Topic 7b

Long Short-Term Memory (LSTM)

CSE465: Pattern Recognition and Neural Network
Sec: 3
Faculty: Silvia Ahmed (SvA)
Spring 2025

Topics

1. Evolution from RNN to LSTM
2. LSTM Architecture and Working Mechanism
    1. Components of LSTM
    2. Mathematical Formulation
    3. Implementation in PyTorch
3. Applications of LSTM
4. Limitations of LSTM
    - GRUs as simplified alternative
    - Introduction to Transformer Networks

Issues with RNN

- Recurrent Neural Networks (RNNs) were introduced to handle sequential data, like time series, speech, and text. However, they had **two major problems**:
  1. **Short-Term Memory Issue:** RNNs struggled to retain information from earlier time steps over long sequences. As a result, important past information was often lost.
  2. **Vanishing & Exploding Gradients:** When training deep RNNs, gradients (used for learning) either became too small (vanishing) or too large (exploding). This made learning difficult, especially for long-term dependencies.

Why RNNs Struggle with Long-Term Dependencies

"In the early morning, the baker opened his shop. The smell of fresh bread filled the air. Customers lined up, eager to buy croissants, baguettes, and muffins. As the day went on, the shop remained busy, with people coming in for sandwiches and coffee. By the evening, the baker was tired but happy, knowing he had sold all his _____."

- Why This is Difficult for an RNN
  - To correctly fill in the blank, the model must **remember what the baker was selling** at the beginning: **bread, croissants, baguettes, muffins**.
  - However, since an RNN processes the sentence step by step, **earlier words lose influence** due to the **vanishing gradient problem**.
  - By the time the model reaches the blank, it may have forgotten **what was being sold** and might predict something generic like **“items”** instead of **“bread”** or **“pastries”**.

Why This Happens: The Vanishing Gradient Problem

- At each step, an RNN updates its hidden state using a function like:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$

- When backpropagation computes gradients for early words, they become **smaller and smaller** due to repeated multiplications of small weight values.
- This means the **first word's influence fades away** by the time the model reaches the end of the sentence.

Solution: LSTM

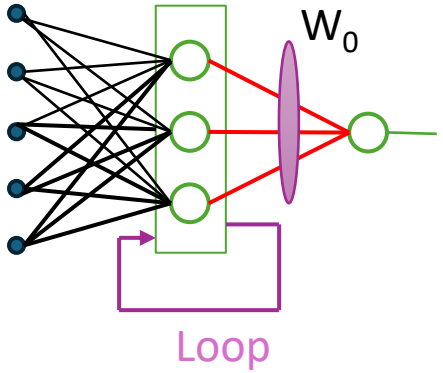
- LSTM: Long Short-Term Memory
- Introduced by Hochreiter and Schmidhuber in 1997
- Designed to **fix RNN's memory limitations**
- It introduced a new memory structure called the **Cell State**, which allowed information to flow unchanged for long periods.
- LSTM uses **gates** to control what information should be remembered or forgotten:
  - **Forget Gate:** Removes unnecessary past data.
  - **Input Gate:** Selects important new data to store.
  - **Output Gate:** Decides what part of memory to use for the current task.

Analogy

- Think of LSTM like a **skilled note-taker** in a classroom. The student listens to a lecture and writes down important points while ignoring unnecessary details.
- They have three strategies:
  1. **Forget old, irrelevant info:** If a topic is no longer useful, they erase those notes (Forget Gate).
  2. **Remember key details:** They carefully decide what new information is important and should be written down (Input Gate).
  3. **Use past knowledge effectively:** When answering a question, they look at both past notes and recent information to make the best response (Output Gate).

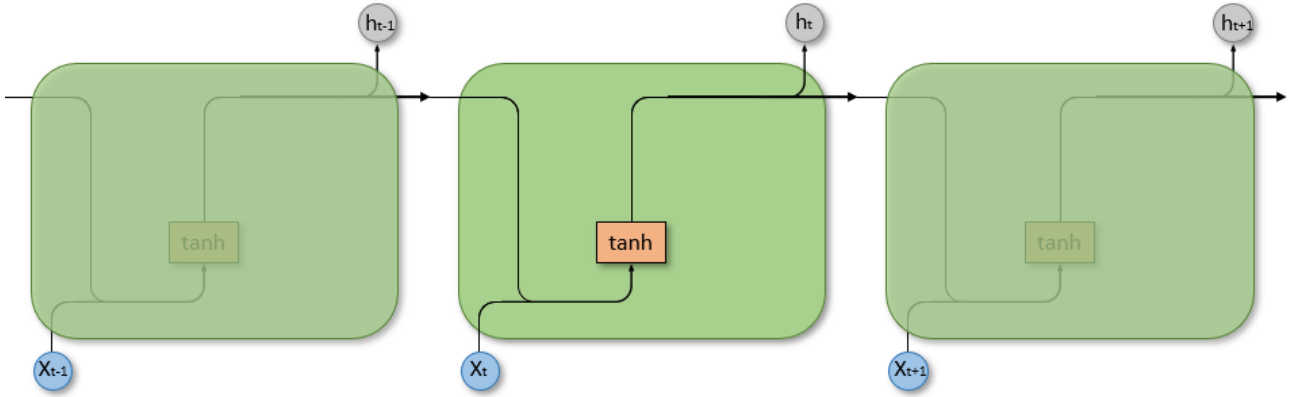
RNN Forward Propagation (Recap)

Review	Sentiment
X_{11}, X_{12}, X_{13}	1
X_{21}, X_{22}, X_{23}	0
$X_{31}, X_{32}, X_{33}, X_{34}$	0

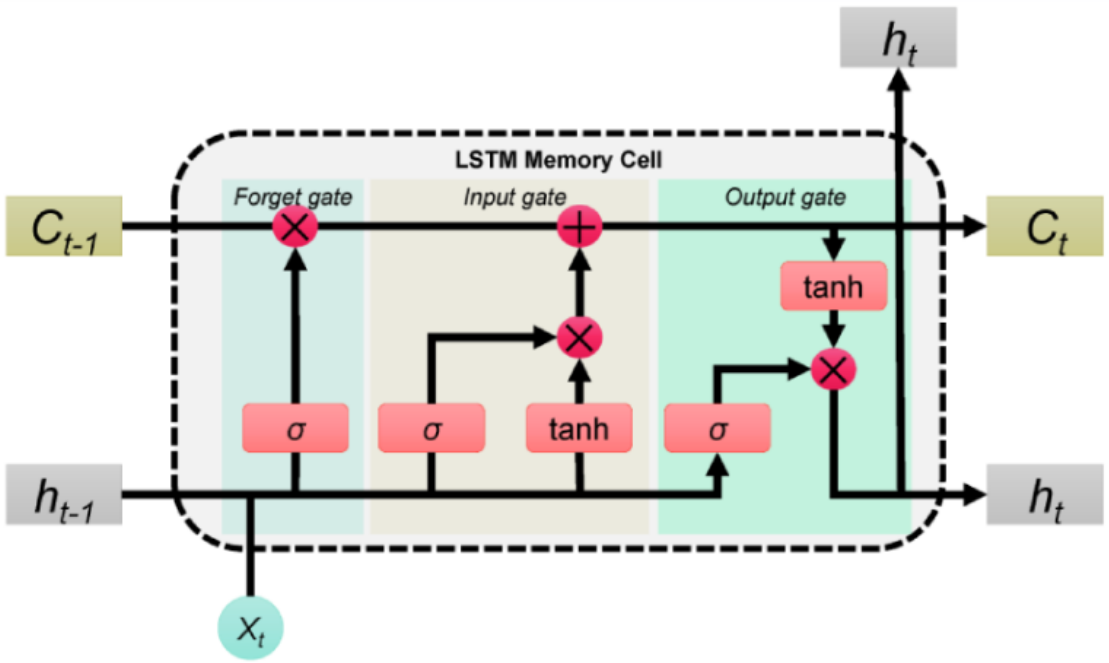


Forward → unfolding/unrolling through time

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$



Mathematical Structure & Architecture of LSTM



Each LSTM unit consists of:

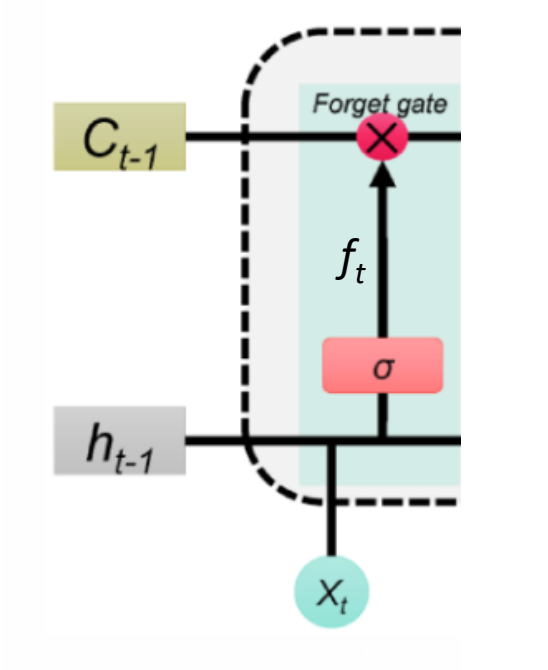
1. Forget Gate (f_t)
2. Input Gate (i_t)
3. Cell State Update (c_t)
4. Output Gate (o_t)

Final Outputs:

At each time step, the LSTM outputs:

- Hidden State (h_t): used for further processing (e.g., in a final dense layer).
- Cell State (c_t): Maintains long-term memory.

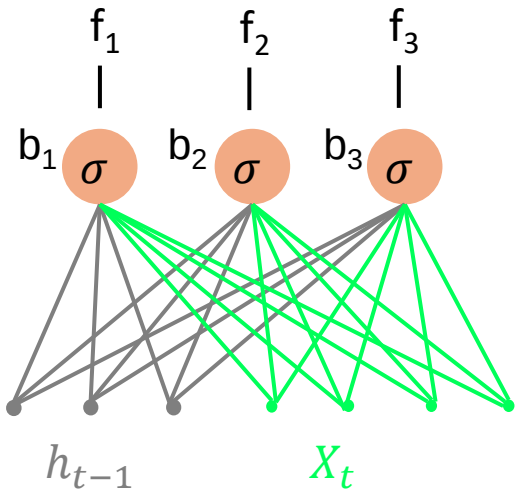
1. Forget Gate



e.g. $[X_{i1}\ X_{i2}\ X_{i3}\ X_{i4}]$

(4-dim vector)

Eg. Number of units: 3



$[h_{t-1}, X_t]$

(7x1)

W_f

$(3x7)$

Step 1:

$$f_t = \sigma\left(W_f[h_{t-1}, X_t] + b_f\right)$$

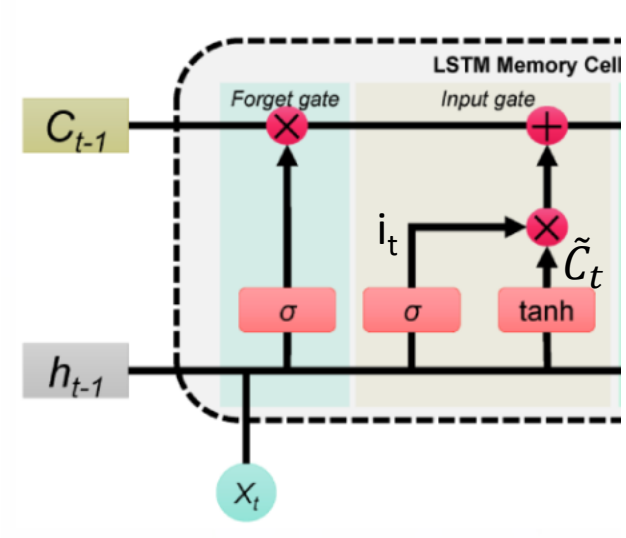
Step 2: ⊗ → Elementwise multiplication

$f_t \otimes C_{t-1}$ → f_t controls how much to remove from C_{t-1}

C_{t-1} = [4 5 6] f_t = [0.5 0.5 0.5] ⟹ $f_t \otimes C_{t-1}$ = [2 2.5 3] ⟹ 50% removal

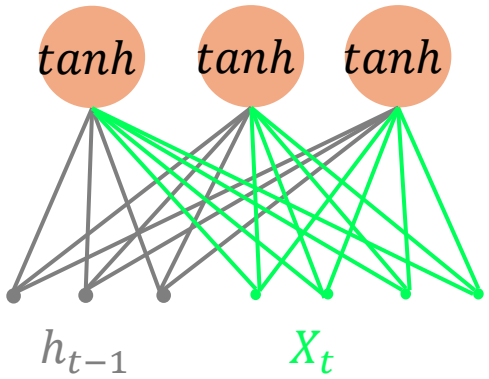
f_t = [0 0 0] ⟹ $f_t \otimes C_{t-1}$ = [0 0 0] ⟹ 100% removal

2. Input Gate

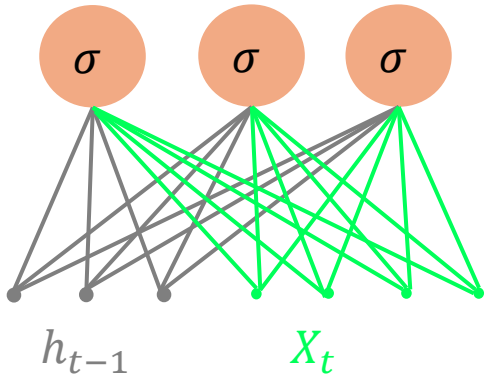


Stages:

1. Calculate candidate cell state, $\tilde{C}_t$
2. Calculate i_t
3. Calculate current cell state, C_t



$$\tilde{C}_t = \tanh(W_c[h_{t-1}, X_t] + b_c)$$



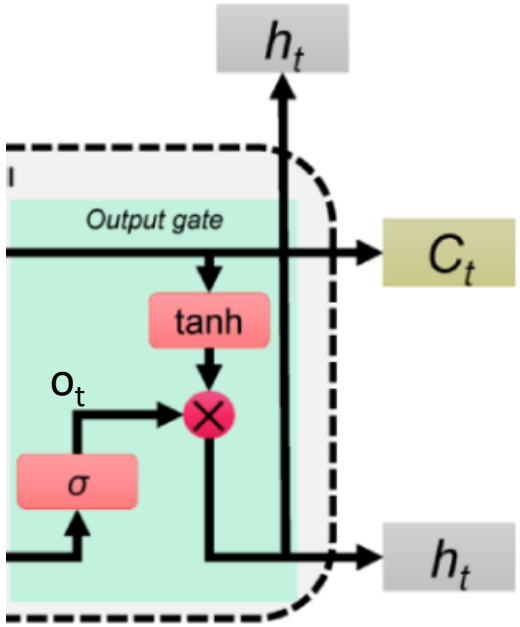
$$i_t = \sigma(W_i[h_{t-1}, X_t] + b_i)$$

$i_t \otimes \tilde{C}_t$ → Filtered candidate cell state

Cell State Update:

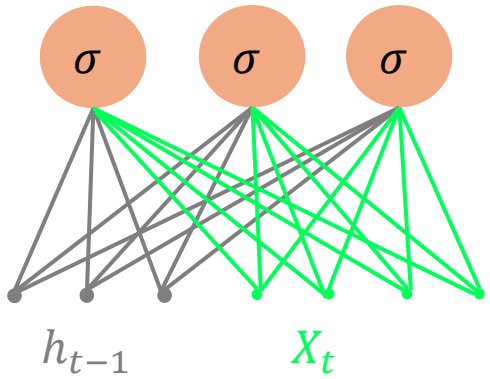
$$C_t = f_t \otimes C_{t-1} \oplus i_t \otimes \tilde{C}_t$$

3. Output Gate



Stages:

1. Calculate o_t
2. Calculate $\tanh(C_t)$
3. Calculate current hidden state, h_t



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

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

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

Applications of LSTM

- Natural Language Processing:
  - Language Modeling
  - Machine Translation
  - Chatbots
- Time Series Forecasting:
  - Weather prediction
  - Stock market trends
- Speech Recognition
- Music Composition

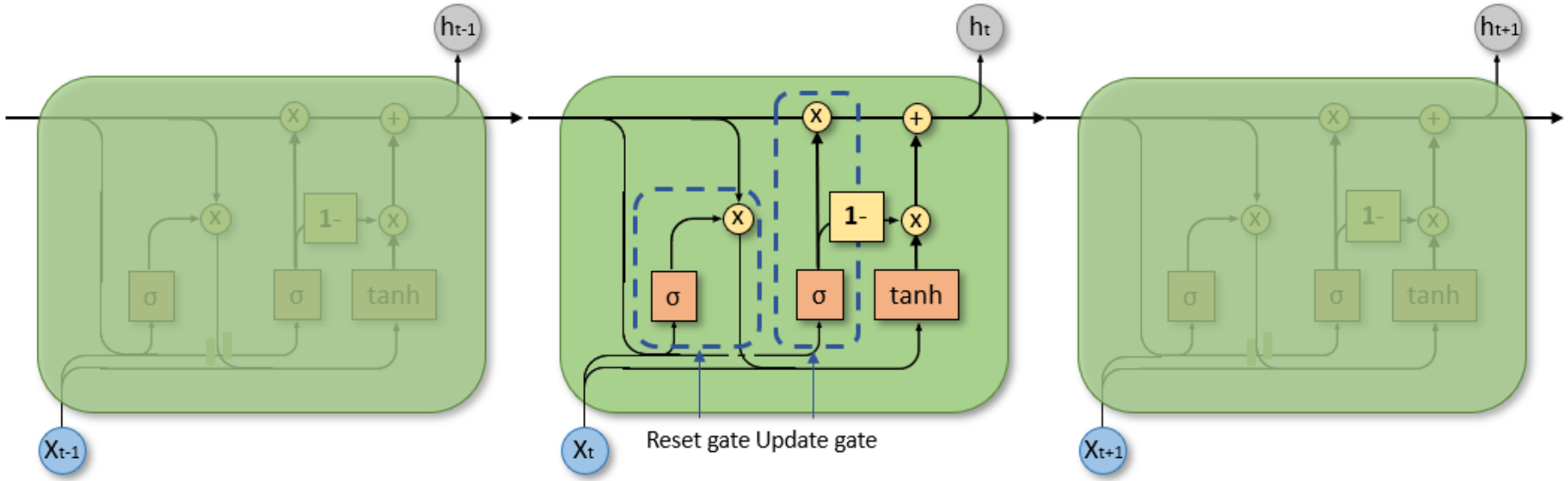
Shortcomings of LSTM

- Computationally expensive
- Difficult to train on very long sequences
- Still sequential (limits parallelism)
- Many parameters per gate (inefficiency)

Need for simplification and improvement.

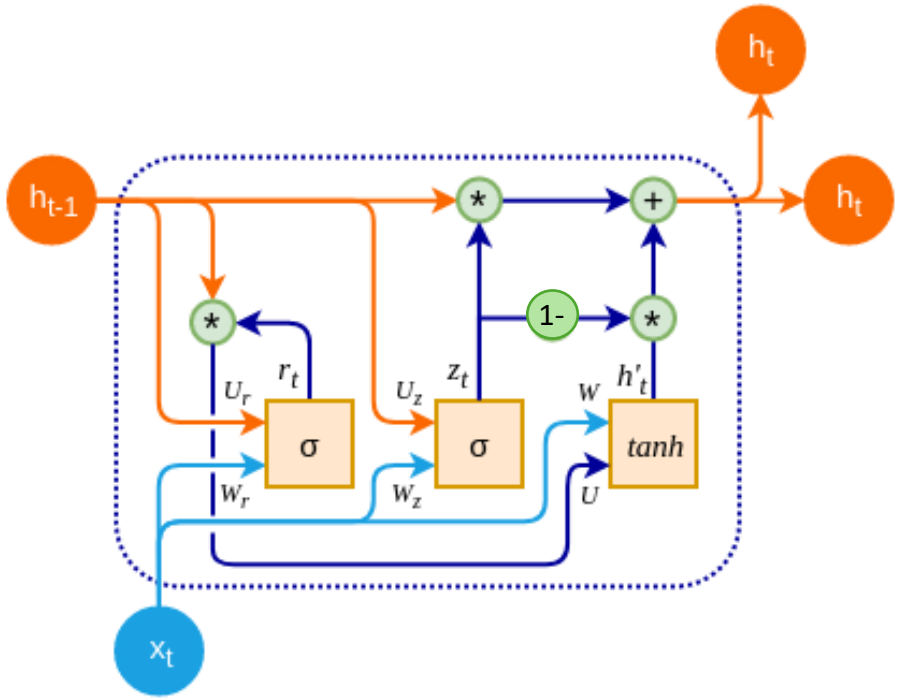
Enter GRU (Gated Recurrent Unit)

- **Proposed by:** Cho et al., 2014
- **Goal:** Simplify LSTM while retaining performance.
  - Combines forget and input gates into a single update gate.
  - No separate cell state.

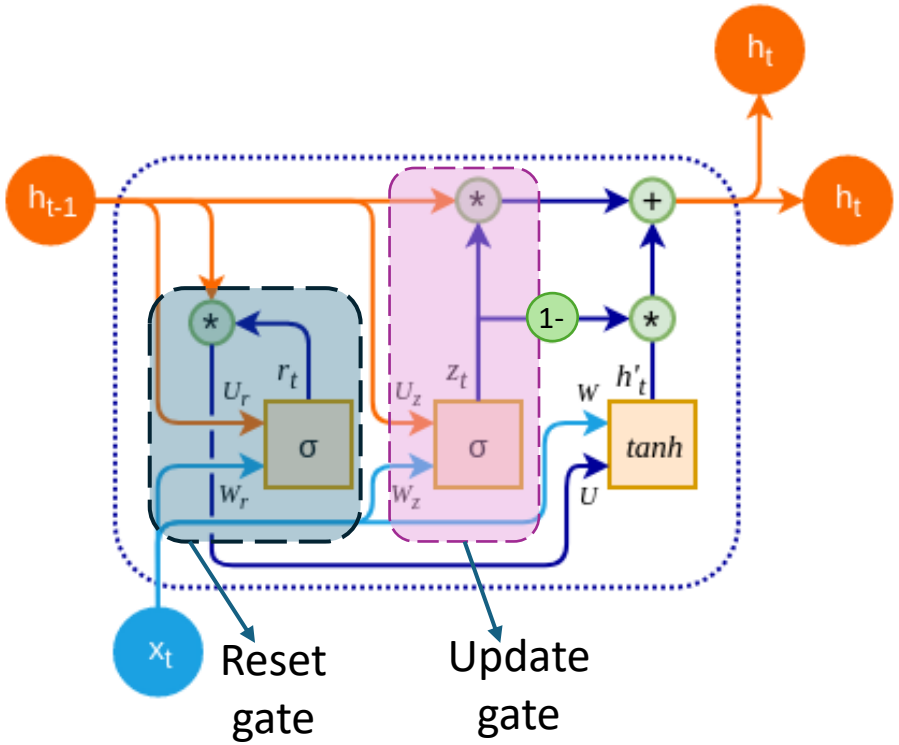


Enter GRU (Gated Recurrent Unit)

- **Proposed by:** Cho et al., 2014
- **Goal:** Simplify LSTM while retaining performance.
  - Combines forget and input gates into a single update gate.
  - No separate cell state.

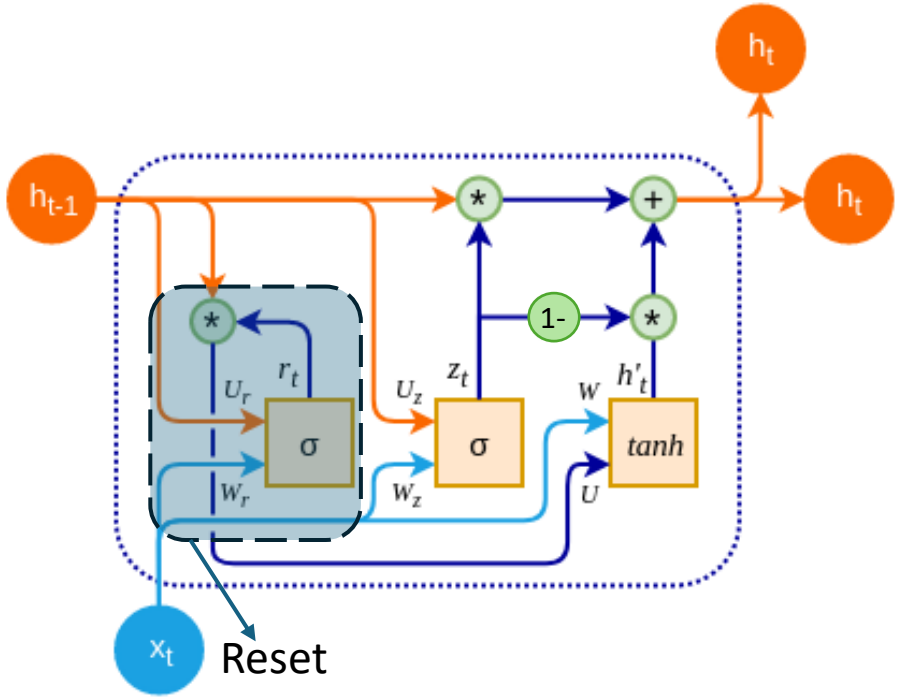


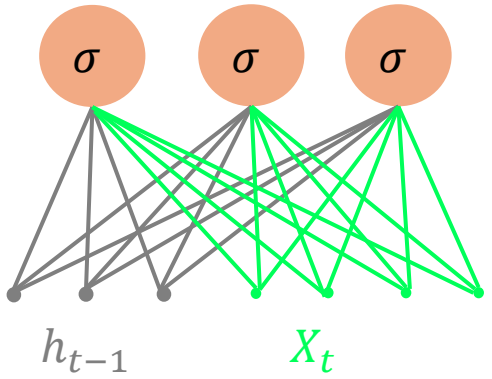
GRU Architecture and Equations



- Reset Gate:
  - Responsible for the short-term memory
  - Decides how much past information is kept and disregarded
- Update Gate:
  - Responsible for the long-term memory
  - Comparable to the LSTM's forget gate

1. Reset Gate



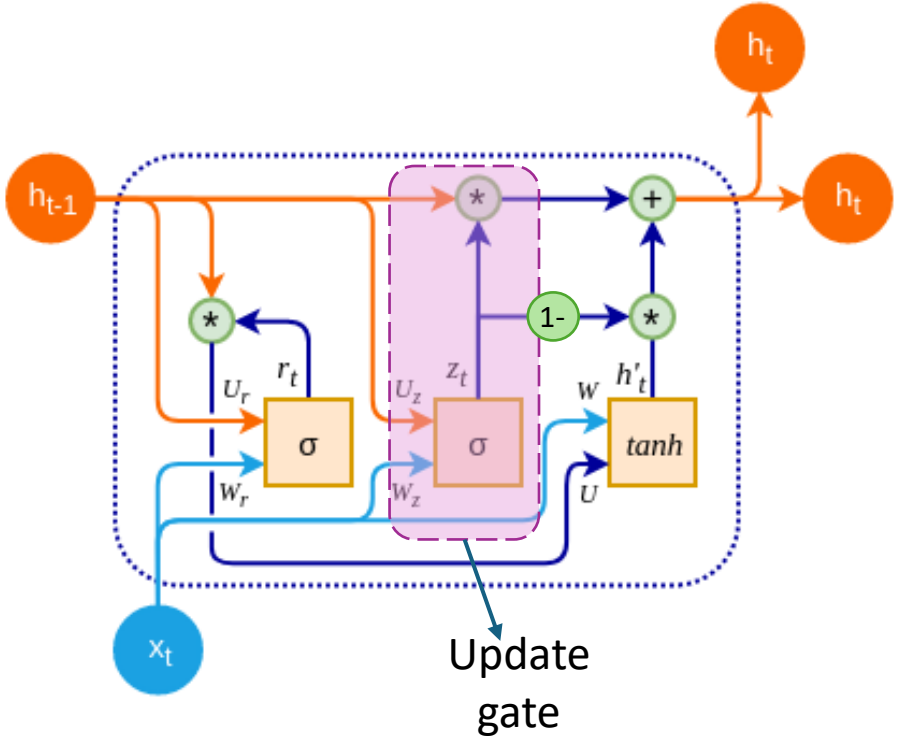


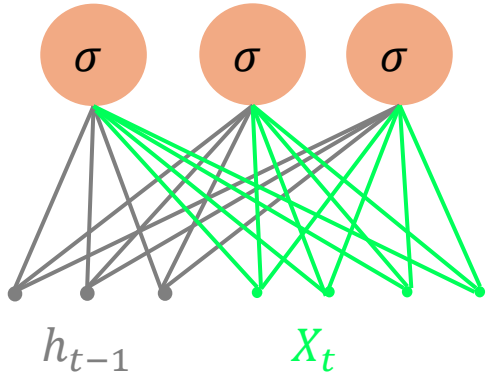
$$r_t = \sigma(W_r[h_{t-1}, X_t] + b_r)$$

⊗ → Elementwise multiplication

$U_r = r_t \otimes h_{t-1}$

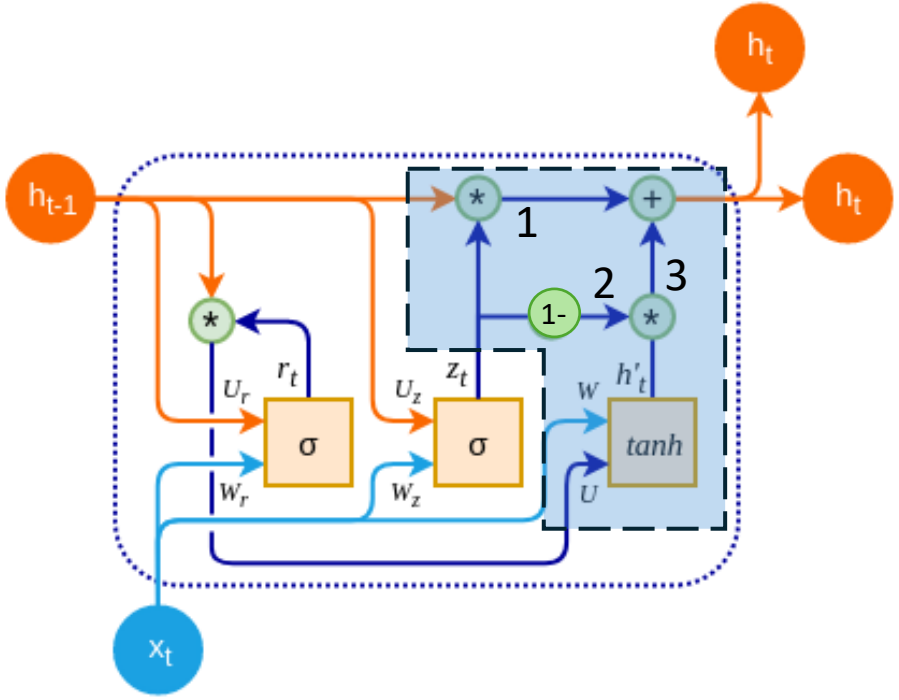
2. Update Gate

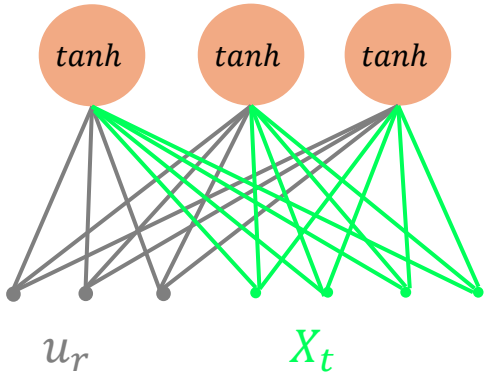




$$z_t = \sigma(W_z[h_{t-1}, X_t] + b_z)$$

Final hidden state calculation





Candidate hidden state:

$$\tilde{h}_t = \tanh(W_h[r_t \otimes h_{t-1}, X_t] + b_h)$$

1. $z_t \otimes h_{t-1}$
2. $1 - z_t$
3. $(1 - z_t) \otimes \tilde{h}_t$

Final hidden state:

$$h_t = (1 - z_t) \otimes \tilde{h}_t + z_t \otimes h_{t-1}$$

GRU vs LSTM

Feature	LSTM	GRU
Gates	3 (Input, forget, output)	2 (update, reset)
Cell state	Yes	No (uses hidden state only)
Complexity	Higher	Lower
Training Time	Longer	Shorter
Performance	Slightly better for long datasets	Competitive or better on small datasets

Summary

- RNNs introduced temporal modeling but suffer from gradient issues.
- LSTM adds gates and memory to manage long-term dependencies.
- GRU simplifies LSTM with fewer gates and competitive performance.

Reference

1. Hochreiter & Schmidhuber (1997). "Long Short-Term Memory."
2. Cho et al. (2014). "Learning Phrase Representations with RNN Encoder–Decoder."
3. https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4/