Great question 👏 — LSTMs (Long Short-Term Memory networks) are a special kind of RNN designed to overcome the problem of **vanishing and exploding gradients** in normal RNNs.

They work by introducing a **memory cell** and a **set of gates (forget, input, output)** that carefully control how information flows through time.

I’ll break it down step by step with **formulas, images (conceptually drawn), and explanations** — including why we use multiplication (×), addition (+), copy (→), and merge (⊕).

**🔹 1. The Core Idea of LSTM**

* Standard RNNs just apply:  
  ht=tanh⁡(W⋅[ht−1,xt])h\_t = \tanh(W \cdot [h\_{t-1}, x\_t])  
  → But this loses long-term dependencies.
* LSTM introduces:
  + **Cell state CtC\_t**: like a conveyor belt (long-term memory).
  + **Hidden state hth\_t**: short-term memory (what’s output to next layer).
  + **Gates**: tiny neural networks that decide *what to keep, update, or output*.

**🔹 2. Structure of an LSTM Cell**

At each timestep tt, we have:

* Input vector: xtx\_t
* Previous hidden state: ht−1h\_{t-1}
* Previous cell state: Ct−1C\_{t-1}

The LSTM outputs:

* New hidden state: hth\_t
* New cell state: CtC\_t

**🔹 3. The Gates and Memory Flow**

**(a) Forget Gate**

**Formula:**

ft=σ(Wf⋅[ht−1,xt]+bf)f\_t = \sigma(W\_f \cdot [h\_{t-1}, x\_t] + b\_f)

* ft∈[0,1]f\_t \in [0,1]: a mask that decides how much of the old memory to keep.
* σ\sigma = sigmoid → outputs between 0 and 1.

**Operation:**

Ct−1×ftC\_{t-1} \times f\_t

Multiplication (×) here = “keep or forget” (0 → delete, 1 → keep).

📘 Example:  
If we’re predicting a sentence and see “not”, the forget gate may drop irrelevant older memory.

**(b) Input Gate & Candidate Memory**

**Input Gate Formula:**

it=σ(Wi⋅[ht−1,xt]+bi)i\_t = \sigma(W\_i \cdot [h\_{t-1}, x\_t] + b\_i)

**Candidate Memory Formula:**

C~t=tanh⁡(Wc⋅[ht−1,xt]+bc)\tilde{C}\_t = \tanh(W\_c \cdot [h\_{t-1}, x\_t] + b\_c)

* iti\_t: controls how much of the new info we allow in.
* C~t\tilde{C}\_t: candidate content to add.

**Operation:**

it×C~ti\_t \times \tilde{C}\_t

Multiplication ensures only *important* candidate info enters memory.

**(c) Update Cell State**

**Formula:**

Ct=ft×Ct−1+it×C~tC\_t = f\_t \times C\_{t-1} + i\_t \times \tilde{C}\_t

* Forget part: ft×Ct−1f\_t \times C\_{t-1}
* Add new info: it×C~ti\_t \times \tilde{C}\_t
* Addition (+) merges them.

📘 Conveyor belt analogy → Old memory flows, modified with updates.

**(d) Output Gate**

**Formula:**

ot=σ(Wo⋅[ht−1,xt]+bo)o\_t = \sigma(W\_o \cdot [h\_{t-1}, x\_t] + b\_o)

**Hidden State Update:**

ht=ot×tanh⁡(Ct)h\_t = o\_t \times \tanh(C\_t)

* Output gate decides *what part of memory to show*.
* Cell state passes through tanh⁡\tanh → squashes values.
* Multiplication (×) filters the output.

**🔹 4. Visual Flow (ASCII Diagram)**

┌─────────────┐

x\_t ---------->│ │

h\_{t-1} ------>│ Gates │---> f\_t, i\_t, o\_t

└─────┬───────┘

│

┌────────┴─────────┐

│ Forget: f\_t \* C\_{t-1}

│ Input: i\_t \* C̃\_t

└────────┬─────────┘

│ (+ merge)

v

New Cell State C\_t

│

tanh(C\_t) × o\_t

│

New Hidden h\_t

**🔹 5. Why Multiplication, Addition, Copy, Merge?**

* **Multiplication (×):** Acts like a gate/valve → scale information (0 = block, 1 = pass).
* **Addition (+):** Combines new and old memory without overwriting (important for long-term storage).
* **Copy (→):** Conveyor belt mechanism → allows unmodified memory to flow.
* **Merge (⊕):** Controlled combination of old + new information.

**🔹 6. Summary**

* **Forget gate** → What to erase from old memory.
* **Input gate + candidate** → What new info to add.
* **Cell state update** → Combine old + new info.
* **Output gate** → What to output as hidden state.
* **Trick:** Sigmoid (σ\sigma) decides importance (0–1), Tanh (−1 to 1) regulates scale.

👉 Would you like me to also **draw you a clean diagram (with labeled gates, arrows, and operations like × and +)** so you can visualize it better?