RNN

LSTM → Long (short term memory

is a special type of RNN, designed to handle Long Term Dependencies and solve vanishing Gradient Problem. It handles both long term & short term dependencies

Imp

- Standard RNNs forget earlier context in long sequences.
- LSTM solves this by introducing a **cell state** — a "conveyor belt" that runs through the sequence, carrying information with minimal changes.
- It uses **gates** to control what to keep, forget, and output.

Think of LSTM as **a smart memory manager** that knows exactly when to remember and when to forget.

2 LSTM Structure

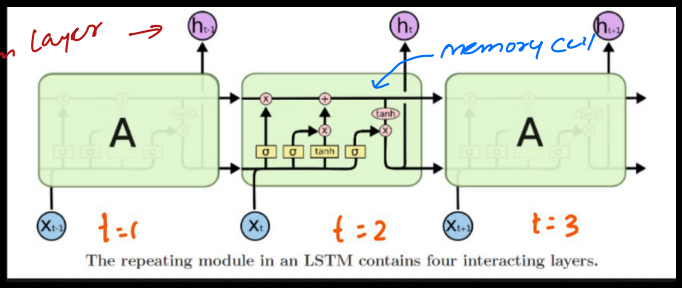
LSTM has **three gates**:

1. **Forget Gate (f_t)** – Decides **what past information to erase**.
2. **Input Gate (i_t)** – Decides **what new information to store**.
3. **Output Gate (o_t)** – Decides **what part of the memory to output** as the hidden state.

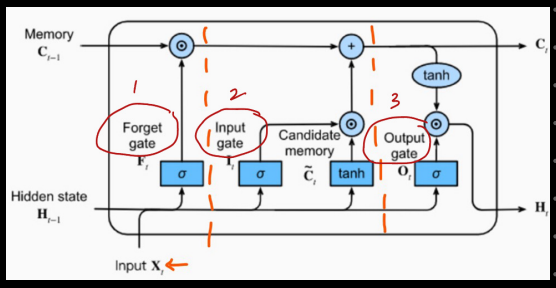
It maintains **two states**:

- **Cell state (C_t)** → Long-term memory. → Memory Cell (Conveyor Belt)
- **Hidden state (h_t)** → Short-term/contextual memory.

Hidden Layer →



The repeating module in an LSTM contains four interacting layers.



Imp

LSTM Computation ; Memory Cell = Conveyor Belt

🧠 **Example Sentence:**

> "The dog chased the **cat**"

At the word **"cat"**, here's what happens:

🔹 **1. Forget Gate**

> "What can I forget from earlier?"
> → Maybe "the dog" isn't needed now.

🔹 **2. Input Gate**

> "Is this new word ('cat') important?"
> → Yes — likely the object of the chase.

🔹 **3. Candidate Memory**

> "Based on 'cat' and previous context, what should I suggest adding to memory?"
> → Suggest: "Cat is the thing being chased."

🔹 **4. Update Memory**

> Forget old parts (like 'dog'), add accepted parts of the new idea ('cat being chased').

🔹 **5. Output Gate**

> "What should I use from this new memory to predict the next word?"
> → Maybe something like "into" (e.g., "into the yard").

✅ **Candidate memory** is just a **suggestion of new info** based on the input and context. The input gate decides whether to use it.

Step	Role
Input	The current word ("cat")
Forget gate	Decides which old ideas to discard ("the dog" isn't needed anymore)
Candidate memory	Suggests new meaning from current input ("cat" is object of action)
Input gate	Decides how much of that suggestion to accept (yes, cat is important)
Memory update	Combines past memory with accepted new info
Output gate	Chooses what parts of memory help predict the next word (e.g., "into")

VVImp

An **LSTM** (Long Short-Term Memory) is a special type of RNN designed to **remember information for longer** and **avoid the vanishing gradient problem** that simple RNNs suffer from.

Why it exists

- Simple RNNs can only remember information for a few timesteps before older info fades away.
- LSTMs add a **memory cell** and **gates** that control what to keep, what to forget, and what to output.

→ VVV.Imp

In simple words:

- **Forget gate:** "What should I erase from memory?"
- **Input gate:** "What new info should I add?"
- **Cell state:** The **memory highway** that carries important info across many timesteps.
- **Output gate:** "What part of memory do I share as output now?"

Memory Gate → maintains Memory (items)

↳ Forget Gate
Input Gate
Output Gate

→ Too many parameters due to 3 gates

LSTM problem = Complex / time Consuming

LSTM Variants

1. Classical LSTM
2. GRU LSTM
3. Bidirectional LSTM
4. LSTM with Attention.
5. Convolutional LSTM

- Classic LSTM: The original architecture with input, forget, and output gates, plus a memory cell to retain information across long sequences. [2] [3] [4]
- Bidirectional LSTM (BiLSTM): Processes input sequences in both forward and backward directions, helping capture both past and future context in sequence data. This is especially beneficial in tasks like language modeling and translation. [3] [2]
- Gated Recurrent Unit (GRU): A streamlined variant that merges the forget and input gates into a single "update" gate, simplifying the overall architecture and computation while often performing comparably to LSTM on many tasks. [1] [4] [2] [3]
- ConvLSTM (Convolutional LSTM): Integrates convolution operations into the LSTM architecture, making it suitable for spatial-temporal data such as video or images where both time and space information are important. [2]
- LSTMs with Attention: Enhance the LSTM model by allowing the network to focus on relevant parts of the input sequence selectively at each time step, which improves performance in complex tasks like translation or summarization. [2]

→ Forget + Input = update

2 Gates

GRU Gated Recurrent Unit

It is a simpler and faster version of LSTM

- In a standard RNN, gradients vanish or explode when sequences are long, so the network forgets earlier context.
- GRU fixes this using **gates** (like LSTM), but with **fewer parameters** and **no separate cell state**.

Think of GRU as **LSTM Lite** — keeps the essence, removes the extras.

2 GRU Structure

→ GRU →

GRU has **two gates** instead of LSTM's three:

1. **Update Gate (z_t)** – Decides **how much of the past information** to keep vs. update with new info.
2. **Reset Gate (r_t)** – Decides **how much of the past information to forget** before combining with new input.

Key difference from LSTM:

- **No separate cell state** → GRU keeps everything in a single **hidden state (h_t)**.
- This means **less memory** and **faster training**.

Imp: 2 Gates, No separate hidden state

3 Step-by-Step Working

At time step `t`:

1. **Update gate**

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

→ If `z_t` is close to 1 → keep most of the old memory.
If `z_t` is close to 0 → replace it with new info.

2. **Reset gate**

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

→ If `r_t` is close to 0 → forget the past completely.
If `r_t` is close to 1 → use full past info.

3. **Candidate hidden state**

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

→ New information considering the reset gate.

4. **Final hidden state**

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

→ A blend of old state and candidate state, controlled by update gate.

4 LSTM vs GRU — Interview-Ready Table

Feature	LSTM	GRU
Gates	3 (input, forget, output)	2 (update, reset)
Memory storage	Separate **cell state** and **hidden state**	Single **hidden state** only
Parameters	More (slower training)	Fewer (faster training)
Long-term dependencies	Better at capturing	Good but sometimes less effective
Complexity	Higher	Lower

How GRU Maintains Sequence Without a Separate Memory State

1️⃣ LSTM's Two-Memory Approach

- **Cell state (C_t)** → Long-term memory (like a storage hard drive).
- **Hidden state (h_t)** → Short-term, immediate output memory (like RAM).

This separation allows LSTM to explicitly keep long-term information mostly untouched in the **cell state**.

2️⃣ GRU's Single-Memory Approach

- GRU merges **cell state** and **hidden state** into **one hidden state (h_t)**.
- This single state stores **both long-term and short-term dependencies** in a **blended form**.

So, instead of having:

$$(C_t, h_t)$$

like LSTM, GRU just has:

$$h_t$$

3️⃣ How It Works

- **Update gate (z_t)** controls how much of the old h_{t-1} to keep.
  If z_t is high → past info is preserved (similar to cell state persistence in LSTM).
- **Reset gate (r_t)** decides how much past info to ignore when computing the new candidate state.

Key point: The update gate's mechanism effectively **lets old information flow unchanged through many time steps**, which is how GRU keeps memory **without explicitly having a cell state**.

4️⃣ Why This Still Works for Sequences

- Even though GRU has only one vector h_t, it:
  - **Retains sequence context** via gating (update gate works like a "forget + keep" controller).
  - **Learns to encode both long- and short-term information** in different parts of the same vector.
  - Passes information through time steps **selectively**, avoiding the vanishing gradient problem.

5️⃣ Analogy

Think of LSTM as having **two separate notebooks** — one for the full story (cell state) and one for today's notes (hidden state).
GRU says: *"Why not just have one notebook but write carefully so old important notes remain while adding new ones?"*

📌 Practical Steps to Build an LSTM Model for Text

1. **Preprocess text** → clean, tokenize, convert to sequences of integers.
2. **Word embeddings** → use Word2Vec, GloVe, or an embedding layer.
3. **Model architecture** → Sequential → Embedding → **LSTM layer** (instead of simple RNN) → Dense (softmax).
4. **Training** → cross-entropy loss + Adam optimizer.
5. **Evaluation** → accuracy, perplexity, or F1 score depending on task.
6. **Prediction** → next-word generation or text classification.

🎯 Interview-friendly one-liner

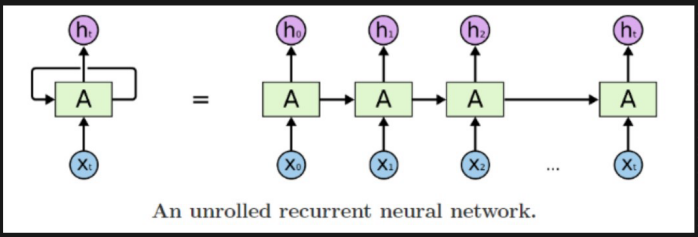
"I would preprocess the text, use embeddings, then build a Sequential model with an Embedding layer, an LSTM layer to capture long-term dependencies, and a Dense softmax output. I'd train with cross-entropy and Adam, evaluate, and finally use it for prediction."

LSTM RNN [Long Short Term Memory RNN]

RNN ⟶ Long Term Dependencies ⟶ Vanishing Gradient Problem

1. RNN ⟶ Problem ? ✓
2. Why LSTM RNN ? ✓ Basic Representation.
3. How LSTM RNN Works
   - Long Term Memory ✓
   - Short Term Memory ✓
4. LSTM Architecture
5. Working of LSTM RNN

Problems With RNN → Long Term Dependency



An unrolled recurrent neural network.

For small sentences simple RNN will just work fine

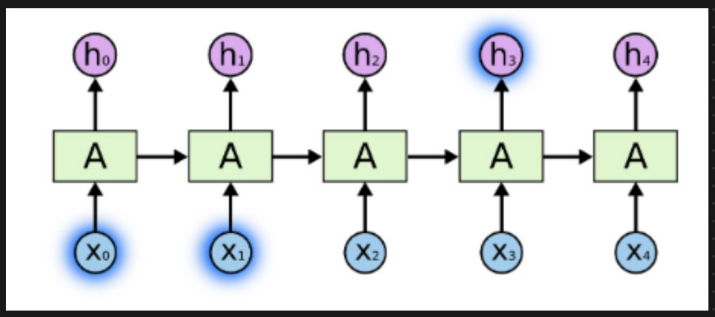
Vanishing Gradient Problem

Task:

Next Word In a Sentence

The color of the sky is blue

further context



Gap is less

Huge gap O/p ← Context

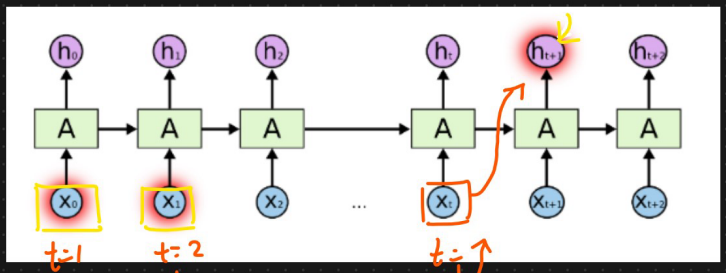
I grew up in India .... I speak fluent ____

↳ Context

Name of language

↓

further Context



Huge Gap → Long Term Dependency

0-0.25

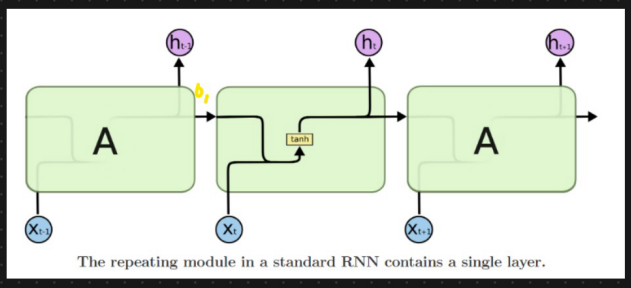
0-1

RNN → Long Term Dependency → Vanishing Gradient Problem

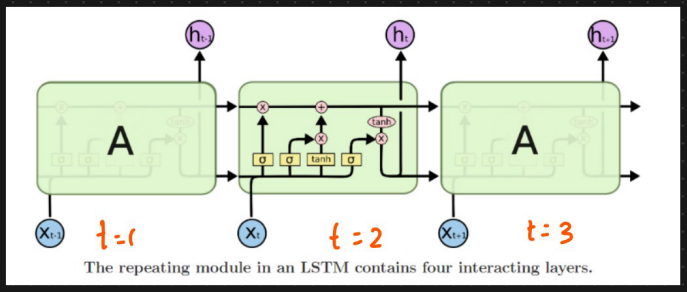
↳ Chain Rule → ≈0.

Basic Representation of RNN And LSTM RNN

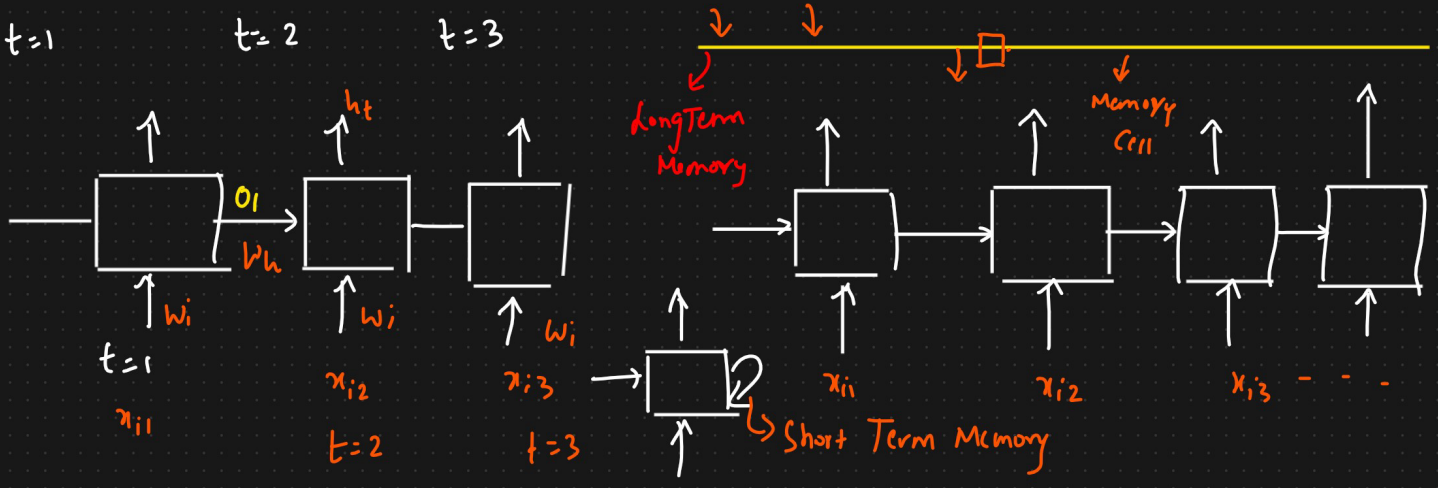
LSTM RNN



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



The repeating module in an LSTM contains four interacting layers.



LSTM RNN → Long Term Memory (Memory Cell with Gates)

→ Short Term Memory (Hidden State)

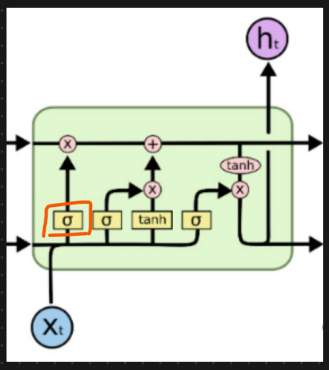
Conveyance But : Luggages

Imp. Why is LSTM ; Long- Short Term ?

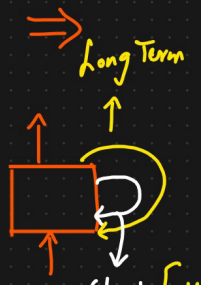
Yes, traditional RNNs function as "short-term memory" models—they maintain a single hidden state that passes through each timestep, capturing recent information. However, they struggle to retain information over long sequences due to the **vanishing gradient problem**, where signals fade as they propagate backward during training Medium Wikipedia .

LSTMs, on the other hand, extend this capability by introducing a **memory cell** along with gating mechanisms *(input, forget, and output gates)* that control what to keep, forget, or pass on. These gates enable the model to **preserve and manage long-term dependencies** more effectively, earning the name "Long Short-Term Memory" Wikipedia GeeksforGeeks Medium .

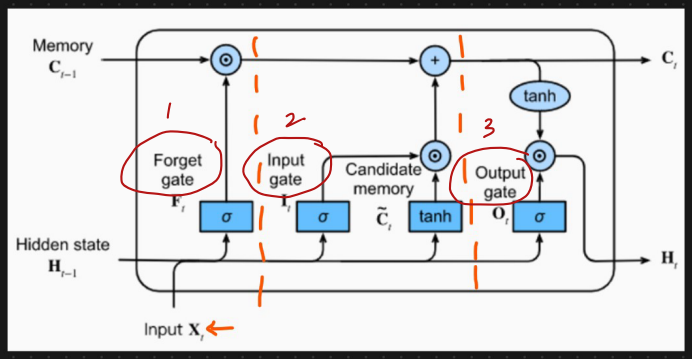
LSTM Architecture



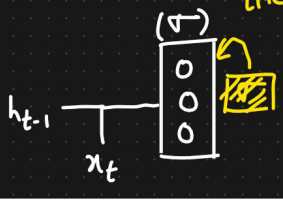
Basic Architecture

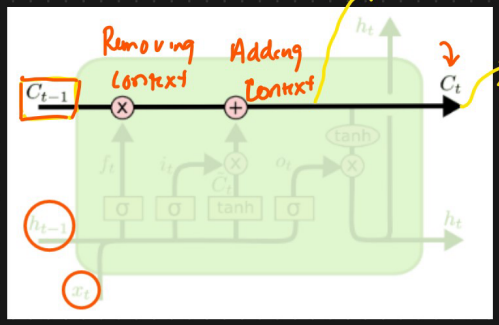


LSTM RNN

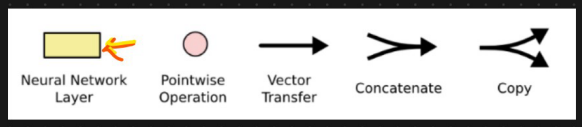


{Neural Nw Layer}





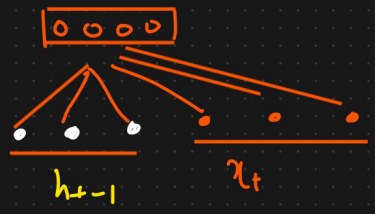
Long Term Memory



Combining 2 vectors

$h_{t-1} = [1 \quad 2 \quad 3]$

$x_t = [2 \quad 3 \quad 4]$



Duplicate copy

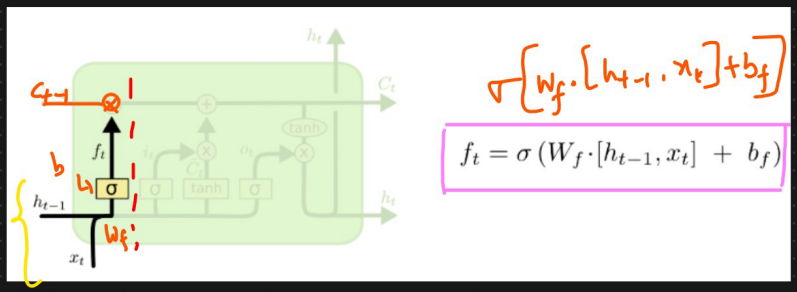
V1 V2

← tanh

⊗ ⇒ [4 10 18]

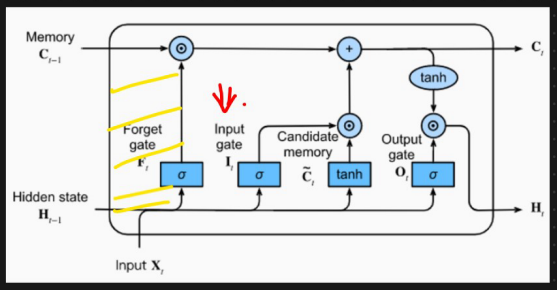
⊕ ⇒ [5 7 9]

tanh ⇒ [tanh(1) tanh(2) tanh(3)]



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

$$\sigma\left[W_f \cdot [h_{t-1}, x_t] + b_f\right]$$



Forget Gate

h_{t-1} = Hidden state of previous time stamp

x_t = Word passed as i/p in the current time stamp

Text

x_{11} x_{12} x_{13} x_{14}

Next Word

y_{15}

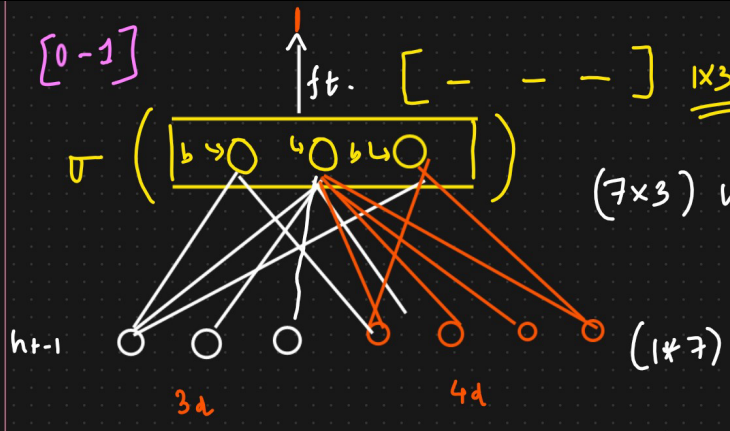
$[0 \quad 2 \quad 4 \quad 1] \quad [4 \quad 5 \quad 12] \quad - - - - -$

x_t x_{t+1}

Memory Cell

h_{t-1} [1 2 4] 3d

C_{t-1} [4 2 1] 3d ⇐



$$f_t = [0 \;\; 0 \;\; 0]$$ ← Complete Sentence

Context changed

① $C_{t-1} = [6 \;\; 8 \;\; 9]$ ⊗ $[0 \;\; 0 \;\; 0]$

$= [0 \;\; 0 \;\; 0]$ ← Removing all the previous Context

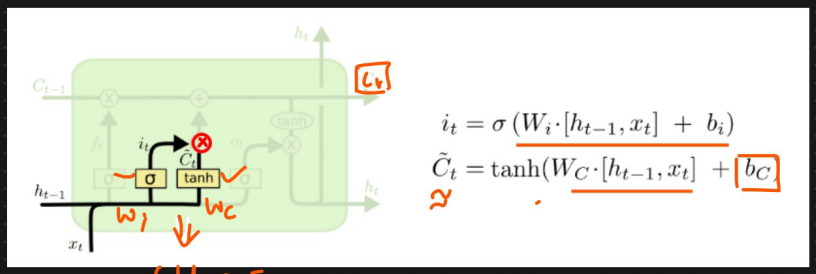
$f_t = [1 \;\; 1 \;\; 1]$

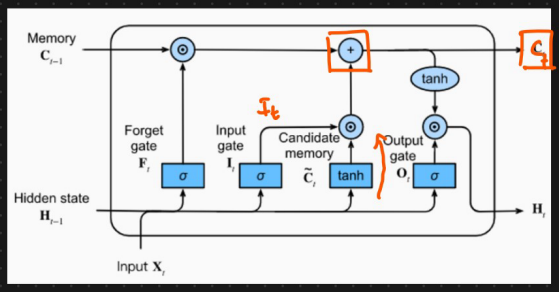
② $C_{t-1} = [6 \;\; 8 \;\; 9]$ ⊗ $[1 \;\; 1 \;\; 1] = [6 \;\; 8 \;\; 9]$

③ $C_{t-1} = [6 \;\; 8 \;\; 9]$ ⊗ $[0.5, 1, 0.5] = [3 \;\; 8 \;\; 4.5]$

Conclusion : Based on the context → Forget gate will let go some information or will not let go some Info {Forgetting}.

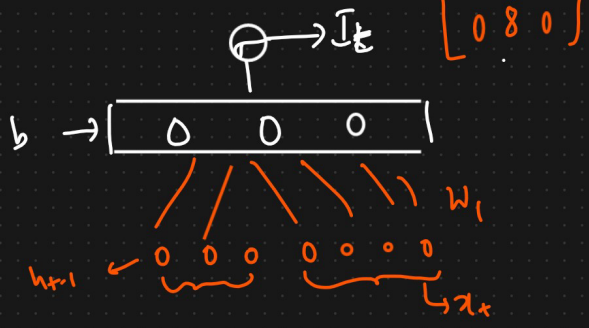
② Input Gate And Candidate Memory

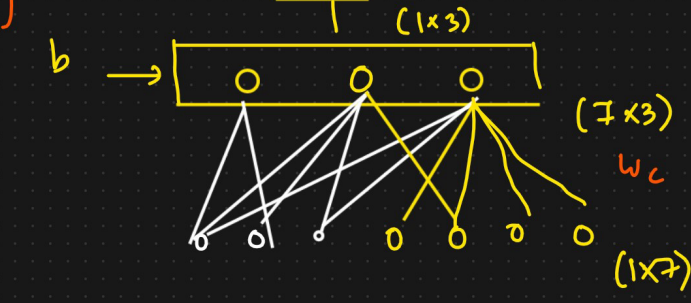




$I_t = [\ 2\ \ 4\ \ 6\ ]$ ——— (*) ——— C_t $[\ 0\ \ 2\ \ 0\ ]$ ⇒ I/p Gate

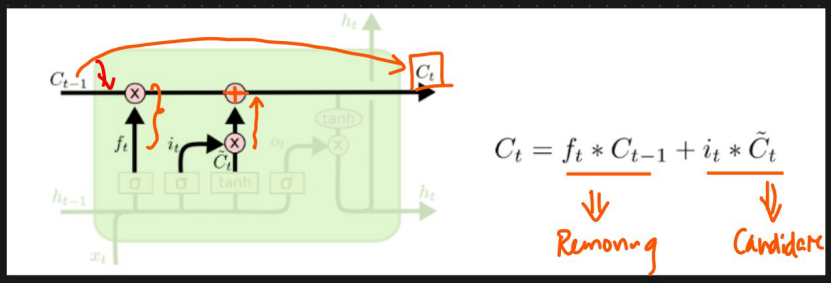
$[\ 0\ \ 8\ \ 0\ ]$

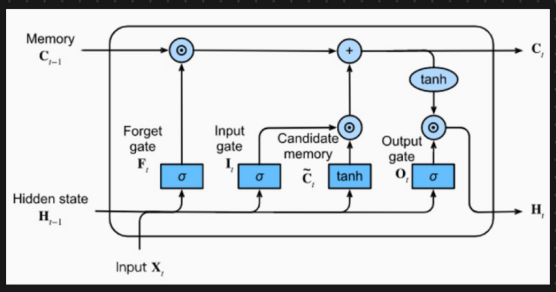




Context = If any information needed to be added it the memory

C_{t-1} ⟶ The information will be added





I stay in India - - - - - -

and I speak → English → Hindi

Or

Forgetting

Some info

Memory.

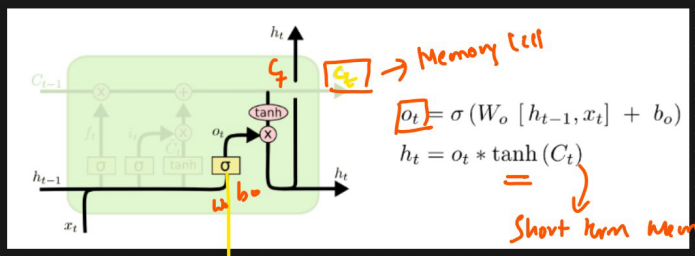
Forget Gate

I/p Gate ⊛ Candidate memory

+

Ct-1 ⇒ Cf.

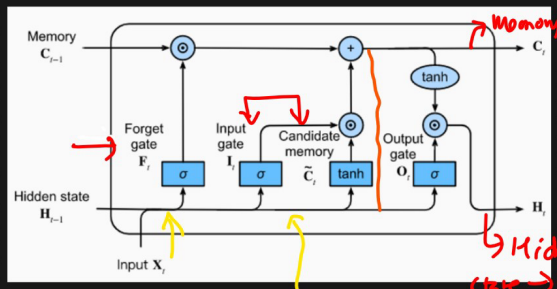
Output gate LSTM RNN



→ Memory Cell

→ Short term Memory.

→ Neural Network

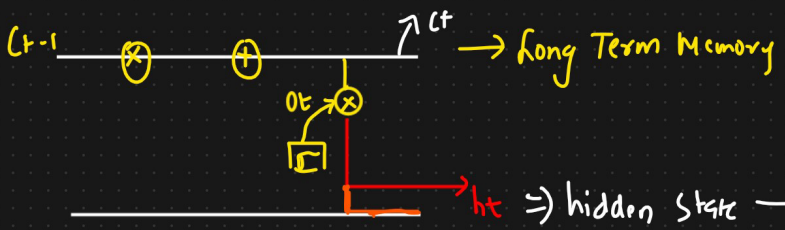


Memory

→ Hidden state → ST Memory.

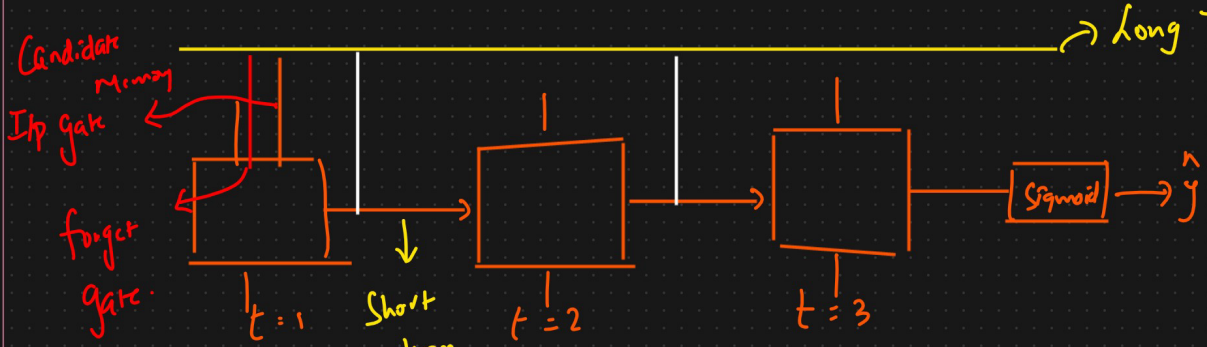
Forget — Add Info → Context

Some Info → context → Memory Cell



→ Long Term Memory

ht ⇒ hidden state → Short term Memory



→ Long Term

loss ↓↓.

[Wi, Wc, Wo] → Updating ← Back Propogation

GRU RNN ⇒ LSTM Variant

Training Data With LSTM RNN

{Training} Text Paragraph

I Went to Restaurant and order burger

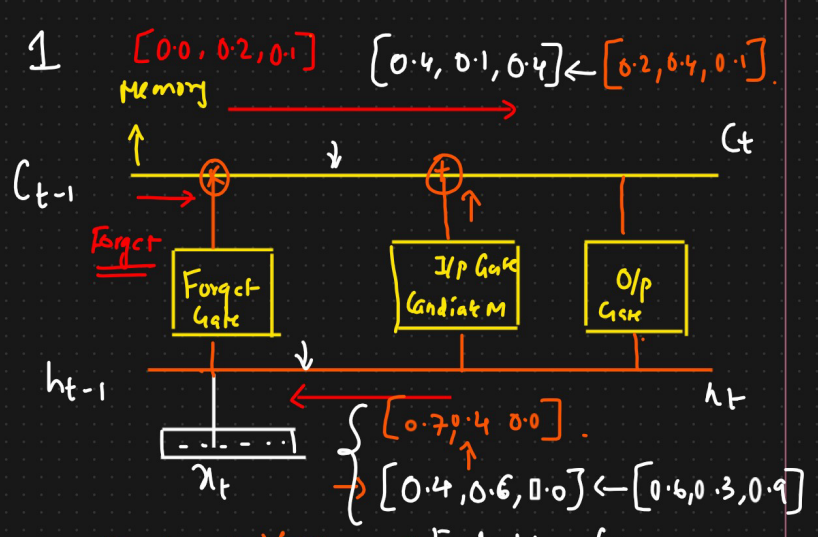
The burger looked tasty and crispy

→ But burger is not good for health }

→ It has lot of fats, cholestrol

→ But this burger was made with whey protein and only vegetables were used, so it was good.

O/p [good/bad]



Word → Vectors → Embedding Layer

Word2Vec [3 dimension-vector]

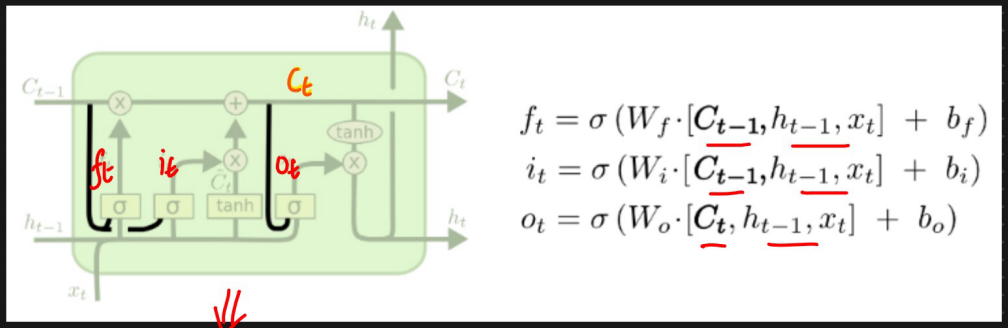
→ [[Good] [Bad] [Healthy]] ← Black Box

Tasty [0.9 0.0 0.1] ← 3 d.

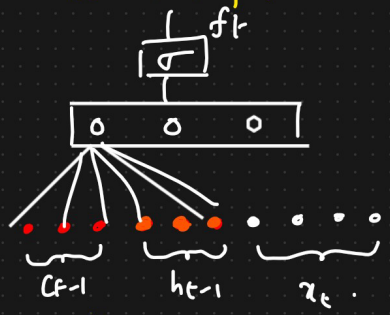
Variants of LSTM RNN

LSTM Variants Introduced By Gers & Schmidhuber [2000] → Research paper

LSTM RNN [1970-80]



$$f_t = \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [C_t, h_{t-1}, x_t] + b_o\right)$$

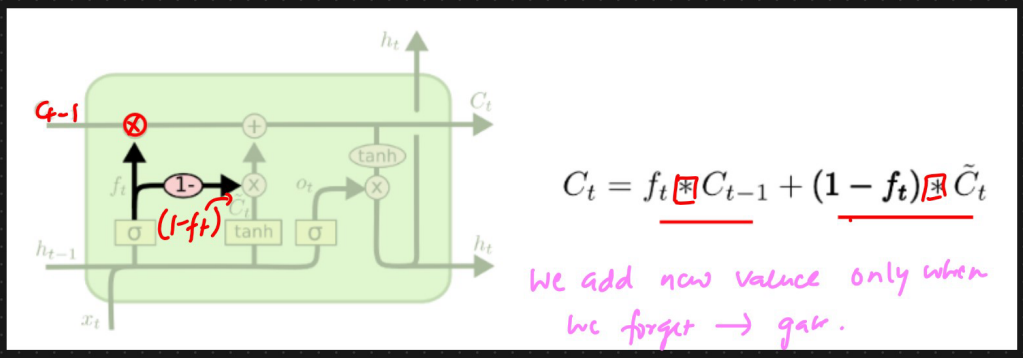


focus

Connections → From Memory cell to forget gate, i/p gate, O/p gate ⇒ Peephole Connections

Peephole Connections : We let the gate layers look at the cell state

Another variation → Coupling Forget And I/p Gates



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

We add new values only when we forget → gate.

Goal : We only forget when we're going to i/p something in its place.

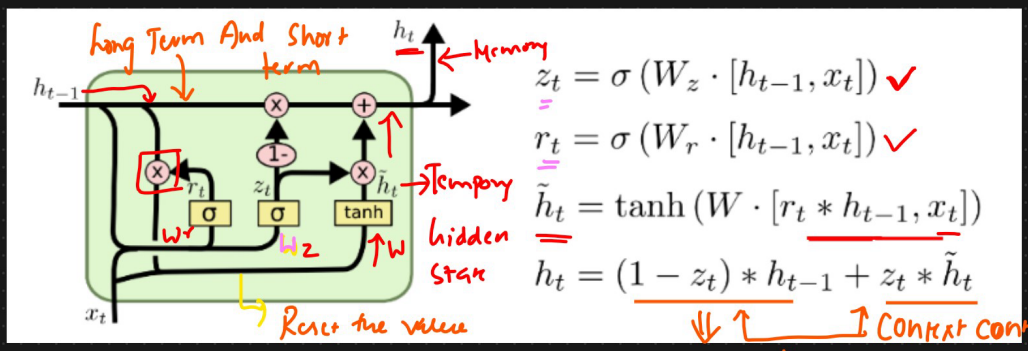
⇓

We only i/p new values to the state when we forget something older.

Instead of seperately deciding what to forget and what we should add new Info, we make this decision together.

GRU → Gated Recurrent Unit [Cho, et al [2014]]

1980 → LSTM
2000 - Variants
2014 → GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Context context

⇓

Coupled.

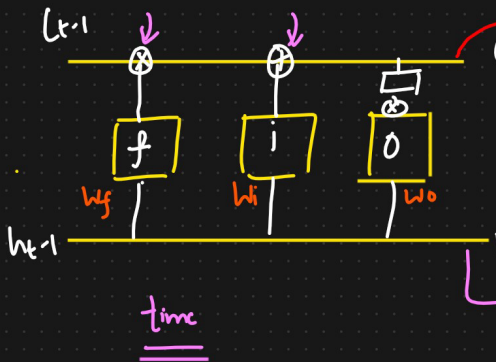
z_t ⇒ Update Gate ←

r_t ⇒ Reset Gate ←

$\tilde{h}_t$ ⇒ Temporary hidden state.

Trainable parameters

bias → $[W_f, W_i, W_o]$

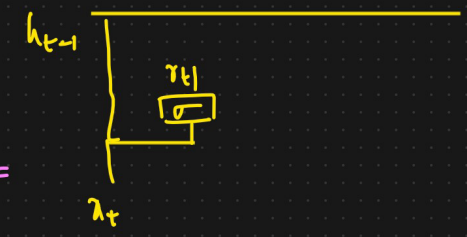


LSTM : Complex Architecture → forget, i/p + Candidate memory, O/p

Training Time ↑↑

z_t ,

Reset gate = r_t =

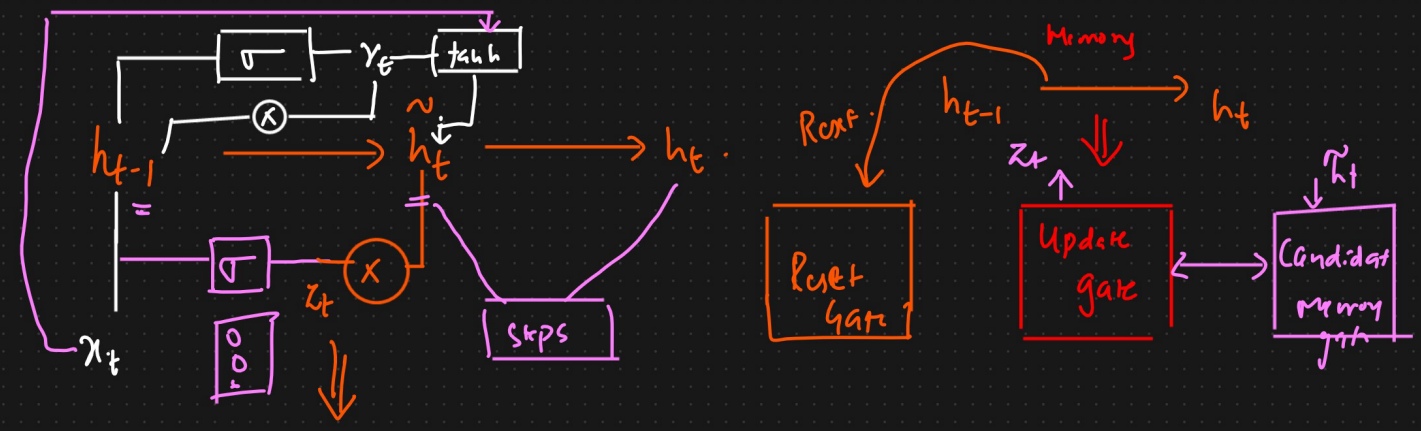


→ Reseting some info from h_{t-1} ⇒ Memory → LTM + STM

h_{t-1} [0.6 0.5 0.3 0.9]

$\otimes \leftarrow r_t$ [0.2 0.4 0.8 0.2]

$x_t \rightarrow$ [0.12 0.20 0.24 0.18] ← Resetting → Context



What context Info needs to be Added

⇓

Candidate hidden state · [Current Context]

⇓

Imp → Add Info

$z_t \Rightarrow$ Less Info