

LSTM Networks: Teaching Machines Long-Term Memory

BSc Enhanced Version with Full Formula Explanations

NLP Course 2025

September 27, 2025

Page 1: The Autocomplete Challenge

The Problem: Predicting What Comes Next

Your Phone Predicts the Next Word

"I love chocolate ice cream but I prefer..."



How does it know?
It remembered "chocolate" context!

Technical Terms Explained:

What You Type:

"I grew up in Paris. I went to school there for 12 years. I

Sequence Modeling: Predicting the next element based on all previous elements in a sequence.

Page 2: Why Simple Approaches Fail

N-gram Models: The Baseline (And Why They Don't Work)

N-Gram: Fixed 2-Word Window (Forgets "cat")



LSTM: Selective Memory (Remembers "cat")!



Notation Explained:

What N-grams Do:

- Count word sequences in training data
- Look at last 1-2 words only
- Pick most common next word

N-gram: A sequence of N words. Bigram = 2 words ("I love"), Trigram = 3 words ("I love chocolate")

$P(w_t \mid w_{t-1})$: Probabil-

Page 3: The Memory Problem - What We Need

Insight from Human Reading

Human Memory Example:

Chapter 1: "Alice was born in London in 1985. She had a happy childhood."

Chapter 3: "After graduating from university, Alice moved to New York."

Chapter 7: "Now 38 years old, Alice reflected on her life in ---"

What You Remember:

- Alice (main character) [YES]
- Born in London [YES]
- Moved to New York [YES]
- Currently 38 [YES]

What You Forgot:

- "had a happy childhood" [NO]
- "graduating from university" [NO]
- Exact wording [NO]

Three Mechanisms We Need:

1. Forget Gate: Decide what to remove from memory
Example: Forget "chocolate" after period

2. Input Gate: Decide what to store
Example: Store "Paris" strongly

3. Output Gate: Decide what to use now
Example: Recall "Paris" when predicting language

Technical Terms:

Gate: A learned decision mechanism that outputs values between 0 (block) and 1 (allow). Acts like a controllable valve

Checkpoint 1: Do You Understand the Problem?

Quick Self-Test Before Moving Forward

Question 1: Why can't N-grams solve the Paris problem?

- A) They're too slow
- B) They can only see 1-2 words back
- C) They require too much memory
- D) They don't understand French

Question 2: What are the three memory mechanisms we need?

- A) Read, Write, Delete
- B) Store, Retrieve, Process
- C) Forget, Input, Output
- D) Encode, Decode, Transform

Question 3: How far back do we need to remember?

Answers & Explanations:

Answer 1: B

N-grams use a fixed window of 1-2 words. In "I grew up in Paris... speak fluent ___", Paris is 18 words back - completely invisible to trigrams!

Answer 2: C

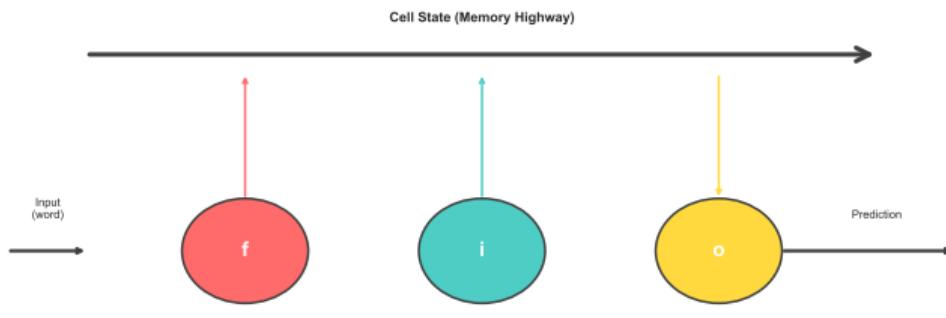
We need: **Forget** (remove old info), **Input** (add new info), **Output** (reveal info when needed). Just like human selective memory!

Answer 3: C

Real sentences can have important information 50-100 words back. LSTMs can handle this, but N-grams (1,2) and RNNs (5,10) both

Long Short-Term Memory: Gated Memory Cells

LSTM Cell: Three Gates Control Memory



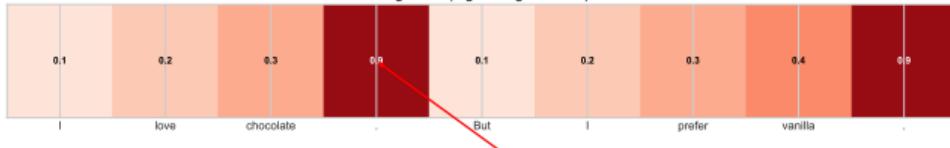
Like Traffic Lights: Red (forget) • Green (input) • Yellow (output)

Page 5: The Three Gates - How They Work

Gate Mechanisms with Concrete Examples

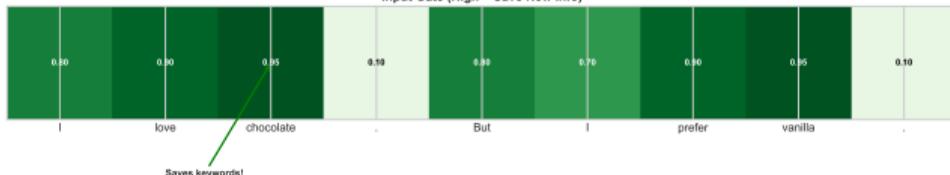
LSTM Gate Activations: "I love chocolate. But I prefer vanilla."

Forget Gate (High = Forget Old Info)



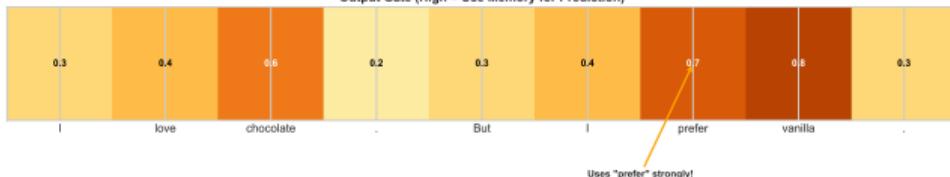
Forgets after period!

Input Gate (High = Save New Info)



Saves keywords!

Output Gate (High = Use Memory for Prediction)



Forget Gate f_t :

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

What This Chart Shows:

- Real gate values over sentence

Understanding the Math: Visual Explanations

What Do These Symbols Actually DO?

1. Sigmoid Function σ :

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

In plain English: Takes ANY number and squashes it to between 0 and 1. Used for gates because 0 = fully close, 1 = fully open

Visual Examples:

- Input: $z = -5 \rightarrow \sigma(-5) = 0.007$ 0 (CLOSE gate)
- Input: $z = 0 \rightarrow \sigma(0) = 0.5$ (HALF open)
- Input: $z = +5 \rightarrow \sigma(+5) = 0.993$ 1 (OPEN gate)

2. Tanh Function:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

In plain English: Like sigmoid but outputs -1 to +1. Used for cell state because memory can be positive or negative

3. Element-wise Multiplication \odot :

In plain English: Multiply each number separately, not matrix multiplication!

Example with actual numbers:

Gate values: $f_t = [0.9, 0.5, 0.1]$
Old memory: $C_{t-1} = [0.8, 0.6, 0.4]$
Result: $f_t \odot C_{t-1} = [0.9 \times 0.8, 0.5 \times 0.6, 0.1 \times 0.4]$
 $= [0.72, 0.30, 0.04]$
Each position multiplied independently!

4. Concatenation $[a, b]$:

In plain English: Stick two vectors together end-to-end

Example:

Previous state: $h_{t-1} = [0.5, 0.3]$ (size 2)
Current word: $x_t = [0.7]$ (size 1)
Concatenated: $[h_{t-1}, x_t] =$

Page 6: Cell State - The Memory Highway

Why LSTMs Can Remember 50-100+ Steps

RNN: Vanishing Gradient

LSTM: Gradient Highway

Cell State Highway



Gradient shrinks exponentially: $0.9^{10} \approx 0.35$

Gradient preserved: $1.0^{10} = 1.0$

The Update Equation:

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

In plain English: New memory = (keep some old) + (add some new). The \odot means multiply each number separately

Why Addition is Magic:

- Old C_{t-1} directly adds to new C_t

Comparison - RNN vs LSTM:

RNN (Multiplicative):

$$h_t = \tanh(W_h h_{t-1} + \dots)$$

After 50 steps:

- Signal: $0.5^{50} \approx 10^{-15}$
- Information lost!

LSTM (Additive):

Checkpoint 2: Do You Understand Cell State?

Test Your Understanding of the Memory Highway

Question 1: Why is addition better than multiplication for memory?

- A) It's faster to compute
- B) It preserves gradients across many steps
- C) It uses less memory
- D) It's easier to learn

Question 2: What does the cell state equation $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ do?

- A) Forgets everything and starts fresh
- B) Keeps some old + adds some new
- C) Only stores new information
- D) Copies the previous state exactly

Question 3: After 50 steps

NLP Course 2025

Answers & Explanations:

Answer 1: B

Addition creates a direct path for gradients! In RNNs, gradients multiply through many matrices and vanish ($0.5^{50} \approx 10^{-15}$). With addition, gradients flow directly backward unchanged.

Answer 2: B

First term ($f_t \odot C_{t-1}$) keeps SOME of the old memory (controlled by forget gate). Second term ($i_t \odot \tilde{C}_t$) adds SOME new information (controlled by input gate). Perfect blend!

Answer 3: B (0.08)

$0.95^{50} = 0.077\% \text{ Still usable}$

The Full Forward Pass (One Time Step)

All Six Equations:

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

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

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

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

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

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

Forget gate

Input gate

Candidate

Cell update

Output gate

Hidden state

Activation Functions:

Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$

- Range: (0, 1)
- Used for gates (0 = close, 1 = open)

Tanh: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Concrete Numerical Example:

Input: word “love” (after “I”)

Step 1: Compute gates

- $f_t = [0.62, 0.45, 0.69, \dots]$ (keep some)
- $i_t = [0.77, 0.69, 0.38, \dots]$ (add much)
- $o_t = [0.71, 0.75, 0.45, \dots]$ (reveal most)

Step 2: Create candidate

- $\tilde{C}_t = [0.54, -0.29, 0.72, \dots]$

Step 3: Update cell state

- Old: $C_{t-1} = [0.5, 0.3, 0.2, \dots]$
- Keep: $f_t \odot C_{t-1} = [0.31, 0.14, 0.14, \dots]$
- Add: $i_t \odot \tilde{C}_t = [0.42, -0.20, 0.27, \dots]$
- New: $C_t = [0.73, -0.06, 0.41, \dots]$

Step 4: Compute output

Checkpoint 3: Can You Read LSTM Equations?

Final Check: Do You Understand the Math?

Question 1: In
 $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$,
what does $[h_{t-1}, x_t]$ mean?

- A) Matrix multiplication
- B) Addition of vectors
- C) Concatenation (stick together)
- D) Element-wise product

Question 2: Which gate uses tanh (not sigmoid)?

- A) Forget gate f_t
- B) Input gate i_t
- C) Candidate memory \tilde{C}_t
- D) Output gate o_t

Question 3: How many parameters does an LSTM learn per gate?

- A) 1 (just the gate value)

Answers & Why They Matter:

Answer 1: C (Concatenation)

$[h_{t-1}, x_t]$ means stick the vectors together: if h_{t-1} is size 256 and x_t is size 100, result is size 356. NOT multiplication or addition!

Answer 2: C (Candidate \tilde{C}_t)

$\tilde{C}_t = \tanh(\dots)$ uses tanh because cell state can be negative or positive. All three GATES (f_t, i_t, o_t) use sigmoid because they need 0-1 range for "how much"!

Answer 3: C (Many)

Each gate has W (weight matrix) + b (bias vector). For hidden=256, input=100: W_f is 256×356 , b_f is 256. That's 91,392 parameters per

Backpropagation Through Time (BPTT)

LSTM Training: Watching It Learn

Epoch 1: Random Initialization

Input: "I love chocolate"

Prediction: "xjwkq"

Loss: 8.5 (Gibberish!)

Epoch 10: Learning Letters

Input: "I love chocolate"

Prediction: "cream"

Loss: 2.1 (Better!)

Epoch 50: Understanding Context

Input: "I love chocolate"

Prediction: "ice cream"

Loss: 0.4 (Good!)

Epoch 200: Fluent Generation

Input: "I love chocolate"

Prediction: "ice cream
and strawberry cake"

Loss: 0.08 (Excellent!)

What This Shows:

Technical Terms:

BPTT: Backpropagation
Through Time. Compute

Where LSTMs Are Used + Summary

Applications:

NLP: Translation, generation, sentiment

Speech: Recognition (Siri), music generation

Time Series: Stock, weather, energy, healthcare

Comparison: N-gram vs RNN vs LSTM

Feature	N-gram	RNN	LSTM
Memory Type	First window	Fading	Selective
Long Context	No	No	Yes
Parameters	Few	Moderate	Many
Training Speed	Fast	Medium	Slow
Vanishing Gradient	N/A	Yes ::	Believe ::
Best For	Short (2-3 words)	Medium (10 words)	Long (50+ words)
Example	"Love."	"The cat sat..."	"The cat, who was... finally..."

Key Insight: LSTMs still relevant in 2025! Complementary to Transformers.

Equations Reference:

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

Summary:

- **Problem:** 50-100 step memory needed
- **Solution:** 3 gates + additive cell state
- **Impact:** Google Translate, foundation for Transformers

When to Use:

LSTMs:

- Time series (SOTA)
- Real-time
- Mobile/edge
- Limited data

Transformers:

- Large datasets
- Parallel training
- Bidirectional
- SOTA NLP

Notation:

- t : time, x_t : input
- h_t : hidden, C_t : cell
- σ : sigmoid (0-1)