LSTM - Long Short-Term Memory
Understanding Through a Complete Example

THE PROBLEM

THE SOLUTION

RNNs Cannot Remember

- Vanishing gradients
- $0.5^{50} \approx 0$ (information dies)
- Forgets early context
- Cannot handle long sentences

Example:

"I grew up in Paris. I speak fluent ___"

RNN forgets "Paris" after 20 words
Cannot predict "French"

LSTM Controls Memory

- Three gates (0-1 values)
- Addition path (not multiplication)
- Preserves gradients
- Remembers 100+ steps

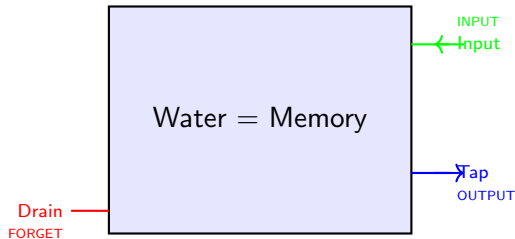**The Method:**

Three Independent Gates:

- FORGET: What to erase
- INPUT: What to add
- OUTPUT: What to use

Let's see how this works with a simple analogy...

Think of Memory as a Water Tank with Three Valves

The Tank System:

INPUT
← Input

Water = Memory

Tap
OUTPUT

Drain
FORGET

How Each Valve Works:

FORGET = Drain Valve
Controls how much water flows OUT
0.1 = Open 10% → 90% drains away
Removes old water (old memory)

INPUT = Input Valve
Controls how much new water flows IN
0.9 = Open 90% → lots added
Adds fresh water (new memory)

OUTPUT = Output Tap

Real Examples:

At period "." in sentence:
- Drain: 90% (0.1 forget)
- Input: 40% (0.4 input)
- Tap: 30% (0.3 output)

→ Tank mostly empties!

At noun "dog":
- Drain: 30% (0.7 forget)
- Input: 90% (0.9 input)
- Tap: 90% (0.9 output)

→ Tank fills up!

Intuition: The Key Insight

Three INDEPENDENT valves on ONE tank!

Each valve controls a different aspect:
- Drain: How much OLD to remove
- Input: How much NEW to add
- Tap: How much to USE now

This is EXACTLY what LSTM does!

Three Gates Control Memory Like Volume Knobs (0 to 1)

FORGET

$\times$

REMOVES
old information

Value 0-1:
- 0.0 = erase all
- 0.5 = keep half
- 1.0 = keep all

Example:
0.1 at period
$\rightarrow$ Erase 90%!

INPUT

$+$

ADDS
new information

Value 0-1:
- 0.0 = add nothing
- 0.5 = add half
- 1.0 = add all

Example:
0.9 on "cat"
$\rightarrow$ Store lots!

OUTPUT

$\rightarrow$

REVEALS
stored information

Value 0-1:
- 0.0 = hide all
- 0.5 = show half
- 1.0 = show all

Example:
0.9 at "was"
$\rightarrow$ Use memory!

How They Work Together:
New Memory = (Forget $\times$ Old Memory) + (Input $\times$ New Info)
Output = Output Gate $\times$ Memory

Now let's see these gates in action with concrete numbers...

Updating Memory From "cat" to "dog"

Starting Point:

Old Memory: [0.8, 0.6, 0.4]

Contains "cat" information

Step 1: FORGET Gate = 0.1
What it does: Multiply old memory by 0.1
[0.8, 0.6, 0.4] × 0.1 = [0.08, 0.06, 0.04]
Result: 90% erased! "cat" mostly removed.
Why? At period, we need fresh start for new sentence.

Step 2: INPUT Gate = 0.9
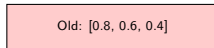**What it does:** Filter new candidate info
Create candidate: [0.7, 0.5, 0.9]
Multiply by 0.9: [0.7, 0.5, 0.9] × 0.9
= [0.63, 0.45, 0.81]
Result: 90% of "dog" info ready to add.
Why? "dog" is new subject, very important!

Visual Flow (Steps 1-2):
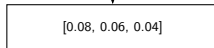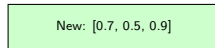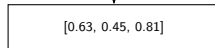
Old: [0.8, 0.6, 0.4]	New: [0.7, 0.5, 0.9]
× 0.1 (FORGET)	× 0.9 (INPUT)
[0.08, 0.06, 0.04]	[0.63, 0.45, 0.81]
Mostly empty (90% erased)	Ready to add (90% of "dog")

Checkpoint: Key Point

Two INDEPENDENT operations:
- FORGET decides what OLD to keep
- INPUT decides what NEW to add
- They don't interfere!

From Separate Results to Final Output

Where We Left Off:

- Erased old: [0.08, 0.06, 0.04]
- Filtered new: [0.63, 0.45, 0.81]

Step 3: COMBINE (Addition!)
What it does: Add the two results together
[0.08, 0.06, 0.04] + [0.63, 0.45, 0.81]
= [0.71, 0.51, 0.85]
Result: Updated memory = "dog" info
Why addition? Preserves gradients! This is the key innovation that prevents vanishing.

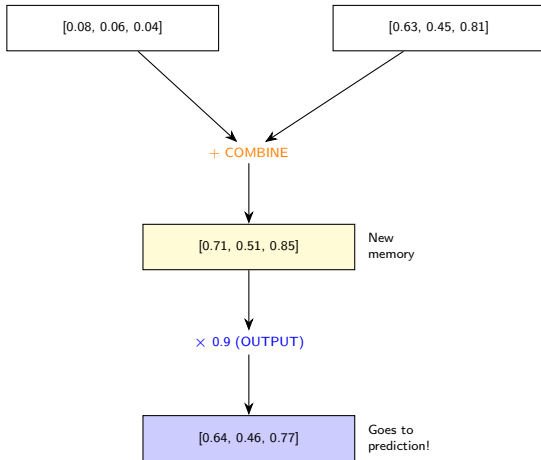**Step 4: OUTPUT Gate = 0.9**
What it does: Filter what network sees
[0.71, 0.51, 0.85] × 0.9
= [0.64, 0.46, 0.77]
Result: 90% revealed to next layer
Why? At "was", we NEED subject info for verb prediction!

Visual Flow (Steps 3-4):

From Hidden State to Prediction

Recap: Output Gate Result

From Step 4: [0.64, 0.46, 0.77]

This is the **hidden state** h_t

Where Does It Go?

1. **To Prediction Layer**
   $h_t \rightarrow$ Linear $\rightarrow$ Softmax $\rightarrow$ Probabilities
   Example at "was": Predict next word

2. **To Next Time Step**
   h_t feeds into next LSTM cell
   Used to compute next gates

3. **Optional: To Attention**
   In seq2seq models, decoder attends to these h_t values

Key Distinction:

- C_t = Long-term memory (protected)
- h_t = Working memory (filtered output)

Now let's see why we need three separate gates...

Concrete Example:

At word "was" in "The dog was sleeping":
Memory C_t contains: [dog, context]
Output gate: 0.9 (reveal 90%)
Hidden state h_t: [0.64, 0.46, 0.77]
Prediction layer receives h_t:
$h_t \rightarrow$ Linear(512 $\rightarrow$ vocab) $\rightarrow$ Softmax
Top predictions:

- "sleeping": 0.35 (verb matches dog)
- "running": 0.18
- "eating": 0.12

The subject info ("dog") in h_t helps predict appropriate verb!

Intuition: Why Filter?

Not all memory is relevant NOW.
Examples:

- At "The": Output gate low (0.2) $\rightarrow$ hide memory, article doesn't need context
- At "was": Output gate high (0.9) $\rightarrow$ reveal memory, verb needs subject!

The OUTPUT gate is smart about WHEN to use memory.

Why Three Separate Gates? Real Scenarios

Reading: "The cat sat. The dog..."

Scenario 1: At "cat"

Gate Values:
- F = 0.8 (keep)
- I = 0.9 (STORE!)
- O = 0.8 (show)

What Happens:
- Keep previous context
- STORE subject strongly
- Show it to network

Goal:
Remember "cat" for rest of sentence

Memory:
→ [cat, context]

Scenario 2: At "."

Gate Values:
- F = 0.1 (ERASE!)
- I = 0.4 (small)
- O = 0.3 (HIDE)

What Happens:
- ERASE old sentence
- Small punctuation add
- HIDE memory

Goal:
Clean slate for new sentence

Memory:
→ [mostly empty]

Scenario 3: At "dog"

Gate Values:
- F = 0.7 (keep some)
- I = 0.9 (NEW!)
- O = 0.9 (REVEAL!)

What Happens:
- Keep some context
- STORE new subject
- REVEAL all info

Goal:
New focus, need it NOW

Memory:
→ [dog, some context]

Key Insight: Each situation needs DIFFERENT gate values!
That's why LSTM has three independent gates, not just one.
The network LEARNS which values to use for each word.

Now let's watch these gates in action on a real sentence...

Sentence: "The cat was hungry. The dog was sleeping."

Word	Forget	Input	Output	Memory State
The	0.9	0.3	0.2	*article*
cat	0.8	**0.9**	0.8	*subject: cat*
was	0.9	0.7	0.9	*cat + verb*
hungry	0.8	0.8	0.7	*cat is hungry*
.	**0.1**	0.4	0.3	*sentence ends*
The	0.1	0.8	0.2	*new article*
dog	0.7	**0.9**	0.9	*subject: dog*
was	0.9	0.8	**0.9**	*using dog info*

0.1 = Forget   **0.9** = Store/Use   Period → Reset

Notice the patterns? Let's explore what you observed...
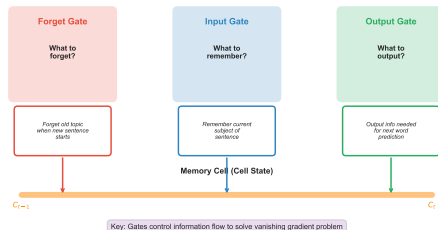
What Did You Notice?

Common Observations:

Students usually notice:

- **"It drops to 0.1 at the period!"**
- **"It's 0.9 on important words (cat, dog)"**
- "The memory changes from cat to dog"
- "It resets between sentences"
- "Three different columns of numbers"

Key Questions:

1. HOW does it know to forget at period?
2. HOW does it know cat and dog are important?
3. HOW does it decide when to use memory?

LSTM Solution: Three Smart Gates

Forget Gate	Input Gate	Output Gate
What to forget?	What to remember?	What to output?
Forget old topic when new sentence starts	Remember current subject of sentence	Output info needed for next word prediction

Memory Cell (Cell State)

c_{t-1} c_t

Key: Gates control information flow to solve vanishing gradient problem

Checkpoint: The Big Reveal

Those three columns are called **GATES**:

- Forget Gate: Controls what to erase
- Input Gate: Controls what to store
- Output Gate: Controls what to use

But WHY do we need gates?

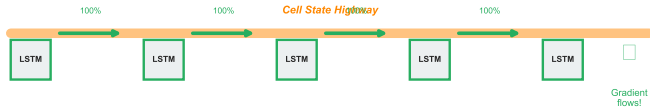**The Vanishing Gradient Problem**

Standard RNN:

RNN → 100% → RNN → 80% → RNN → 60% → RNN → 39% → RNN   ✗

Gradient vanishes!

LSTM:

100% — 100% — *Cell State Highway* 100% — 100%

LSTM — LSTM — LSTM — LSTM — LSTM   ☐

Gradient flows!

Key: LSTM uses addition (cell state) instead of multiplication (RNN hidden state)

RNN Problem:
- Gradients vanish ($0.5^{50} \approx 0$)
- Forgets early information
- Can't handle long dependencies
- Would lose "cat" by "dog"

LSTM Solution:
- Cell state highway (addition not multiplication)
- Three gates for CONTROL
- Can preserve info for 100+ steps
- Then ERASE when sentence ends

Forget Gate: What to Erase?

Example: "The cat was hungry. The dog ..."

Inputs:

h_{t-1}: Previous output

x_t: Current word ("dog")

Forget Gate

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

Output: 0 to 1

Decision:

"cat" info   **10%** *Forget! (new subject)*

"hungry" info   **20%** *Forget! (not relevant)*

Lower values (close to 0) = FORGET
Higher values (close to 1) = KEEP

Intuition: When you see "dog", forget information about "cat"

Back to Our Table - Row 5:

Word	Forget
"."	0.1

What This 0.1 Means:

- 0.0 = forget everything
- 1.0 = keep everything
- 0.1 = forget 90% (keep only 10%)

Why at period?

The Formula That Produces 0.1:

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

How It Decides:

1. Look at current word (".")
2. Look at previous hidden state
3. Compute weighted sum
4. Apply sigmoid $\rightarrow$ output 0 to 1
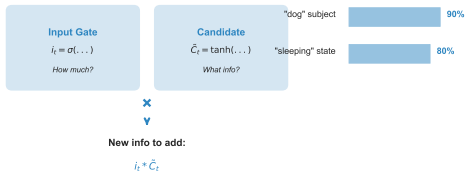
Cell State Update:

Input Gate: What to Remember?

Example: "The dog was sleeping ..."

Inputs:

h_{t-1}: Previous output

x_t: Current word ("sleeping")

Input Gate	Candidate
$i_t = \sigma(\dots)$	$\bar{C}_t = \tanh(\dots)$
How much?	*What info?*

"dog" subject

"sleeping" state

Decision:

90%

80%

×

∨

New info to add:

$i_t * \bar{C}_t$

Intuition: Remember "dog is sleeping" for future predictions

Back to Our Table - Row 7:

Word	Input
"dog"	**0.9**

What This 0.9 Means:

- $0.0 =$ add nothing
- $1.0 =$ add everything
- $0.9 =$ add 90% of candidate

Why at "dog"?

The Formulas (Two Parts):

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

$$\bar{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

How It Works:

1. Create candidate info ($\bar{C}_t$) with tanh
2. Decide how much to use ($i_t = 0.9$)
3. Multiply: $0.9 \times$ candidate
4. Add to cell state

Output Gate: What to Output?

Example: "The dog was sleeping and ..." → predict next word

Cell State:

Contains: dog, sleeping, etc.

Question: What's relevant NOW?

Output Gate

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

How much to output?

Decision:

"dog" info		**90%**	*Output! (subject)*
"sleeping" info		**70%**	*Output! (state)*
old context		**10%**	*Hide (not needed)*

Final Output:

$h_t = o_t * \tanh(C_t)$

▼

To next layer / prediction

Intuition: Only share relevant parts of memory for current prediction

Back to Our Table - Row 8:

Word	Output
"was"	0.9

What This 0.9 Means:

- 0.0 = hide everything
- 1.0 = reveal everything
- 0.9 = output 90% of memory

Why at "was"?

The Formulas:

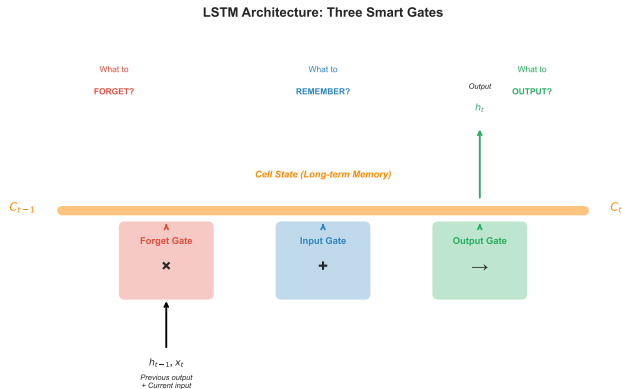$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

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

How It Works:

1. Look at cell state (has "dog" info)
2. Decide what's relevant NOW
3. Filter memory through gate (0.9)
4. Send h_t to prediction layer

Key Insight:

The Big Picture: Three Gates Working Together

LSTM Architecture: Three Smart Gates

The Cell State Highway:
- Protected memory channel
- Information flows easily
- Gates control entry/exit
- Gradients don't vanish!

At Each Time Step:
1. **Forget:** Erase old ($0.1 \rightarrow$ erase "cat")
2. **Input:** Add new ($0.9 \rightarrow$ add "dog")

Intuition: Visual Analogy

Think of LSTM like a notebook:
- **Forget Gate** = Eraser
  (Clear old notes at period)
- **Input Gate** = Pen

Now Look Again - You Understand EVERYTHING!

Sentence: "The cat was hungry. The dog was sleeping."

Word	Forget	Input	Output	What LSTM "Thinks"
The	0.9 (keep)	0.3 (weak)	0.2 (hide)	Article seen, nothing special yet
cat	0.8 (keep)	0.9 (STORE!)	0.8 (show)	Subject! Important noun!
was	0.9 (keep)	0.7 (add)	0.9 (need!)	Verb connects to cat
hungry	0.8 (keep)	0.8 (add)	0.7 (show)	Describes the cat's state
.	0.1 (ERASE!)	0.4 (end)	0.3 (hide)	Sentence over! Clear memory!
The	0.1 (clear)	0.8 (new!)	0.2 (hide)	NEW sentence starts fresh
dog	0.7 (keep)	0.9 (NEW!)	0.9 (use!)	NEW subject! (forgot cat)
was	0.9 (keep)	0.8 (add)	0.9 (USE!)	Using DOG info for prediction

Checkpoint: The Magic Transition

Watch rows 4→5→6→7: **hungry → . → The → dog**
Memory Evolution: [cat, hungry] → FORGET (0.1) → [end] → ADD (0.9) → [dog]
This intelligent memory control is what RNNs cannot do! LSTM uses gates to:

- Preserve important info (0.9 on subject nouns)
- Erase when context changes (0.1 at sentence boundaries)
- Reveal info when needed (0.9 output for predictions)

Summary: From Table to Understanding

Your Learning Journey:

1. **Intuition:** Water tank analogy
   (Three valves on one tank)
2. **Concepts:** What gates are
   (Volume knobs from 0 to 1)
3. **Mechanics:** How they work
   (4-step process with real numbers)
4. **Usage:** What happens to output
   (Hidden state $\rightarrow$ prediction layer)
5. **Mastery:** Complete understanding
   (Table makes perfect sense now!)

Key Equations:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$
$$h_t = o_t \odot \tanh(C_t)$$

Real World: Where LSTMs Excel

Applications (2015-2020):
- Machine Translation (Google Translate)
- Speech Recognition (Siri, Alexa)
- Text Generation (early GPT)
- Video Analysis
- Music Generation
- Handwriting Recognition

Modern Context (2024):
Transformers now dominate NLP, but LSTMs:
- Still used in time series
- Efficient for streaming data
- Foundation for understanding attention

The Core Insight:
That table showed you *exactly* how gates work. Every 0.1 and 0.9 has a purpose. That's the real magic of LSTMs!

Questions?