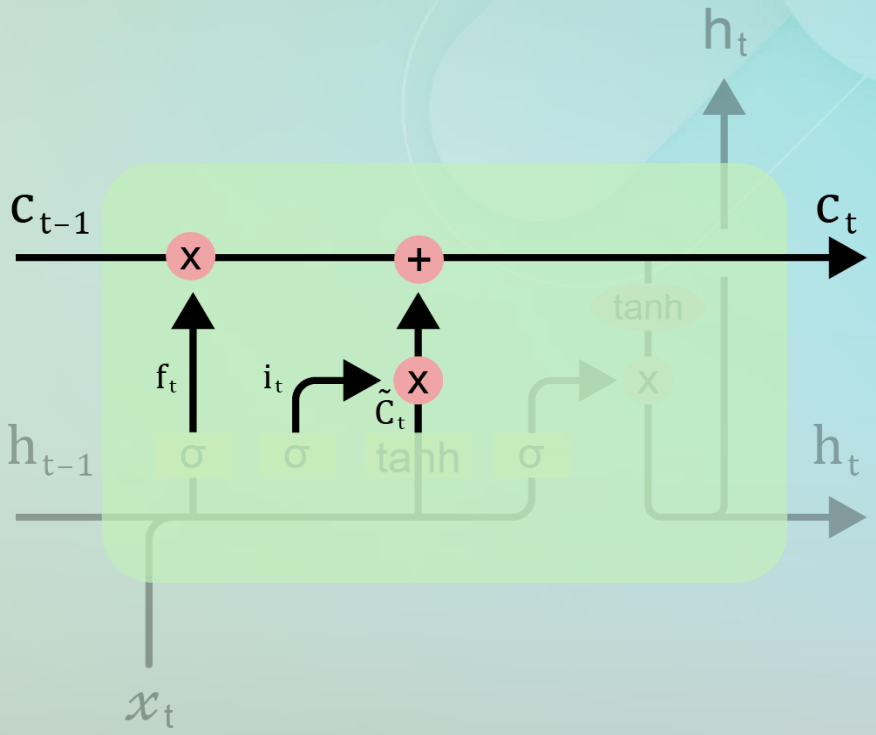




LSTM – Cell State Update

- The cell state is updated by a combination of the results from the forget gate and the input gate.
- The updated cell content is given by the equation:

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t$$



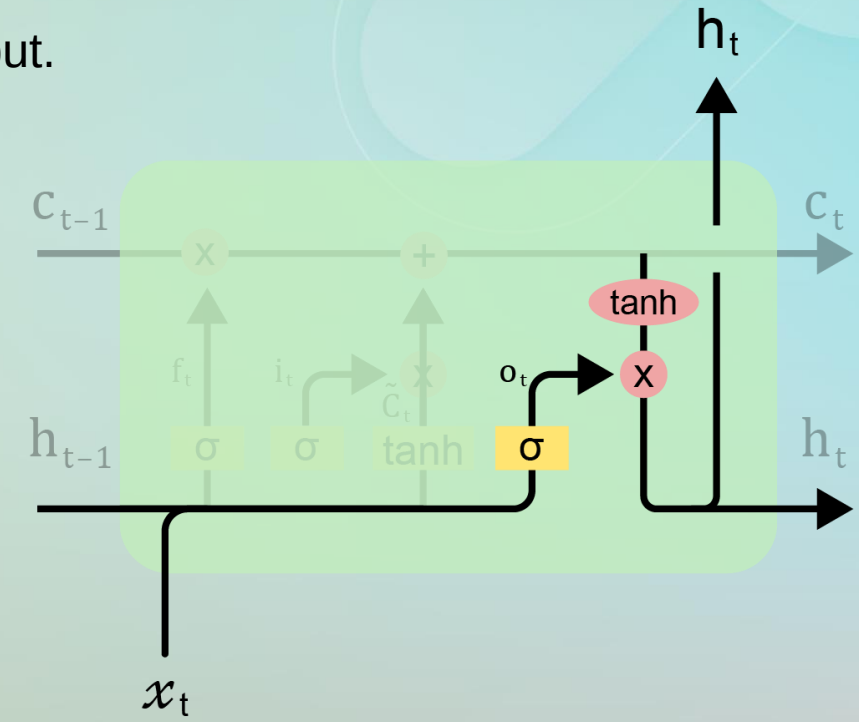
LSTM – Output Gates

- The output gate determines which parts of the cell state are used to generate the output at the current time step.
- The sigmoid layer decides how the current input and hidden state contribute to the output.
- The cell state passes through tanh function for normalisation.
- The two outputs are multiplied to produce the final output.
- The output gate equation is given by:

$$o_t = \sigma(U^{(o)}x_t + W^{(o)}h_{t-1} + b^{(o)})$$

- The hidden state is given by:

$$h_t = o_t \tanh(c_t)$$



GRUs

- Gated Recurrent Units (GRUs) were introduced in 2014 as a simplified variant of LSTM.
- They aim to achieve similar results with a reduced number of gates and parameters.
- GRUs do not have a separate cell state like LSTMs, and they merge the roles of the cell state and hidden state.
- As a result, they are computationally less expensive.
- A GRU has two fundamental components:

Update Gate:
Controls the extent to which the previous hidden state is updated.

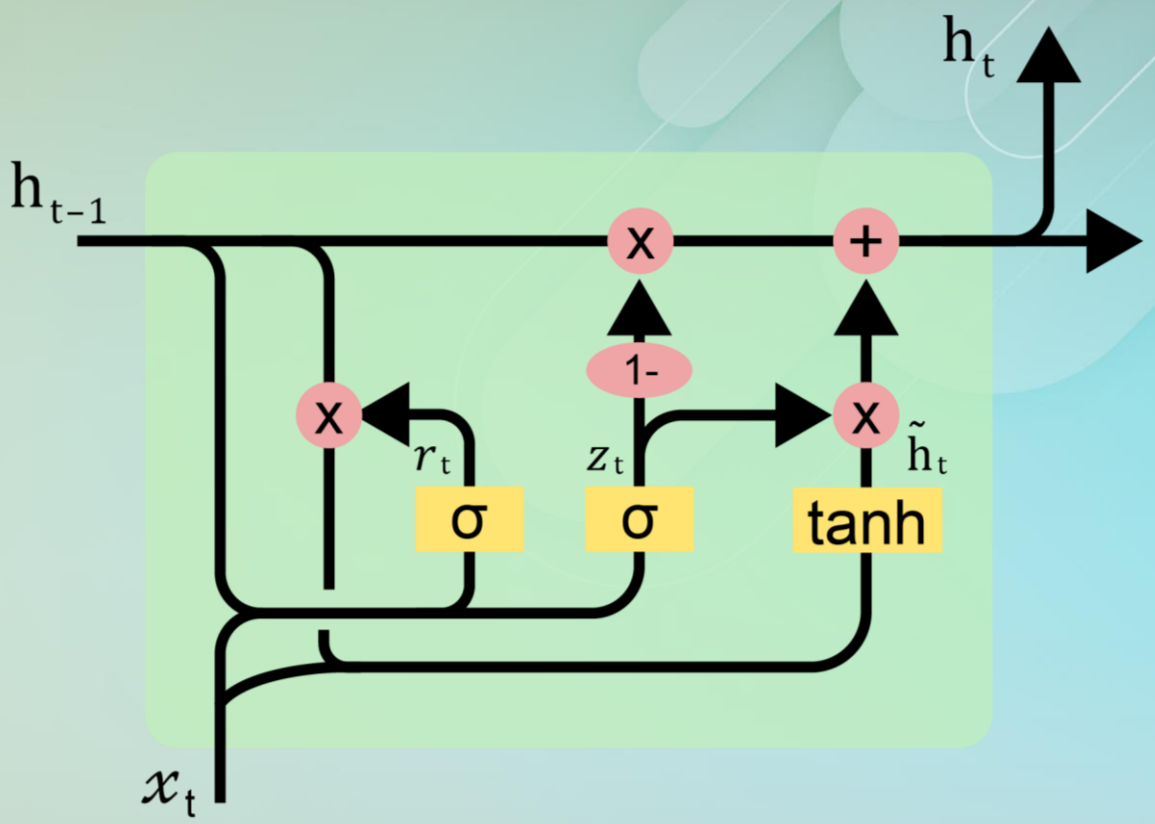
Reset Gate:
Controls the extent to which the previous hidden state is reset, allowing it to forget some information.

GRUs – Reset Gates

- The reset gate, denoted as r_t determines how much of the previous hidden state h_{t-1} should be forgotten.
- The information from the previous hidden state and the current input passes through a sigmod:

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

- The result scales the previous hidden state.
- The modified previous hidden state is then combined with current input and passes through a tanh to produce a candidate hidden state $\tilde{h}_t$.

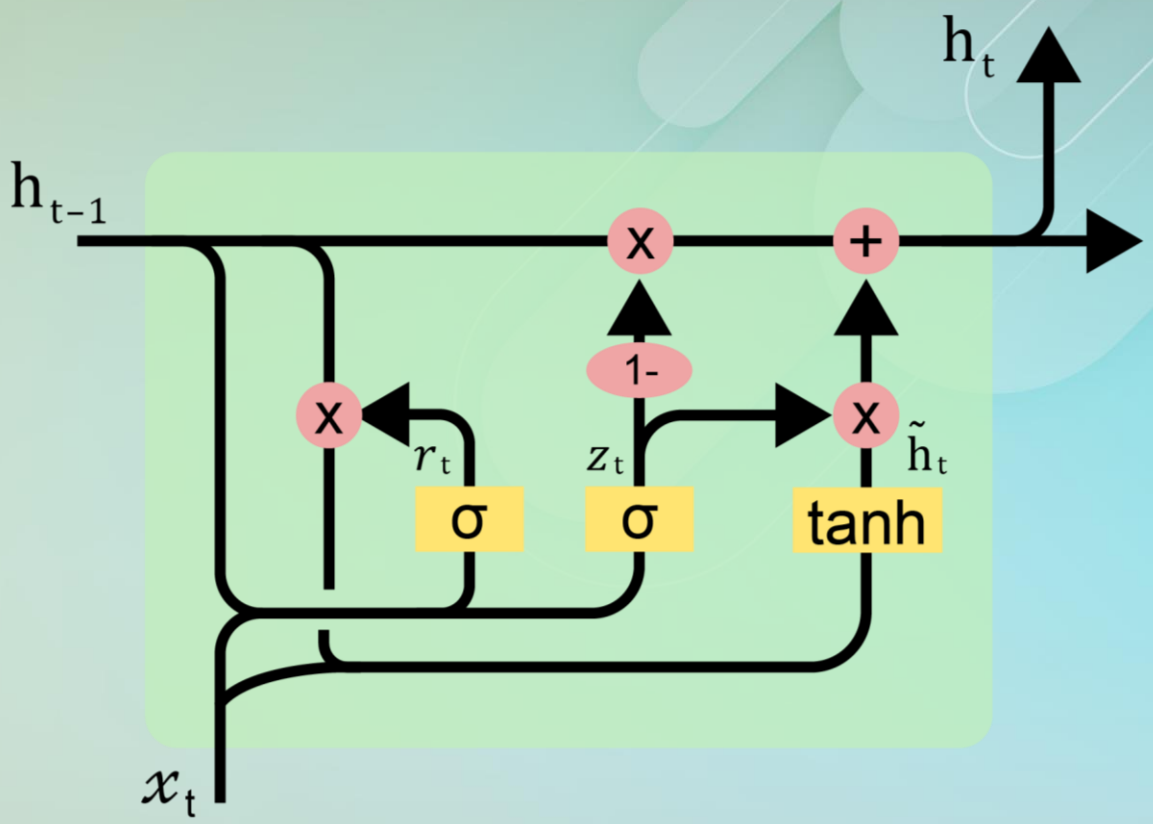


GRUs – Update Gates

- The update gate, denoted as z_t determines how much of the previous hidden state h_{t-1} should be retained and how much of the new candidate hidden state $\tilde{h}_t$ should be added to the current state.
- The formula for the update gate is given by:

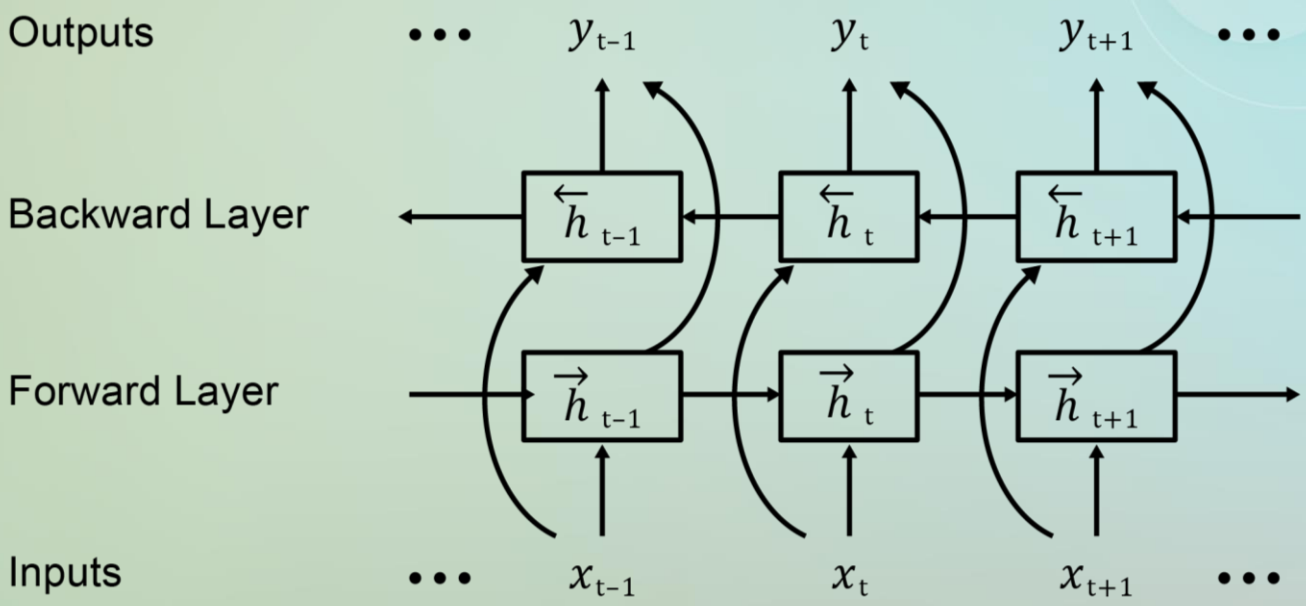
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

- The final new hidden state is scaled combination of the previous hidden state and the new candidate hidden state.



Bi-Directional RNNs

- Bi-Directional RNNs consist of two separate RNNs: one moving forward through the input sequence and the other moving backward.
- Bi-Directional RNNs are especially useful in applications where understanding the context from both past and future data points in a sequence is crucial.
- LSTM and GRU are commonly used in Bi-RNNs.



Summary

- **Sequential data:** Data that is chronologically ordered. Textual data falls under this category.
  - How traditional models fail to capture this sequential nature due to a lack of memory.

- Models that handle sequential data:

  - **RNNS:**
    A neural network for processing sequential data, capturing temporal information through its internal state.
  - **LSTMS:**
    An advanced RNN variant designed to learn long-term dependencies using special structures called gates.
  - **GRUs:**
    A streamlined version of LSTM with a simpler architecture, combining several gates for efficient learning of dependencies in sequences.
  - **Bi-RNNs:**
    An RNN that processes data in both forward and backward directions to capture context from the entire sequence.