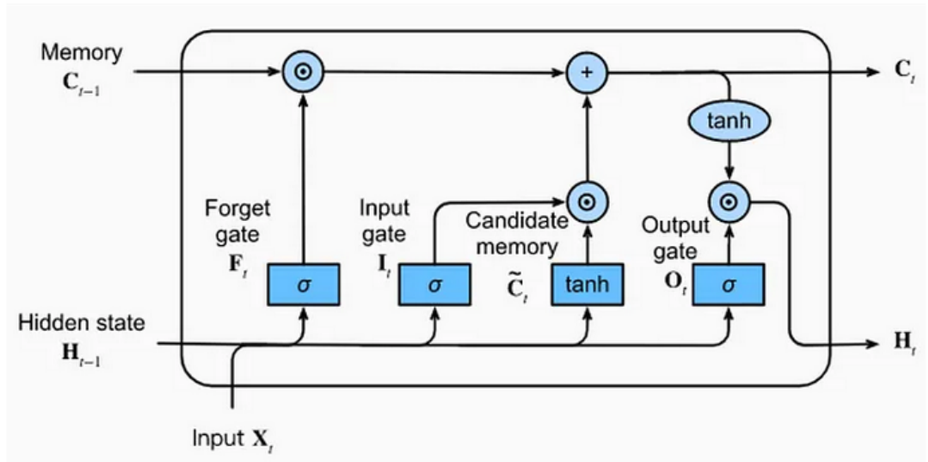
LSTM(Long Short Term Memory)

Why LSTM ? Not Rnn

A standard RNN updates its hidden state as:
ht=tanh(Whht−1+Wxxt).

1. During backpropagation through time(BPTT):
Gradients are repeatedly multiplied by the same weight matrix.
2.This causes vanishing gradients (values → 0) or exploding gradients (values → ∞).



Practical consequence:
1.RNNs forget information quickly
2.They struggle when important context appears many time steps earlier.

LSTM performs better than RNN on:
1.Language modeling
2.Machine translation
3.Speech recognition
4.Time-series forecasting
5.POS tagging and NER

This is why almost all classical NLP pipelines replaced RNNs with LSTMs before Transformers.

Why tanh is stable activation function for rnn and lstm?

tanh is considered stable because it is bounded, zero-centered, has a smooth derivative, and a maximum gradient of 1, making it well-suited for repeated application across time steps in recurrent architectures. It avoids the exploding activations of ReLU and the biased gradients of sigmoid, while working harmoniously with gating mechanisms in LSTMs to preserve long-term dependencies.

tanh does not eliminate vanishing gradients by itself.

What eliminates vanishing gradients in LSTM is:

ct=ft⊙ct−1+it⊙c~t
tanh simply does not make the problem worse.

Why LSTM uses tanh specifically?

In LSTM:
1.sigmoid → control gates (how much to allow)
2.tanh → content values (what to store/output)

Example:
c~t=tanh(Wc[ht−1,xt])

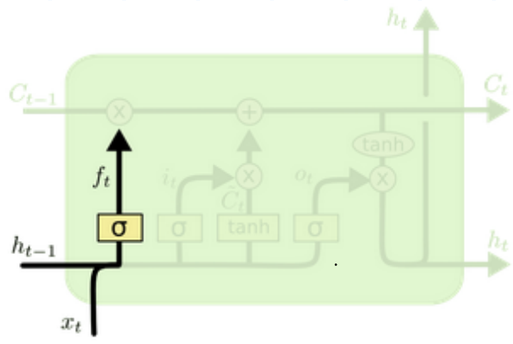
Why this is ideal:

1.Keeps memory values bounded
2.Allows both positive and negative information
3.Stable gradient flow into cell state

LSTM Arcitecture and gates

Forget Gate:

Decides how much of the old memory to keep.
ft=σ(Wf[ht−1,xt]+bf)
Output range: [0, 1]
0 → forget everything
1 → keep everything
if earlier information is no longer relevant, the forget gate removes it.

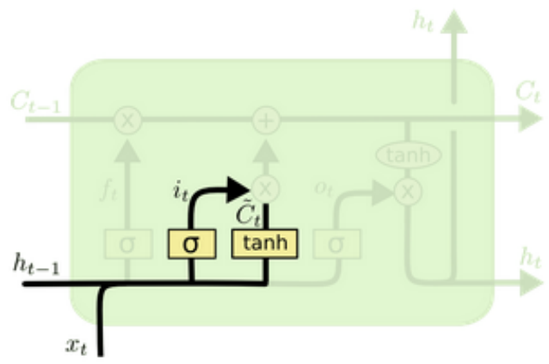


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Input Gate:

4. Input Gate — “What should I write?”
Step 1: Decide where to write
it=σ(Wi[ht−1,xt]+bi)

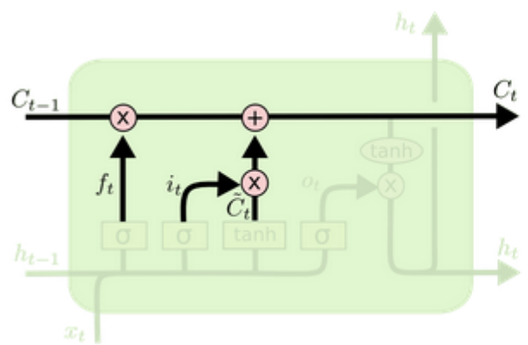
Step 2: Decide what to write
c~t=tanh(Wc[ht−1,xt]+bc)



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Cell State Update:

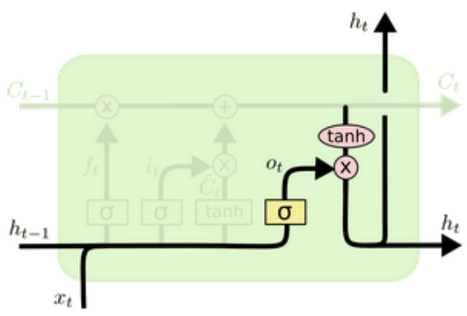
This is the most important equation.
ct=ft⊙ct−1+it⊙c~t
What happens here:
Old memory is scaled, not overwritten
New memory is selectively added
Gradient can flow through time almost unchanged
This is why LSTM solves vanishing gradients.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate:

Step 1: Decide how much to expose
ot=σ(Wo[ht−1,xt]+bo)
Step 2: Generate hidden state
ht=ot⊙tanh(ct)



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Key idea:
Cell state = internal memory
Hidden state = controlled view of that memory

One complete LSTM step (ordered)

At time step t:
Concatenate [h_{t-1}, x_t]
Compute forget gate f_t
Compute input gate i_t
Compute candidate memory \tilde{c}_t
Update cell state c_t
Compute output gate o_t
Compute hidden state h_t