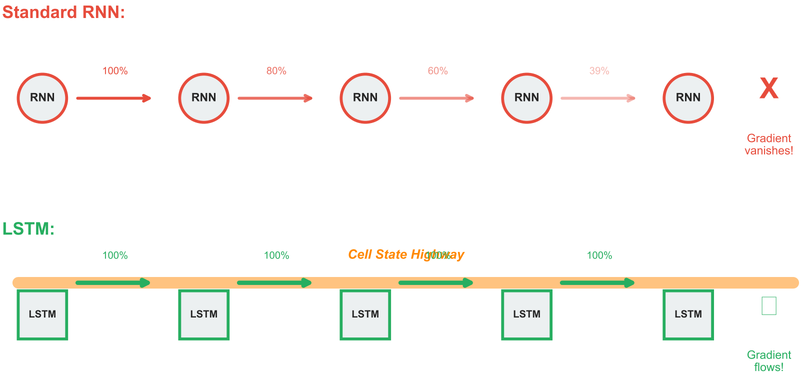
LSTM - Long Short-Term Memory

Solving the Vanishing Gradient Problem

The Problem: Vanishing Gradients

The Vanishing Gradient Problem



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

Example: Why We Need LSTM

The Challenge:

Sentence:
"The **cat** that chased the mouse that ate the cheese that sat on the table **was** hungry."

Problem: Predict "was" (singular) based on "cat" (15 words earlier)

RNN Result: Forgets "cat", might predict "were"
LSTM Result: Remembers "cat", correctly predicts "was"

Intuition: The Memory Gap

Standard RNN:

- Memory fades after 5-10 words
- Cannot handle long dependencies
- Gradient vanishes over time

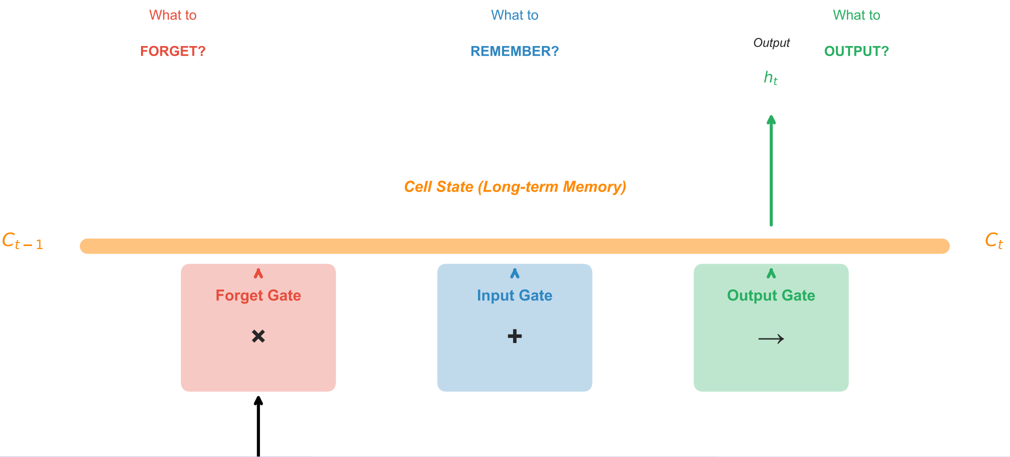
LSTM:

- Memory persists 100+ words
- Handles long dependencies
- Gradient flows through cell state

Real Impact:

- Better grammar understanding
- Improved translation quality
- More coherent text generation

LSTM Architecture: Three Smart Gates



Example: LSTM as a Smart Notebook

Imagine taking notes in class:

1. **Forget Gate (Eraser):**
   - Professor changes topic
   - You cross out old notes
   - Make room for new information
2. **Input Gate (Pen):**
   - Write down important points
   - Decide what's worth noting
   - Add to existing notes
3. **Output Gate (Highlighter):**
   - Highlight relevant parts
   - Focus on what's needed now
   - Share selected information

Real World: Notebook = Cell State

Cell State (C_t): Your notebook

- Stores all important information
- Persists across time steps
- Protected by gates
- Information highway

Key Difference from RNN:
RNN: Rewrites entire memory each time
LSTM: Selectively updates memory

- Keeps what's important
- Erases what's not
- Adds new information

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"

Mathematical Formula:

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

How It Works:

1. Look at current input and previous output

Input Gate: What to Remember?

Example: "The dog was sleeping ..."

Inputs:

h_{t-1}: Previous output

x_t: Current word ("sleeping")

Input Gate

$i_t = \sigma(\ldots)$

How much?

Candidate

$\tilde{C}_t = \tanh(\ldots)$

What info?

Decision:

×

New info to add:

$i_t * \tilde{C}_t$

Intuition: Remember "dog is sleeping" for future predictions

Mathematical Formulas:

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

How It Works:

1. Create candidate values (tanh gives -1 to 1)

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

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

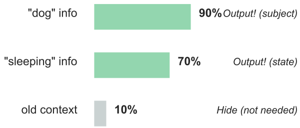
How much to output?

Final Output:

$h_t = o_t * \tanh(C_t)$

To next layer / prediction

Decision:



Intuition: Only share relevant parts of memory for current prediction

Mathematical Formulas:

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

How It Works:

1. Compute output decision (o_t)

Example: Processing a Sentence

Sentence: “The cat was hungry. The dog was sleeping.”

Word	Forget Gate	Input Gate	Output Gate	Cell State
“The”	Keep all (0.9)	Add article (0.3)	Output little (0.2)	[article]
“cat”	Keep article (0.8)	Add subject (0.9)	Output subject (0.8)	[cat, article]
“was”	Keep subject (0.9)	Add verb (0.7)	Output for pred. (0.9)	[cat, was]
“hungry”	Keep subject (0.8)	Add state (0.8)	Output state (0.7)	[cat, hungry]
“.”	Reduce old (0.3)	New sentence (0.4)	Low output (0.3)	[sentence end]
“The”	Clear old (0.1)	New article (0.8)	Low output (0.2)	[article]
“dog”	Keep article (0.7)	New subject (0.9)	Output subject (0.9)	[dog, article]
“was”	Keep subject (0.9)	Add verb (0.8)	Output for pred. (0.9)	[dog, was]

Key Observations:

- Forget gate clears old subject at period
- Input gate adds new subject (“dog”)
- Output gate adjusts based on task
- Cell state maintains context

Checkpoint: Understanding

Q: Why forget “cat” when seeing “.”?
A: New sentence starting, old subject no longer relevant for predictions

Summary: LSTM in Practice

When to Use LSTM:

1. Long sequences (100+ words)
2. Long-term dependencies needed
3. Sequential data with context
4. Grammar and structure matter

LSTM vs RNN:

	RNN	LSTM
Memory span	5-10 steps	100+ steps
Parameters	Fewer	More
Training time	Faster	Slower
Accuracy	Lower	Higher
Best for	Short text	Long text

Real World: Applications

Where LSTMs Excel:

- **Machine Translation:** Google Translate
- **Speech Recognition:** Siri, Alexa
- **Text Generation:** Story writing
- **Video Analysis:** Action recognition
- **Music Generation:** Composition
- **Handwriting Recognition:** OCR systems

Key Takeaways:

- 3 gates control information flow
- Cell state is the memory highway
- Solves vanishing gradient problem
- Essential for modern NLP

Questions?