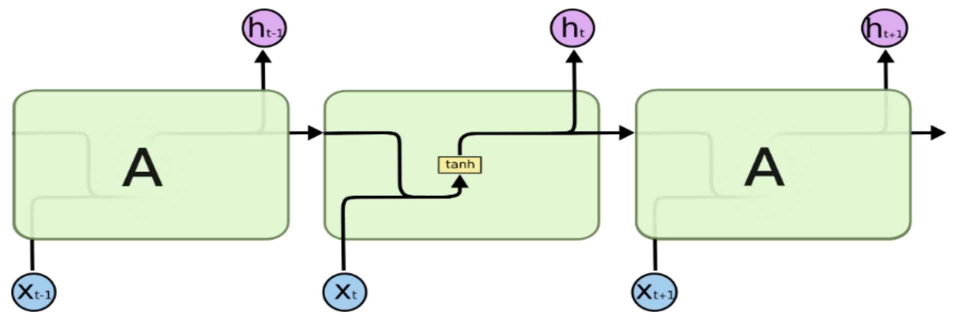
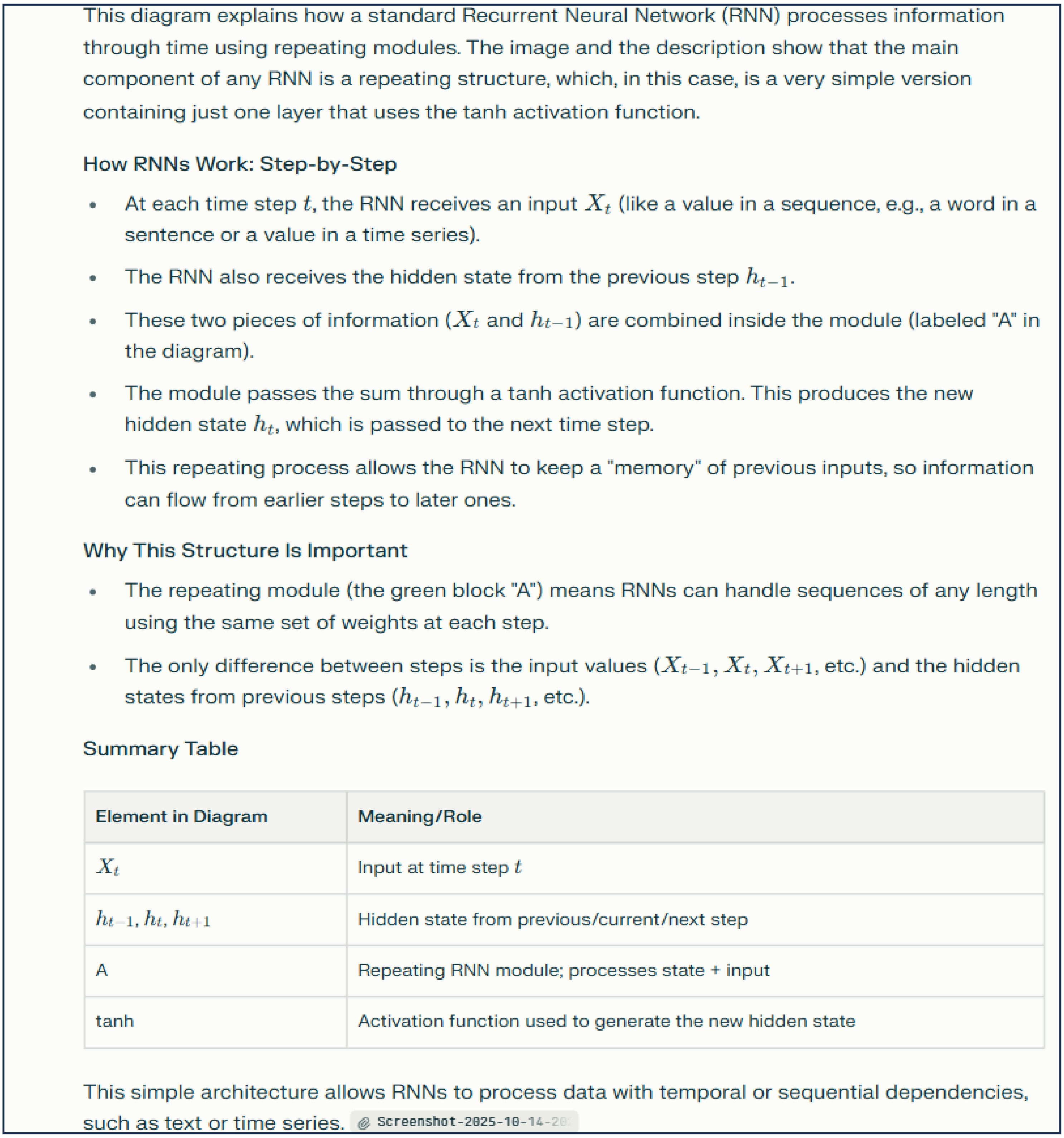
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



The repeating module in a standard RNN contains a single layer.

RNN Architecture



This diagram explains how a standard Recurrent Neural Network (RNN) processes information through time using repeating modules. The image and the description show that the main component of any RNN is a repeating structure, which, in this case, is a very simple version containing just one layer that uses the tanh activation function.

How RNNs Work: Step-by-Step

- At each time step t, the RNN receives an input X_t (like a value in a sequence, e.g., a word in a sentence or a value in a time series).
- The RNN also receives the hidden state from the previous step h_{t-1}.
- These two pieces of information (X_t and h_{t-1}) are combined inside the module (labeled "A" in the diagram).
- The module passes the sum through a tanh activation function. This produces the new hidden state h_t, which is passed to the next time step.
- This repeating process allows the RNN to keep a "memory" of previous inputs, so information can flow from earlier steps to later ones.

Why This Structure Is Important

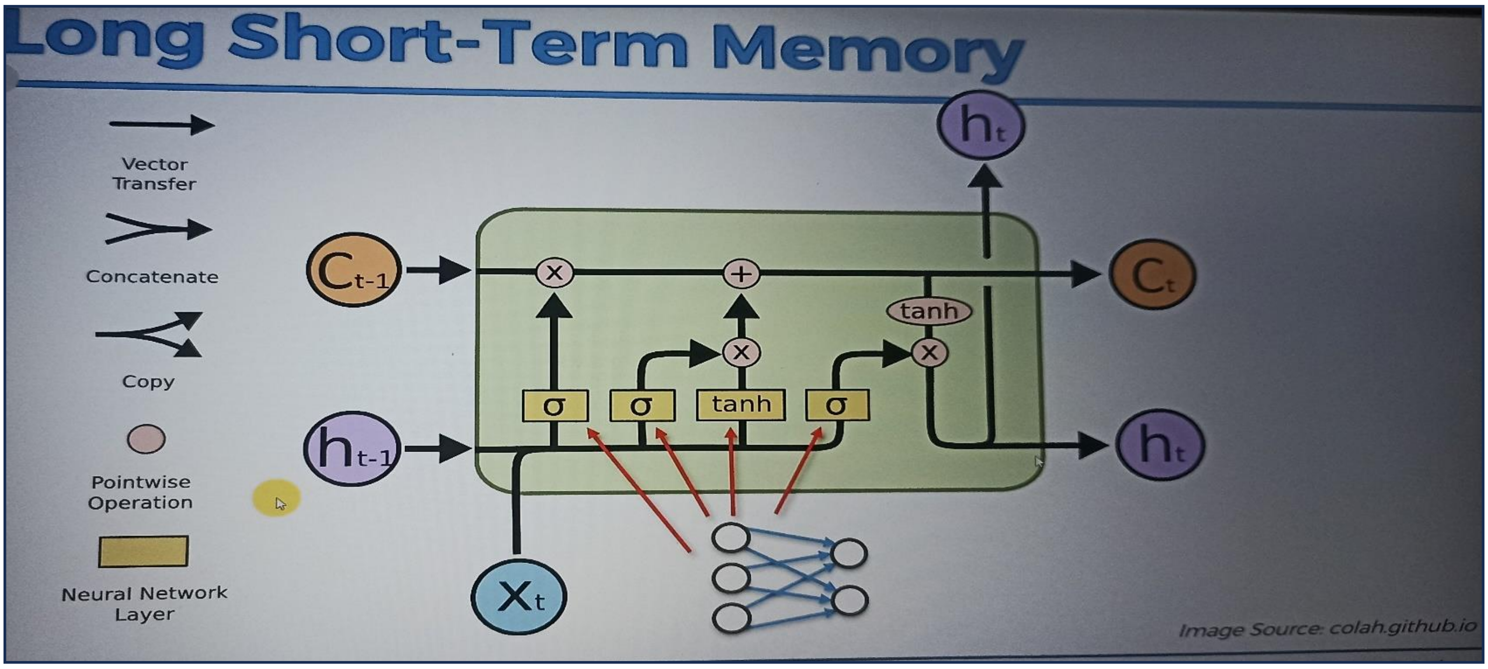
- The repeating module (the green block "A") means RNNs can handle sequences of any length using the same set of weights at each step.
- The only difference between steps is the input values (X_{t-1}, X_t, X_{t+1}, etc.) and the hidden states from previous steps (h_{t-1}, h_t, h_{t+1}, etc.).

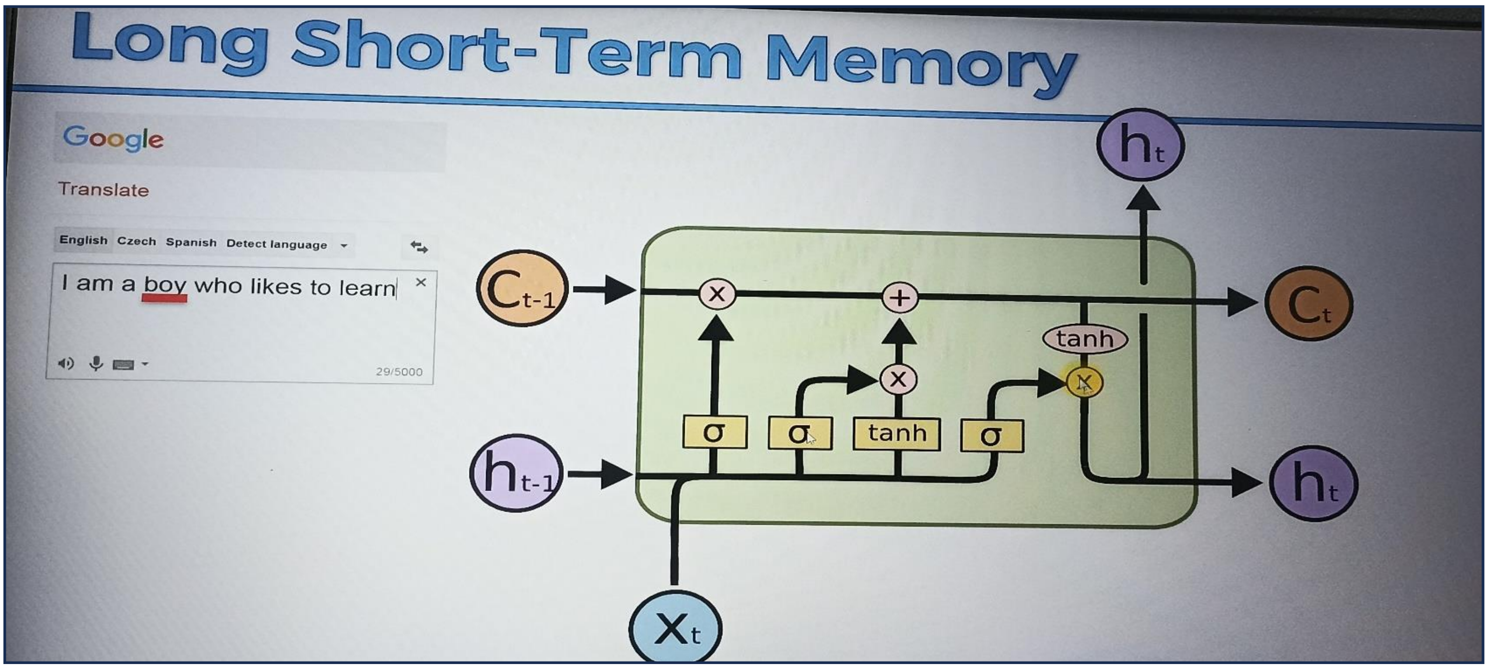
Summary Table

Element in Diagram	Meaning/Role
X_t	Input at time step t
h_{t-1}, h_t, h_{t+1}	Hidden state from previous/current/next step
A	Repeating RNN module; processes state + input
tanh	Activation function used to generate the new hidden state

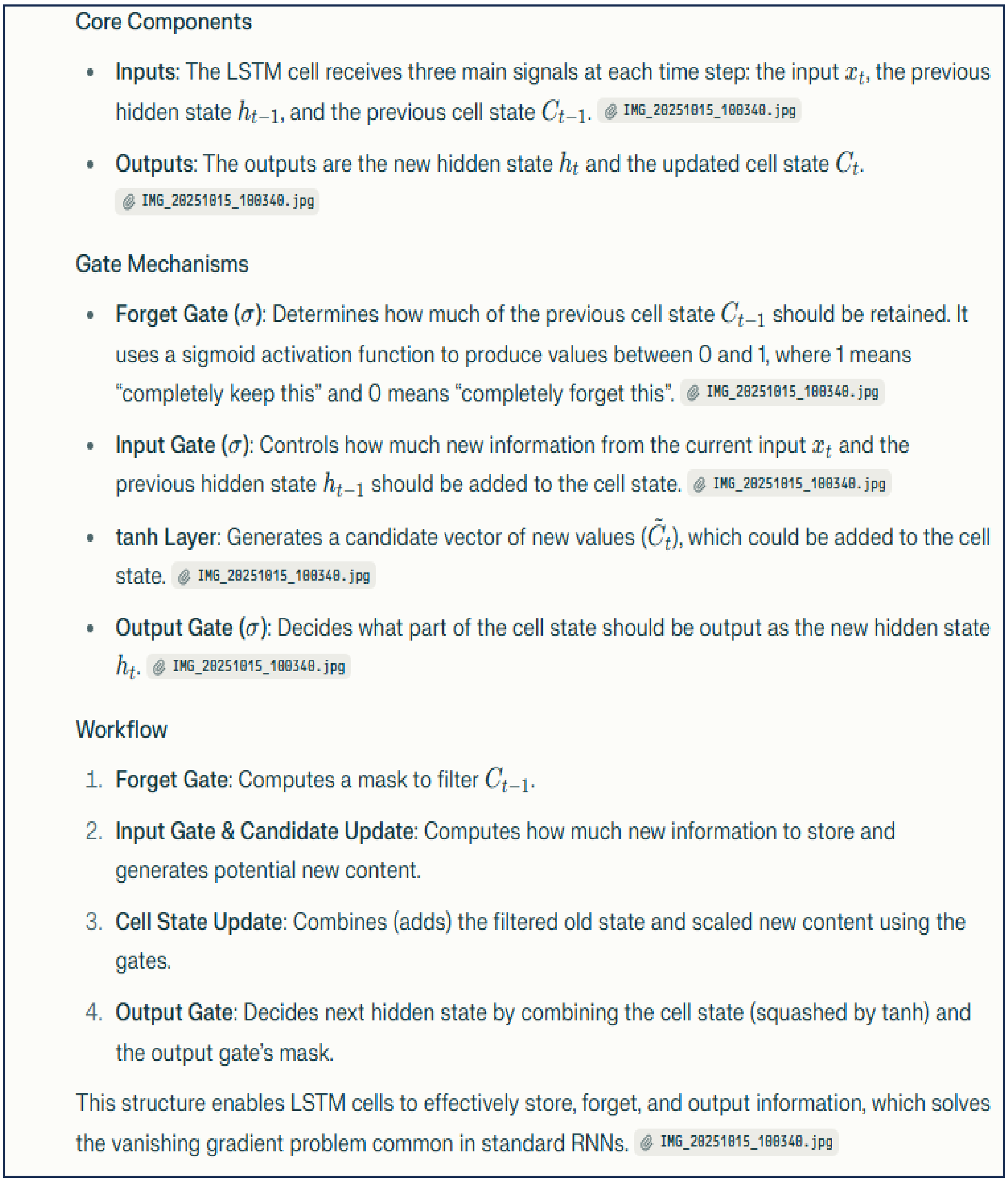
This simple architecture allows RNNs to process data with temporal or sequential dependencies, such as text or time series. Screenshot-2025-10-14-20

LSTM Architecture





Refer this Image in https://colah.github.io/posts/2015-08-Understanding-LSTMs/


Core Components

- **Inputs**: The LSTM cell receives three main signals at each time step: the input x_t, the previous hidden state h_{t-1}, and the previous cell state C_{t-1}. IMG_20251015_100340.jpg
- **Outputs**: The outputs are the new hidden state h_t and the updated cell state C_t. IMG_20251015_100340.jpg

Gate Mechanisms

- **Forget Gate** (σ): Determines how much of the previous cell state C_{t-1} should be retained. It uses a sigmoid activation function to produce values between 0 and 1, where 1 means "completely keep this" and 0 means "completely forget this". IMG_20251015_100340.jpg
- **Input Gate** (σ): Controls how much new information from the current input x_t and the previous hidden state h_{t-1} should be added to the cell state. IMG_20251015_100340.jpg
- **tanh Layer**: Generates a candidate vector of new values ($\tilde{C}_t$), which could be added to the cell state. IMG_20251015_100340.jpg
- **Output Gate** (σ): Decides what part of the cell state should be output as the new hidden state h_t. IMG_20251015_100340.jpg

Workflow

1. **Forget Gate**: Computes a mask to filter C_{t-1}.
2. **Input Gate & Candidate Update**: Computes how much new information to store and generates potential new content.
3. **Cell State Update**: Combines (adds) the filtered old state and scaled new content using the gates.
4. **Output Gate**: Decides next hidden state by combining the cell state (squashed by tanh) and the output gate's mask.

This structure enables LSTM cells to effectively store, forget, and output information, which solves the vanishing gradient problem common in standard RNNs. IMG_20251015_100340.jpg

The "previous cell state" in LSTM refers to the long-term memory value (*Ct*−1) that was stored by the LSTM cell at the preceding time step. This cell state preserves important information from earlier in the sequence and helps the LSTM maintain context over long periods.

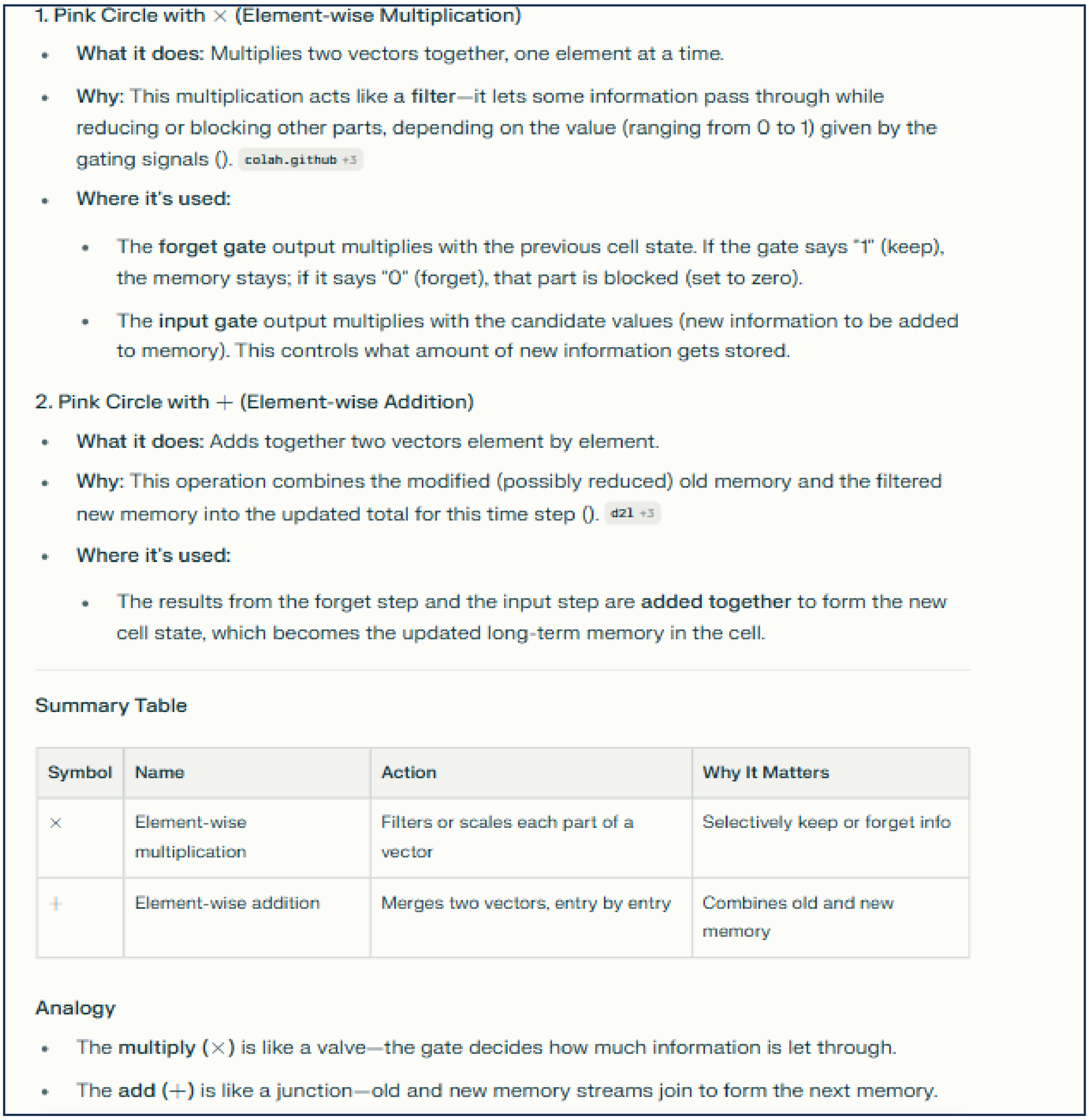
At each time step, LSTM uses this previous cell state in combination with the current input and hidden state to decide what information to retain, what to discard, and what new data to add. This selective memory process allows LSTM networks to model sequences with long-range dependencies by carrying forward relevant information as the cell state.

The previous hidden state (*ht*−1) and previous cell state (*Ct*−1) in LSTM are distinct and play different roles:

- Previous Cell State (*Ct*−1): This is the LSTM's long-term memory. It carries forward important information through many time steps and is updated by the forget and input gates. It acts as a conveyor belt, allowing data to flow mostly unchanged, unless actively modified by the cell's mechanisms.

- Previous Hidden State (*ht*−1): This is the short-term memory or output from the previous time step. It is used as input alongside the current input (xt) to determine the gate activations and to form the next output hidden state (ht). The hidden state is sent to other layers or used for prediction, whereas the cell state is not directly output.

In summary: cell state (Ct−1) is for long-term memory and context propagation, while hidden state (ht−1) is the short-term output that influences decisions and predictions at each step.



1. Pink Circle with × (Element-wise Multiplication)

- **What it does:** Multiplies two vectors together, one element at a time.
- **Why:** This multiplication acts like a **filter**—it lets some information pass through while reducing or blocking other parts, depending on the value (ranging from 0 to 1) given by the gating signals (). colah.github +3
- **Where it's used:**
  - The **forget gate** output multiplies with the previous cell state. If the gate says "1" (keep), the memory stays; if it says "0" (forget), that part is blocked (set to zero).
  - The **input gate** output multiplies with the candidate values (new information to be added to memory). This controls what amount of new information gets stored.

2. Pink Circle with + (Element-wise Addition)

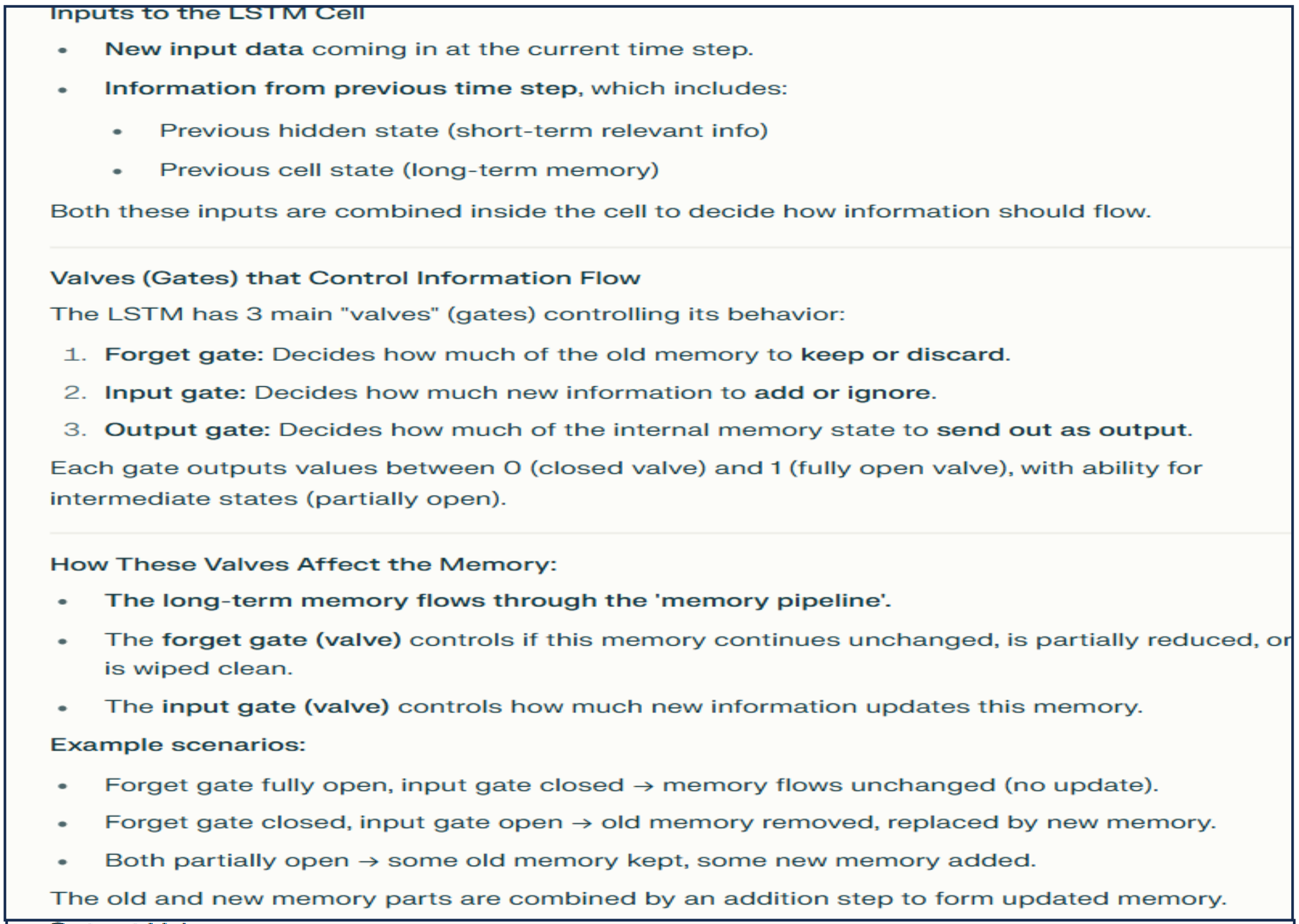
- **What it does:** Adds together two vectors element by element.
- **Why:** This operation combines the modified (possibly reduced) old memory and the filtered new memory into the updated total for this time step (). d2l +3
- **Where it's used:**
  - The results from the forget step and the input step are **added together** to form the new cell state, which becomes the updated long-term memory in the cell.

Summary Table

Symbol	Name	Action	Why It Matters
×	Element-wise multiplication	Filters or scales each part of a vector	Selectively keep or forget info
+	Element-wise addition	Merges two vectors, entry by entry	Combines old and new memory

Analogy

- The **multiply** (×) is like a valve—the gate decides how much information is let through.
- The **add** (+) is like a junction—old and new memory streams join to form the next memory.



Inputs to the LSTM Cell

- **New input data** coming in at the current time step.
- **Information from previous time step**, which includes:
  - Previous hidden state (short-term relevant info)
  - Previous cell state (long-term memory)

Both these inputs are combined inside the cell to decide how information should flow.

Valves (Gates) that Control Information Flow

The LSTM has 3 main "valves" (gates) controlling its behavior:

1. **Forget gate:** Decides how much of the old memory to **keep or discard**.
2. **Input gate:** Decides how much new information to **add or ignore**.
3. **Output gate:** Decides how much of the internal memory state to **send out as output**.

Each gate outputs values between 0 (closed valve) and 1 (fully open valve), with ability for intermediate states (partially open).

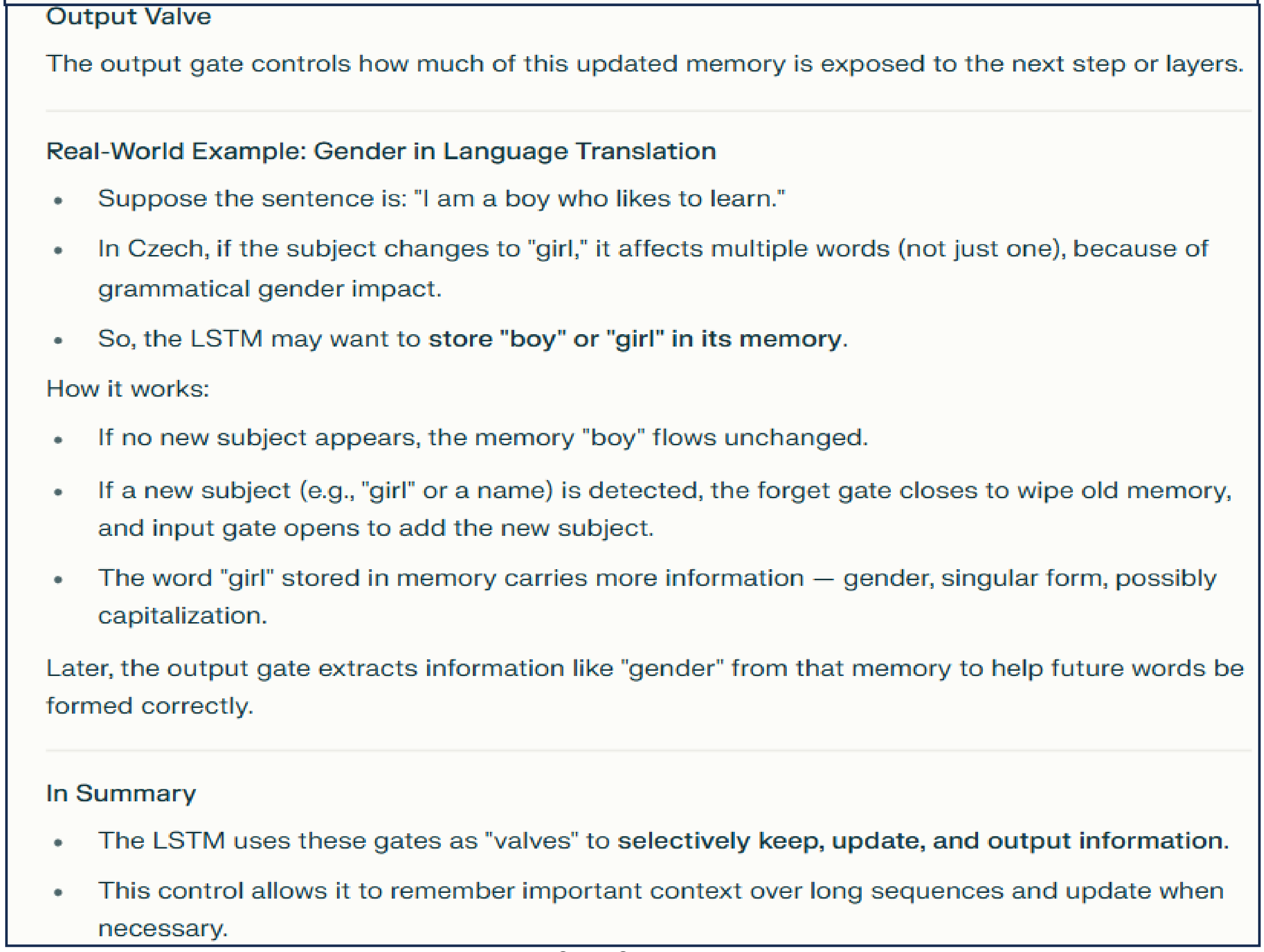
How These Valves Affect the Memory:

- **The long-term memory flows through the 'memory pipeline'.**
- The **forget gate (valve)** controls if this memory continues unchanged, is partially reduced, or is wiped clean.
- The **input gate (valve)** controls how much new information updates this memory.

Example scenarios:

- Forget gate fully open, input gate closed → memory flows unchanged (no update).
- Forget gate closed, input gate open → old memory removed, replaced by new memory.
- Both partially open → some old memory kept, some new memory added.

The old and new memory parts are combined by an addition step to form updated memory.



Output Valve

The output gate controls how much of this updated memory is exposed to the next step or layers.

Real-World Example: Gender in Language Translation

- Suppose the sentence is: "I am a boy who likes to learn."
- In Czech, if the subject changes to "girl," it affects multiple words (not just one), because of grammatical gender impact.
- So, the LSTM may want to **store "boy" or "girl" in its memory**.

How it works:

- If no new subject appears, the memory "boy" flows unchanged.
- If a new subject (e.g., "girl" or a name) is detected, the forget gate closes to wipe old memory, and input gate opens to add the new subject.
- The word "girl" stored in memory carries more information — gender, singular form, possibly capitalization.

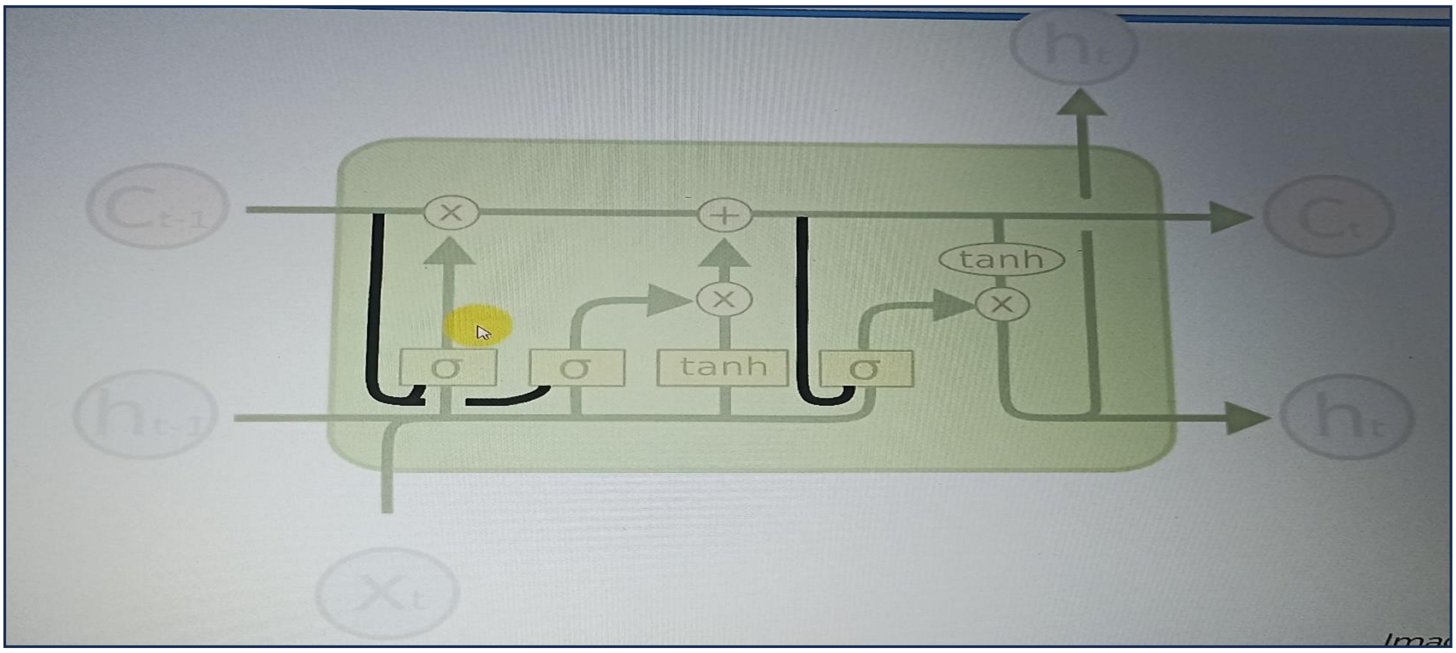
Later, the output gate extracts information like "gender" from that memory to help future words be formed correctly.

In Summary

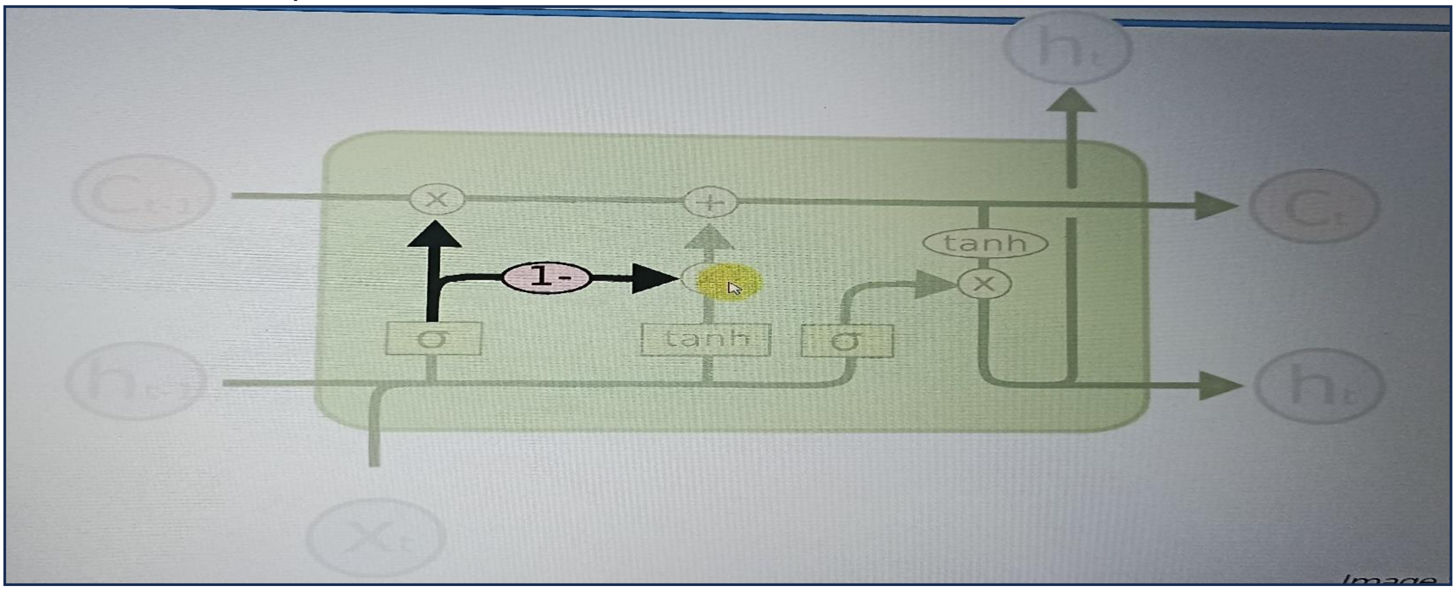
- The LSTM uses these gates as "valves" to **selectively keep, update, and output information.**
- This control allows it to remember important context over long sequences and update when necessary.

LSTM Variations:

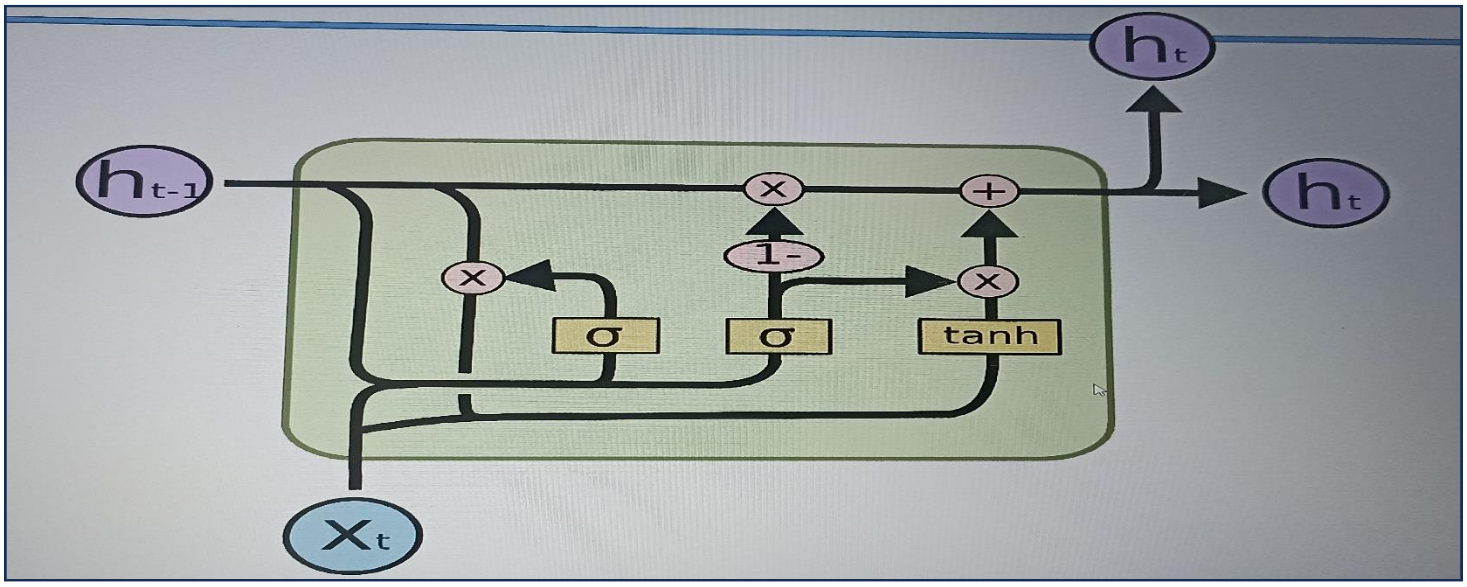
1. Peephole connections

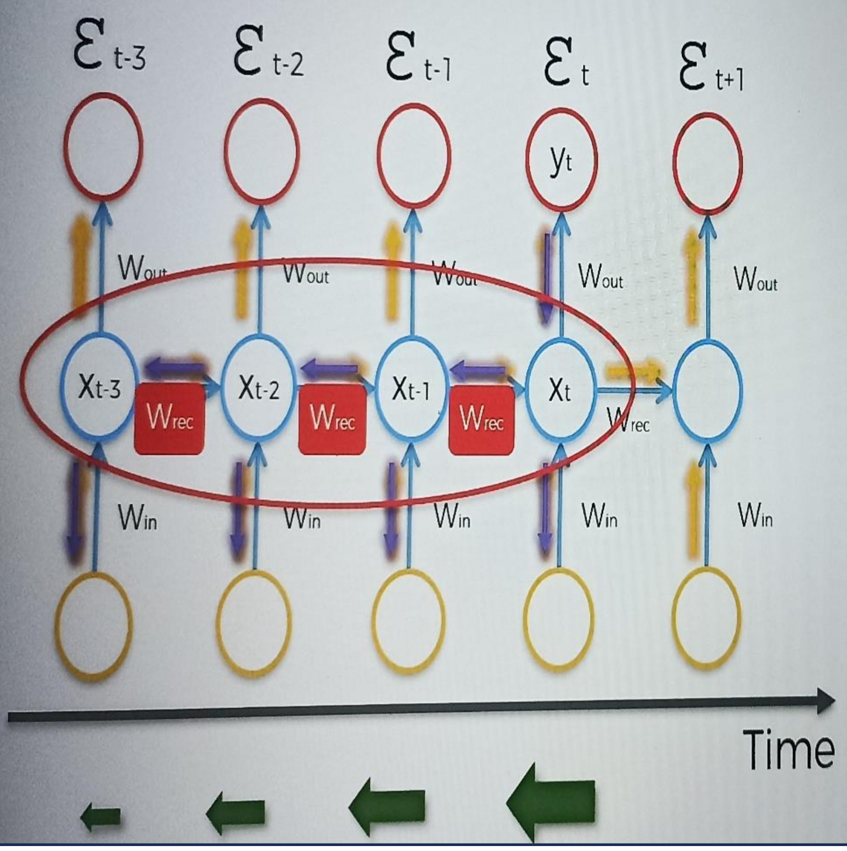


2. Coupled Gates



3. GRU





$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1\leq t\leq T} \frac{\partial \mathcal{E}_t}{\partial \theta} \qquad (3)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1\leq k\leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \qquad (4)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t\geq i>k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t\geq i>k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \qquad (5)$$

W_{rec} < 1 ⇨ Vanishing

W_{rec} ~ large ⇨ Exploding

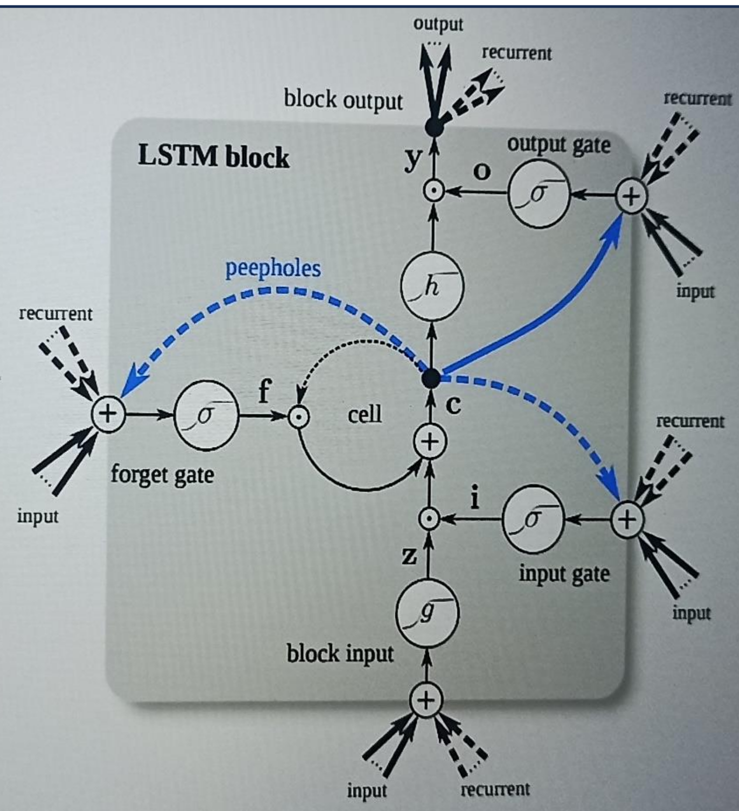


Image Source: arxiv.org/pdf/1503.04069.pdf