Sequence modeling

Hossam Ahmed
Ziad Waleed
Mario Mamdouh

Sequence data

Sequence data is data where the **order** of elements is meaningful, and each element depends on previous (and sometimes future) elements in the sequence.

Video

Audio

Time-series

Text

DNA/RNA

GPS trajectory

Sequence data features

Order matters : The position of each element is essential; reordering changes the meaning

Context dependence : Each element is related to previous (and sometimes future) elements

Variable length : Sequences can be of different lengths — not all inputs have the same size.

Sequence modeling challenges

Sequence modeling requires handling variable-length inputs, preserving temporal dependencies and order, aligning inputs and outputs over time, and maintaining long-term memory — all of which are difficult for traditional neural networks.

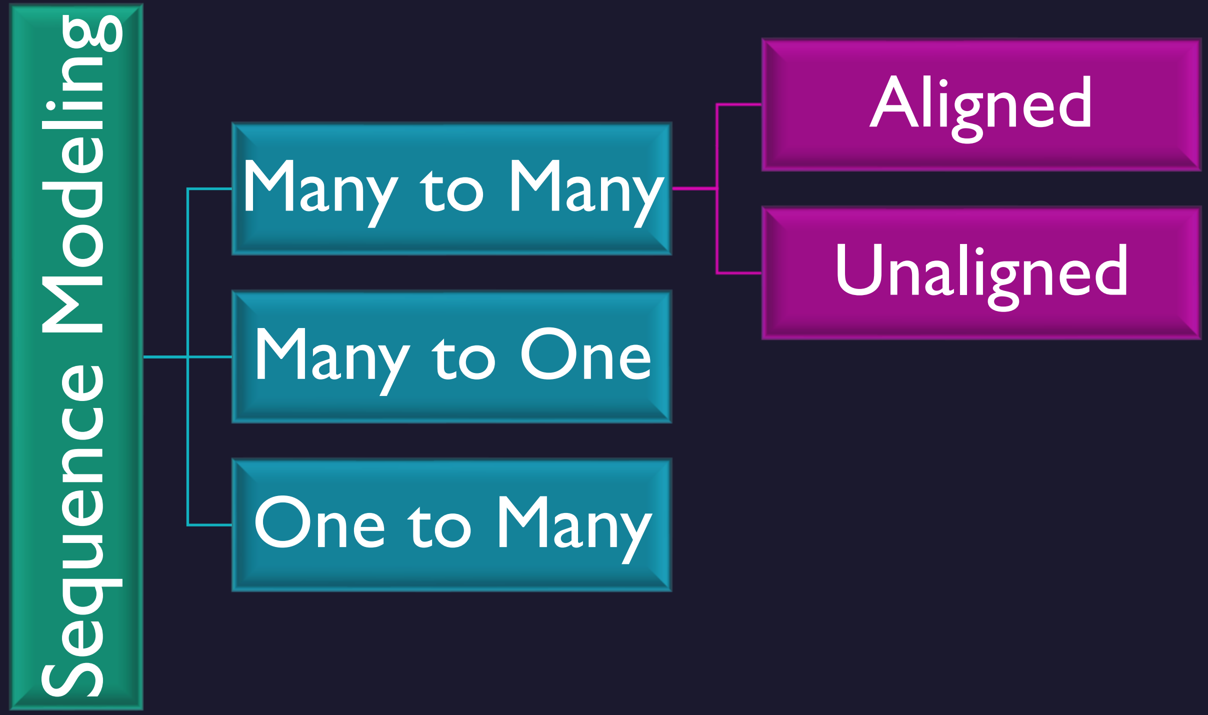
Model Type	Limitation
ANNs	• Fixed input size: can't handle variable-length sequences. • No memory: each input is treated independently. • No temporal awareness.
CNNs	• Local receptive fields only capture short-range patterns. • Work with grid data and don't expect a sequence. • Position is learned indirectly (via filters). • Can't model long-term dependencies well.

Sequence modeling types

Many to One: Sequence is reduced to a single output — e.g., classifying a full sentence or time-series

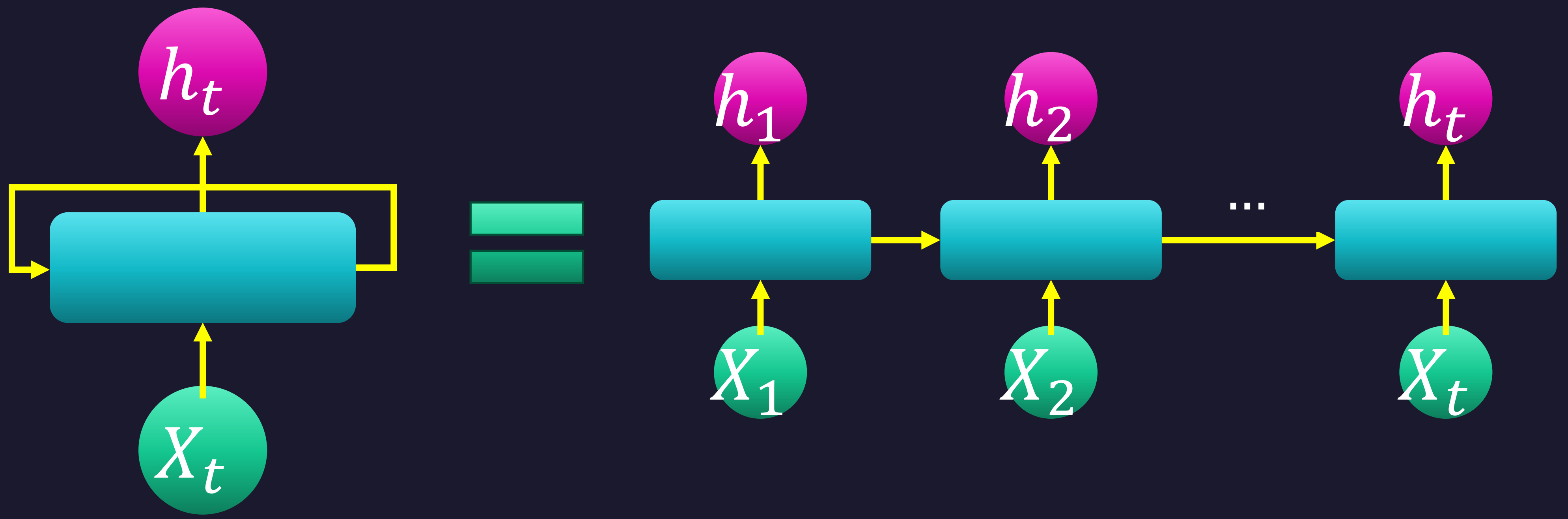
One to Many: Fixed input produces a sequence — e.g., image → caption generation

Many-to-many: maps input to output sequences, either **aligned** (same length, e.g., tagging) or **unaligned** (different length, e.g., translation).



Recurrent Neural Networks (RNN)

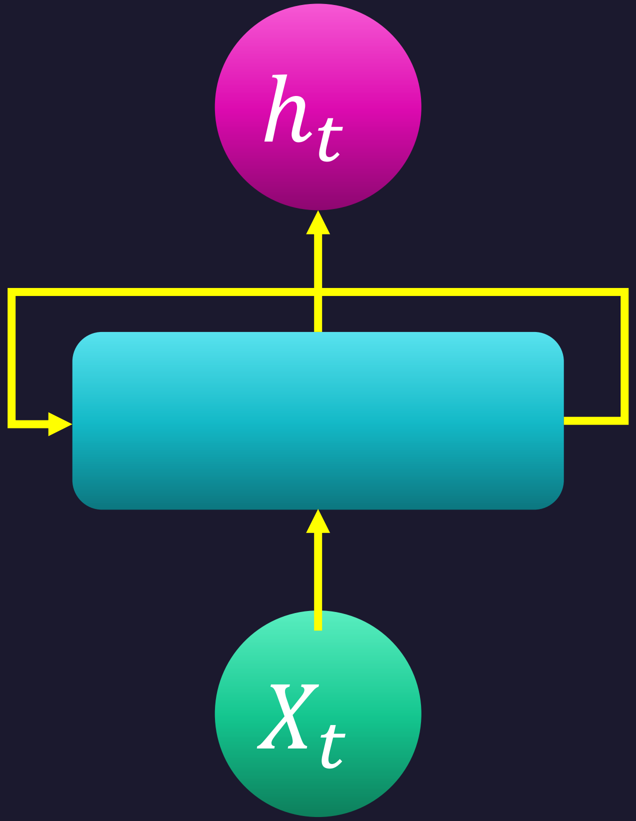
Recurrent Neural Networks (RNNs) are neural architectures designed for sequential data, where the model maintains a **hidden state** that is updated at each time step to capture **context and dependencies** across the sequence.



Inference using RNNs

At each time step ⏳ t, the RNN :

1. Takes the **current input X_t** and the **previous hidden state h_{t-1}**
2. Combine them using learned weight matrices (W, U)
3. Applies a non-linear activation function to ***update*** the hidden state h_t
4. Uses the updated hidden state to produce the output y_t



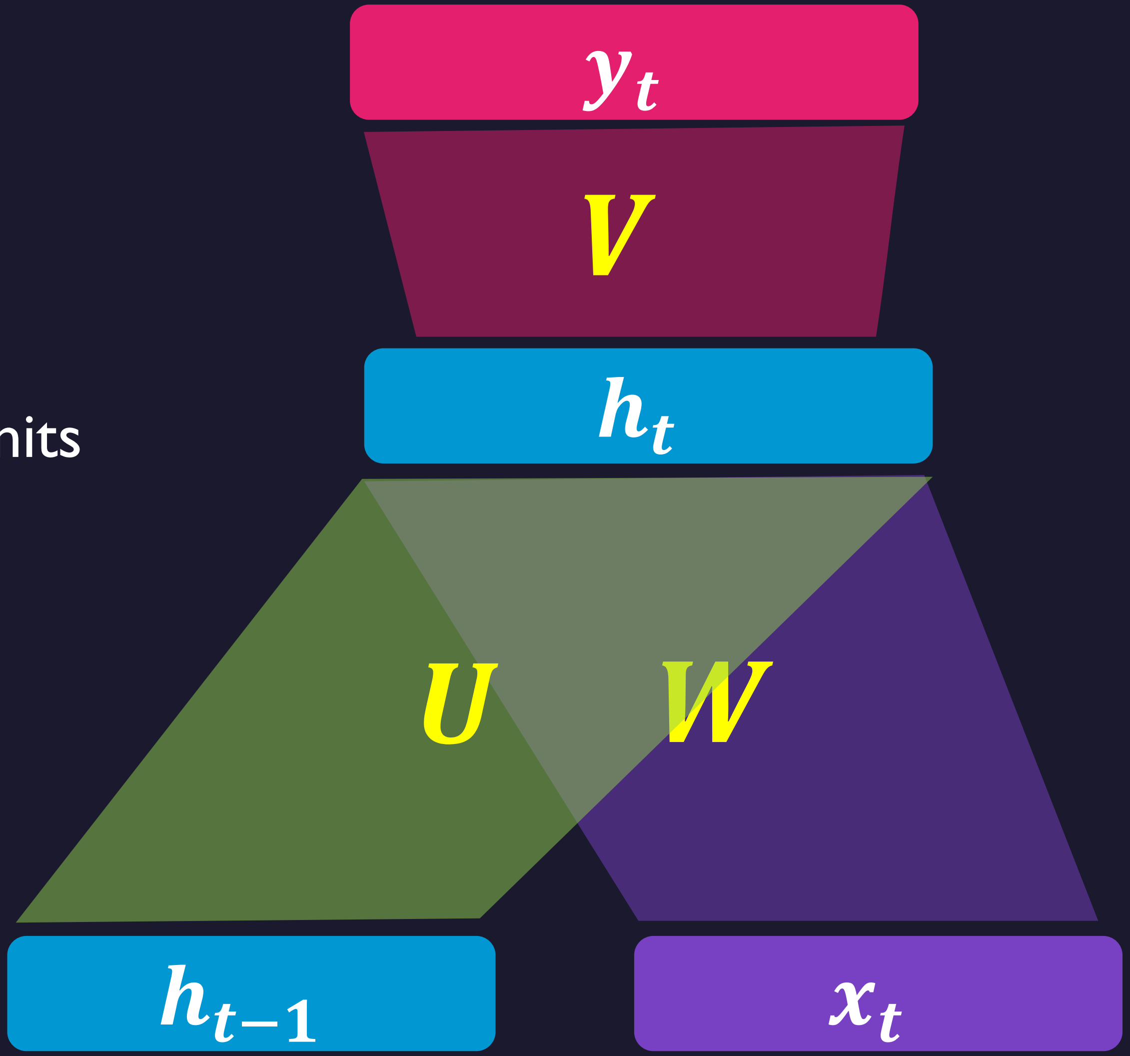
- The hidden state acts like **memory**, carrying information from previous time steps forward.

$$h_t = g(Uh_{t-1} + WX_t)$$

$$y_t = f(Vh_t)$$

Inference using RNNs

- **Input vector X_t** $\in \mathbb{R}^{n_x}$, is the number of features per time step
  - Then $W \in \mathbb{R}^{n_h \times n_x}$
- **Hidden state h_t** $\in \mathbb{R}^{n_h}$, number of hidden units (memory capacity) like the neurons in ANNs.
  - Then $U \in \mathbb{R}^{n_h \times n_h}$
- **Output vector y_t** $\in \mathbb{R}^{n_y}$, task dependent
  - Then $V \in \mathbb{R}^{n_h \times n_y}$

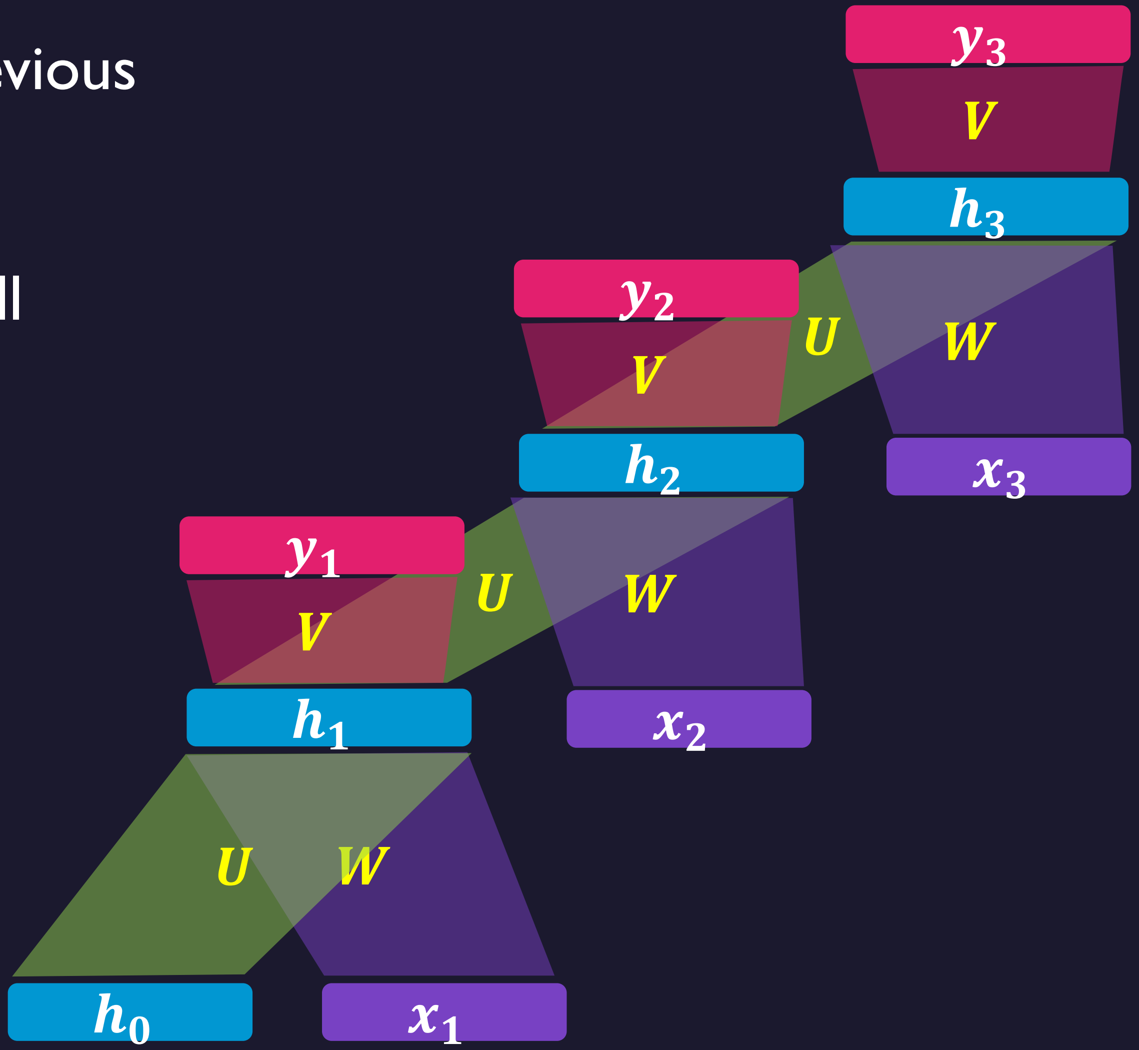


Inference using RNNs

- h_0 is initialized to be zeros as there is no previous hidden states.
- Weight matrices $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}$ are shared across all time steps
- This is many to many RNN
- https://joshvarty.github.io/VisualizingRNNs/

$$\boldsymbol{h_t = g(Uh_{t-1} + WX_t)}$$

$$\boldsymbol{y_t = f(Vh_t)}$$



Limitations of RNNs

Vanishing/Exploding gradients: Gradients shrink or grow exponentially during backpropagation through time, making training unstable or slow.

Short-term memory: Struggles to capture long-range dependencies — important information fades over time.

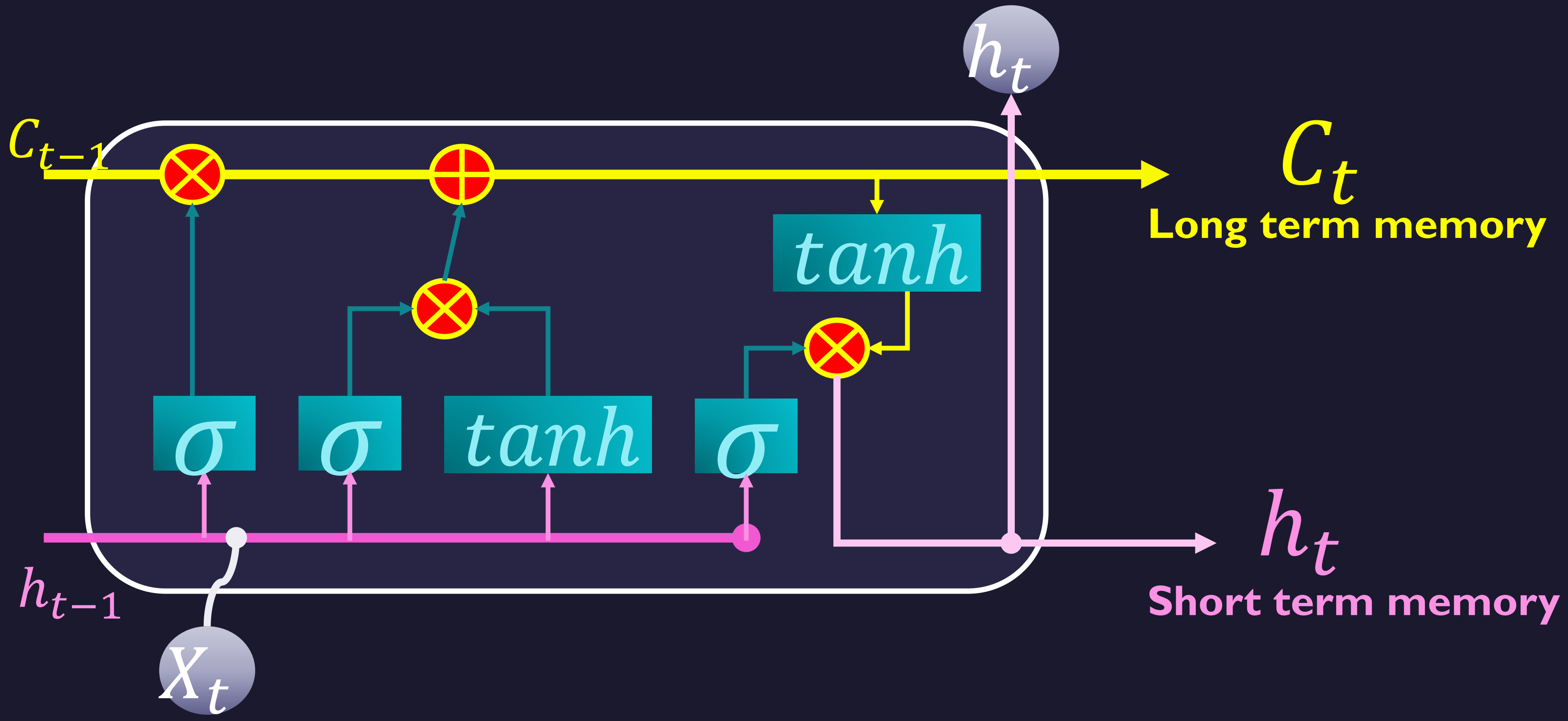
Sequential computation: Cannot parallelize across time steps — slows training and inference.

Inventions to alleviate RNNs limitations

Invention	Solves	How It Helps
LSTM (Long Short-Term Memory)	Vanishing gradients, long-term memory	Adds **gates** (forget, input, output) and a **cell state** to maintain long-term dependencies
GRU (Gated Recurrent Unit)	Like LSTM, but with fewer parameters	Combines forget & input gates into an **update gate** — simpler, faster training
Bidirectional RNNs	Limited context, one-directional dependency	Processes sequences **forward and backward**, improving context awareness
Attention Mechanisms	Long-range dependency, fixed memory bottleneck	Allows the model to **focus selectively** on relevant parts of the sequence
Transformer Architecture	Sequential computation, long-term memory	Replaces recurrence with **self-attention**, enabling **parallelization** and better long-range modeling

Long Short-Term memory (LSTM)

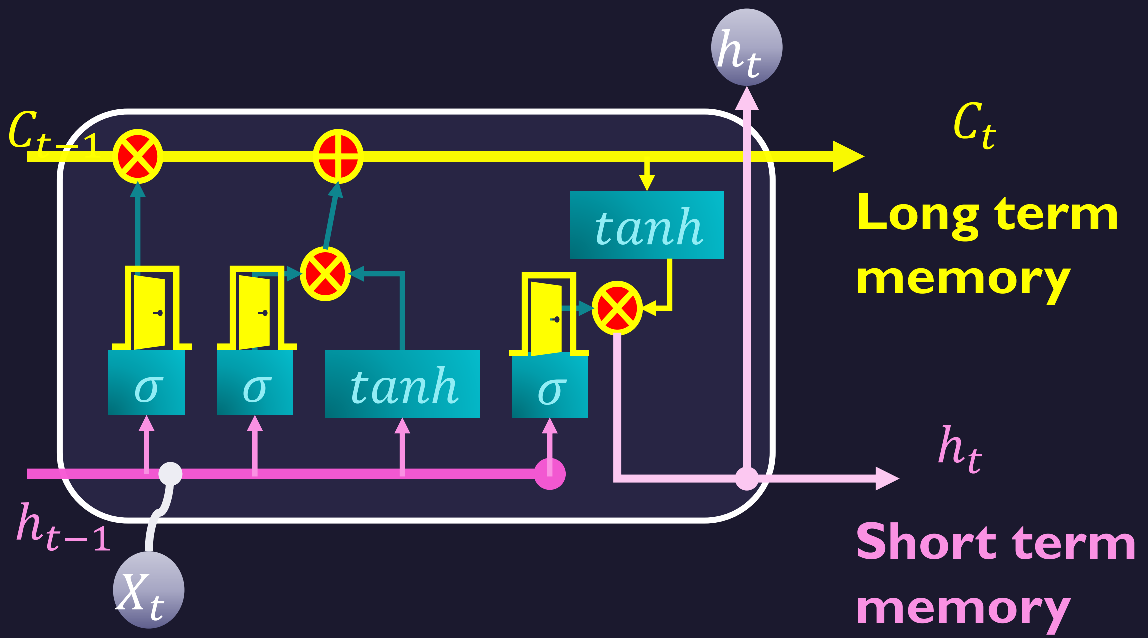
- LSTM was introduced to solve the **vanishing gradients** problem to be able to train deeper RNN and to alleviate the loss of old information in the sequence.



Long Short-Term memory (LSTM)

An **LSTM unit** consists of a **cell state C_t**, a **hidden state h_t**, and **three gates** (forget, input, output) that **control the flow of information** to preserve important signals across long sequences.

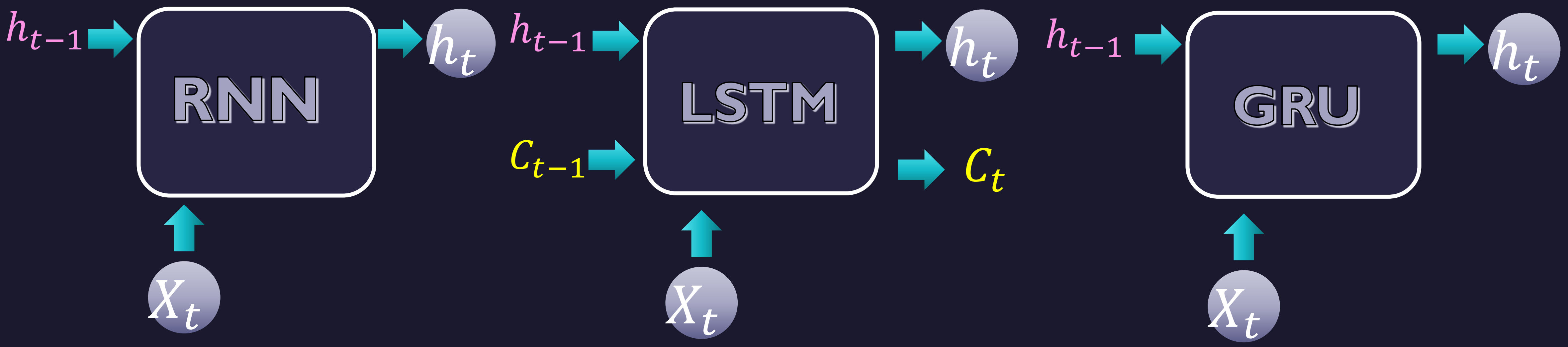
- Forget gate f_t discards irrelevant past memory
- Input gate i_t decides what new info to store
- Cell state C_t combining information from old and new memory to update the long memory
- Output gate o_t use the updated memory to calculate the next hidden state



Long Short-Term memory (LSTM)

Gate	Function	Equation
Forget Gate	Discards irrelevant past memory	$f_t = \sigma\,(U_f h_{t-1} + W_f X_t + b_f)$ $K_t = C_{t-1} \odot f_t$
Input Gate	Decides what new info to store	$i_t = \sigma\,(U_i h_{t-1} + W_i X_t + b_i)$ $g_t = tanh(U_g h_{t-1} + W_g X_t)$ $J_t = i_t \odot g_t$
Cell Update	Combines old and new memory	$C_t = C_{t-1} \odot f_t + i_t \odot g_t$ $C_t = K_t \odot J_t$
Output Gate	Controls what memory is revealed	$o_t = \sigma\,(U_o h_{t-1} + W_o X_t + b_o)$ $h_t = o_t \odot tanh(c_t)$

RNN *vs* LSTM *vs* GRU



RNN has no gates and the fewest parameters, making it fast but weak at remembering. **LSTM** uses **3 gates** (input, forget, output) and a **cell state**, resulting in **4× the parameters** of a simple RNN, **GRU** uses **2 gates** (update and reset), merges the cell and hidden state, and needs **3× the parameters** of an RNN.

Number of parameters

- To calculate the number of parameters for a recurrent neural network we use this formula
  - $W * |GATES| + \boldsymbol{U} * |GATES| + b * |GATES|$
  - $W \in \mathbb{R}^{n_h \times n_x}, U \in \mathbb{R}^{n_h \times n_h}$
  - Assuming it's Many-to-One (e.g. Classification) so we don't bother with V
- If we have 64 recurrent unit, and the input vector resulting from the embedding layer 45
  - For Simple RNN $(45 * 64) * |1| + (\mathbf{64} * \mathbf{64}) * |1| + 64 * |1| = \mathbf{7040}$
  - For LSTM $(45 * 64) * |4| + (\mathbf{64} * \mathbf{64}) * |4| + 64 * |4| = \mathbf{28160}$
  - For GRU $(45 * 64) * |3| + (\mathbf{64} * \mathbf{64}) * |3| + 64 * |3| = \mathbf{21120}$
  - In TensorFlow GRUs has two biases for each recurrent unit so
    - $(45 * 64) * |3| + (\mathbf{64} * \mathbf{64}) * |3| + 64 * 2 * |3| = \mathbf{21312}$

References

- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
- https://d2l.ai/chapter_recurrent-modern/bi-rnn.html
- https://youtu.be/AsNTP8Kwu80?si=dDL6yuahw1zocxIC
- https://youtu.be/YCzL96nL7j0?si=ZerUz-cTqG-EwMvb

See 👀

- https://joshvarty.github.io/VisualizingRNNs/ ✨
- https://distill.pub/2019/memorization-in-rnns/
- https://damien0x0023.github.io/rnnExplainer/ ✨