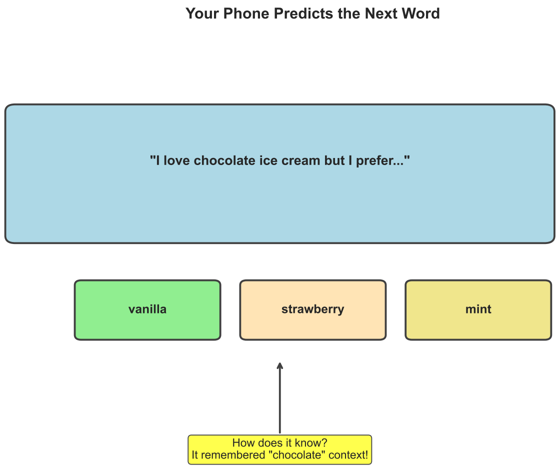
LSTM Networks: A Visual Journey

Understanding Long Short-Term Memory Through Charts

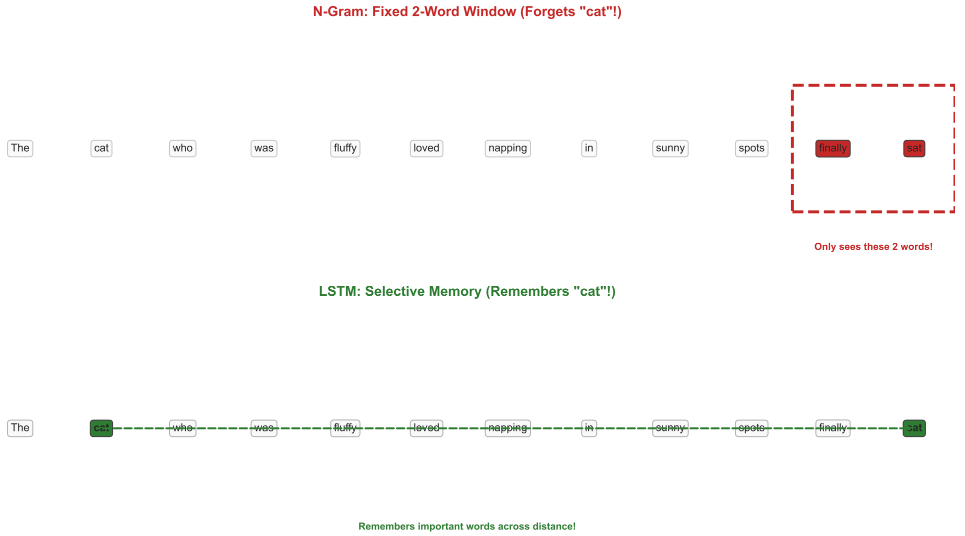
Neural Language Processing

The Challenge: Long-Distance Dependencies


Your Phone Predicts the Next Word
"I love chocolate ice cream but I prefer..."
vanilla
strawberry
mint
How does it know?
It remembered "chocolate" context!

The Problem:

N-grams Can't See Far Enough



What Humans Remember vs What N-grams Remember

Human Memory:

+ Remembers Paris mentioned earlier
+ Connects Paris $\rightarrow$ French
+ Forgets irrelevant details
+ Updates memory with new info

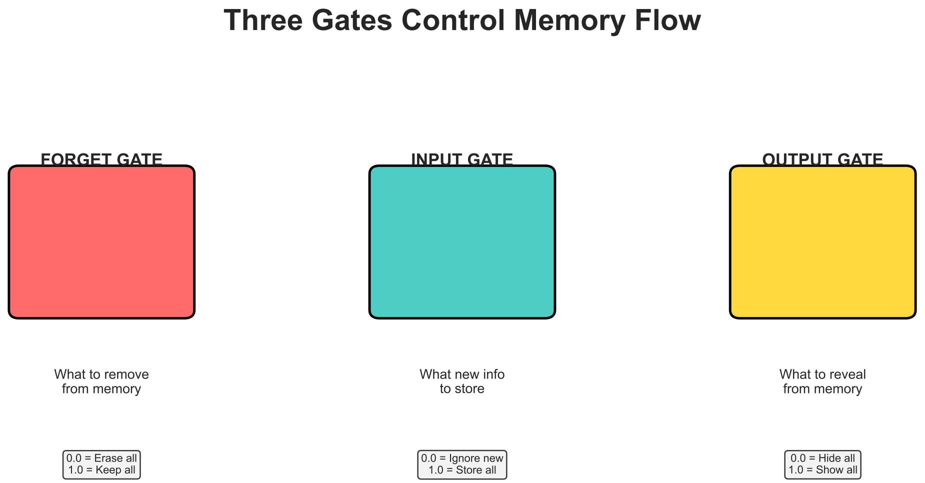
N-gram Memory:

- Only sees last 1-3 words
- No connection to Paris
- Can't distinguish important from irrelevant
- Fixed window, can't adapt

Key Insight:
Humans have **selective memory**:
- Keep important info
- Forget irrelevant details
- Update with new context

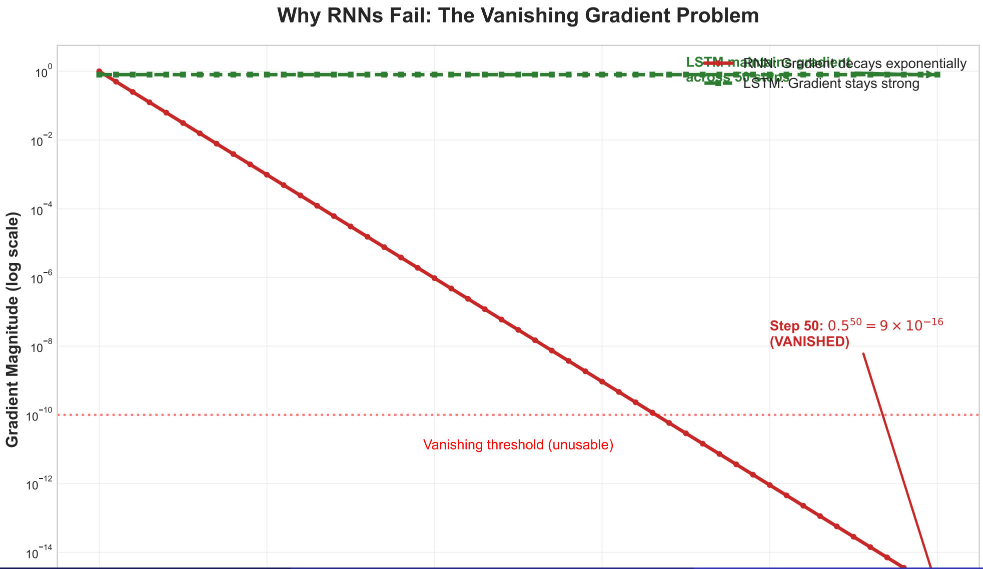
What We Need:
A memory system with:
1. Forget unimportant info
2. Store new important info
3. Retrieve when needed

Three Mechanisms We Need



All gates use Sigmoid function: output between 0 and 1

Why Simple RNNs Fail



Checkpoint 1: Understanding the Problem

Q1: Why can't N-grams solve the Paris problem?

A) Too slow
B) Fixed 1-3 word window
C) Too much memory
D) Don't understand French

A1: B - Fixed window
N-grams use fixed 1-3 word context. Paris is 18 words back - completely invisible!

Q2: What causes RNN gradient vanishing?

A) Too many layers
B) Exponential decay over time
C) Learning rate too high
D) Wrong activation function

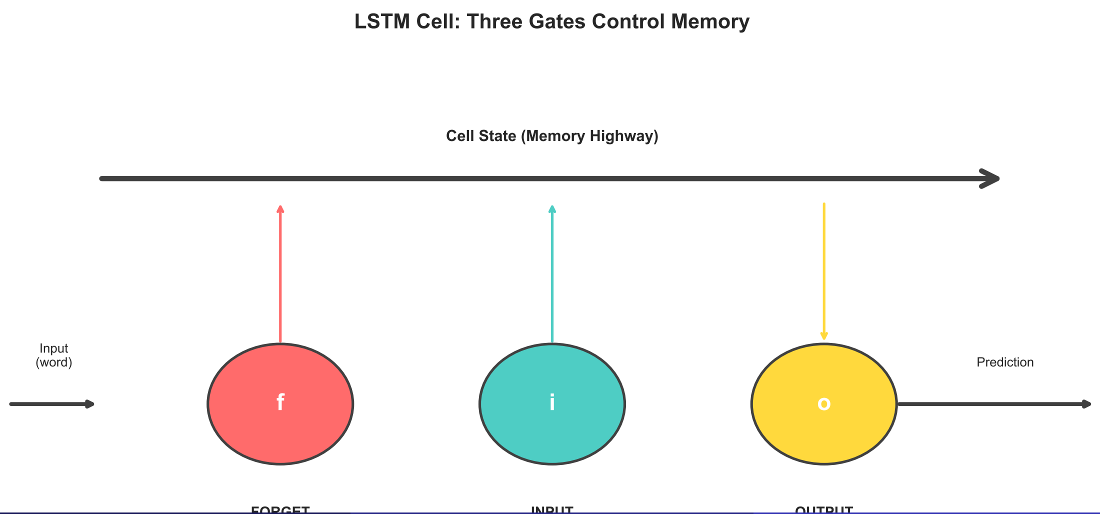
A2: B - Exponential decay
Gradients multiply by <1 at each step. 0.5^{50} becomes vanishingly small!

Q3: How many gate mechanisms does LSTM need?

A) 1: Memory
B) 2: Input + Output
C) 3: Forget + Input + Output
D) 4: All gates + Cell

A3: C - Three gates
Forget (remove), Input (add), Output (reveal)

LSTM Architecture: The Complete Picture


LSTM Cell: Three Gates Control Memory
Cell State (Memory Highway)
Input
(word)
f
i
o
Prediction
FORGET
INPUT
OUTPUT

Notation Guide: Understanding the Symbols

States and Inputs:

- x_t - Input at time t (current word)
- h_t - Hidden state (output) at time t
- h_{t-1} - Previous hidden state
- C_t - Cell state (memory) at time t
- C_{t-1} - Previous cell state

Gates (all 0 to 1):

- f_t - Forget gate (what to erase)
- i_t - Input gate (what to store)
- o_t - Output gate (what to reveal)
- $\tilde{C}_t$ - Candidate memory (-1 to 1)

Operations:

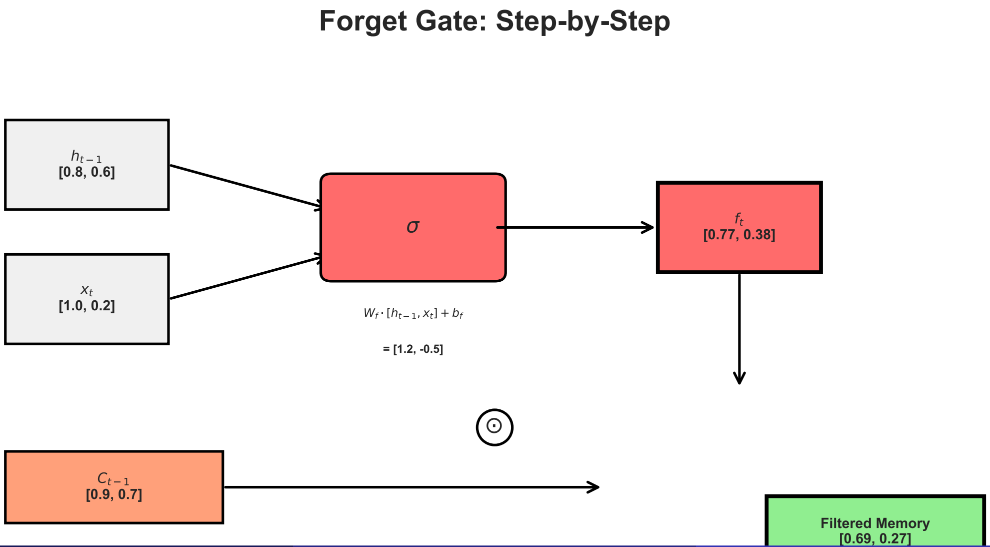
- σ - Sigmoid (0 to 1) for gates
- tanh - Tanh (-1 to 1) for memory
- $\odot$ - Element-wise multiplication
- $[a, b]$ - Concatenation (stick together)

Learned Parameters:

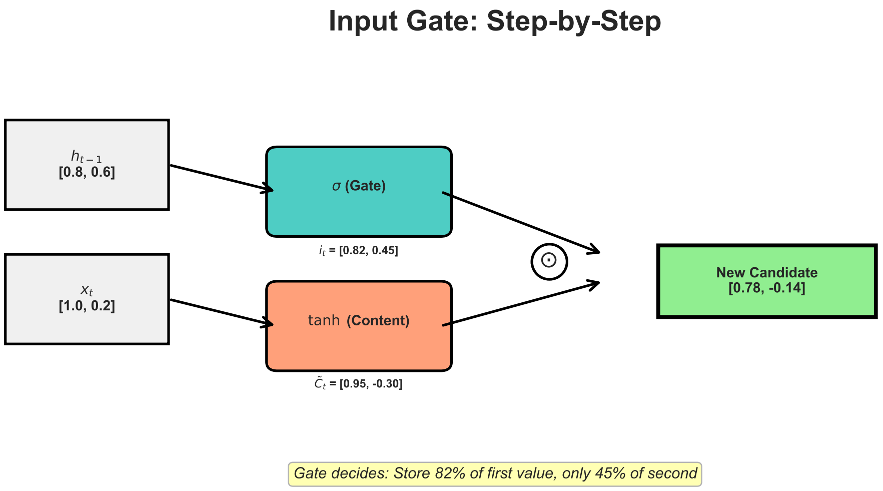
- W_f, W_i, W_o, W_C - Weight matrices
- b_f, b_i, b_o, b_C - Bias vectors

Time step t: Current position in sequence

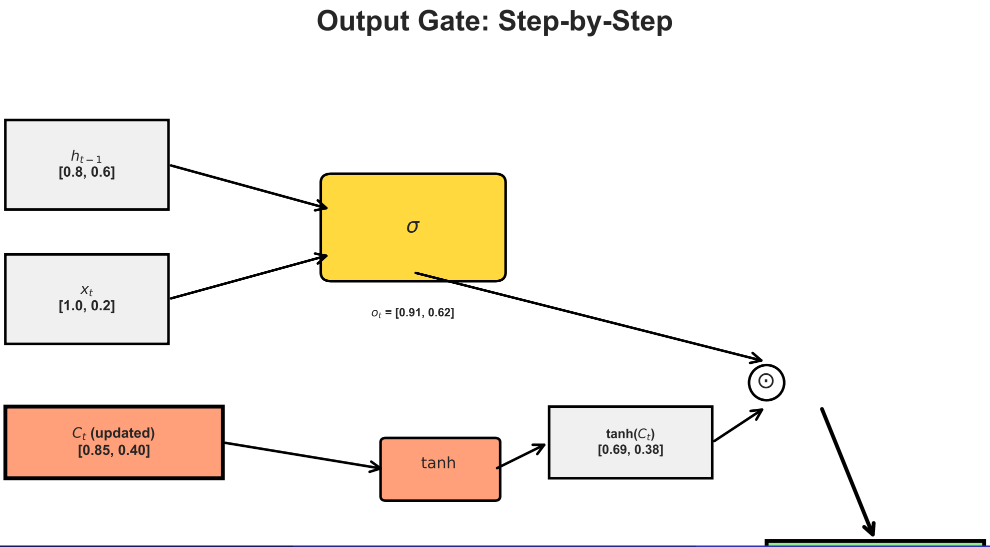
Forget Gate: Deciding What to Erase



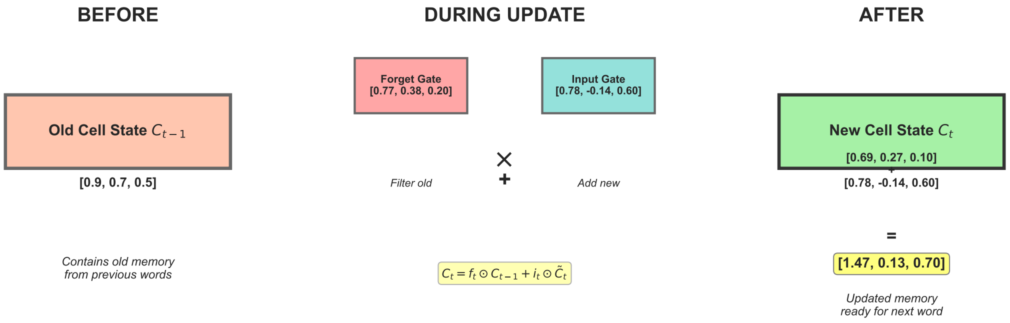
Input Gate: Deciding What to Store



Output Gate: Deciding What to Reveal



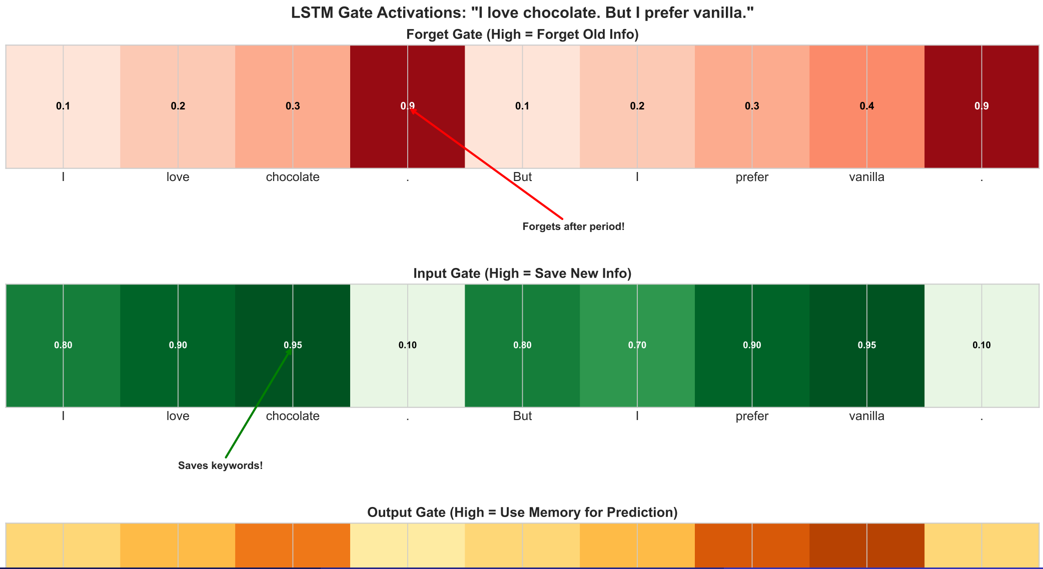
Cell State Update: The Memory Highway



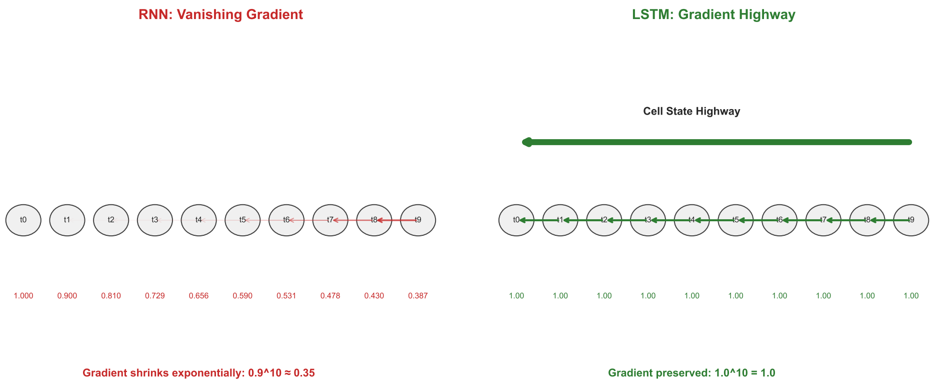
BEFORE
Old Cell State C_{t-1}
[0.9, 0.7, 0.5]
Contains old memory from previous words
DURING UPDATE
Forget Gate [0.77, 0.38, 0.20]
Input Gate [0.78, -0.14, 0.60]
×
+
Filter old
Add new
$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$
AFTER
New Cell State C_t
[0.69, 0.27, 0.10]
+
[0.78, -0.14, 0.60]
=
[1.47, 0.13, 0.70]
Updated memory ready for next word

In Plain English: Old memory C_{t-1} flows through: (1) Forget gate filters it, (2) Input gate adds new info, (3) Result is updated memory C_t. This is the ADDITIVE update that prevents gradient vanishing!

Gate Activations: Real Numbers in Action



Why LSTM Solves Gradient Vanishing



In Plain English: RNN: Gradients MULTIPLY through time ($\times 0.5$ each step = exponential decay). LSTM: Gradients ADD through cell state ($+f_t \odot$ = linear flow). Addition preserves gradients!

Checkpoint 2: Understanding Gates and Cell State

Q1: What does $f_t = 0.2$ mean?

A) Keep 20% of old memory
B) Erase 20% of old memory
C) Add 20% new memory
D) Output 20%

Q2: Why does LSTM have TWO highways (C_t and h_t)?

A) Redundancy
B) Speed
C) C_t = long-term memory, h_t = short-term output
D) Prevent overfitting

A1: A - Keep 20%
Forget gate f_t controls what to KEEP, not erase. $f_t = 0.2$ means keep 20%, erase 80%.

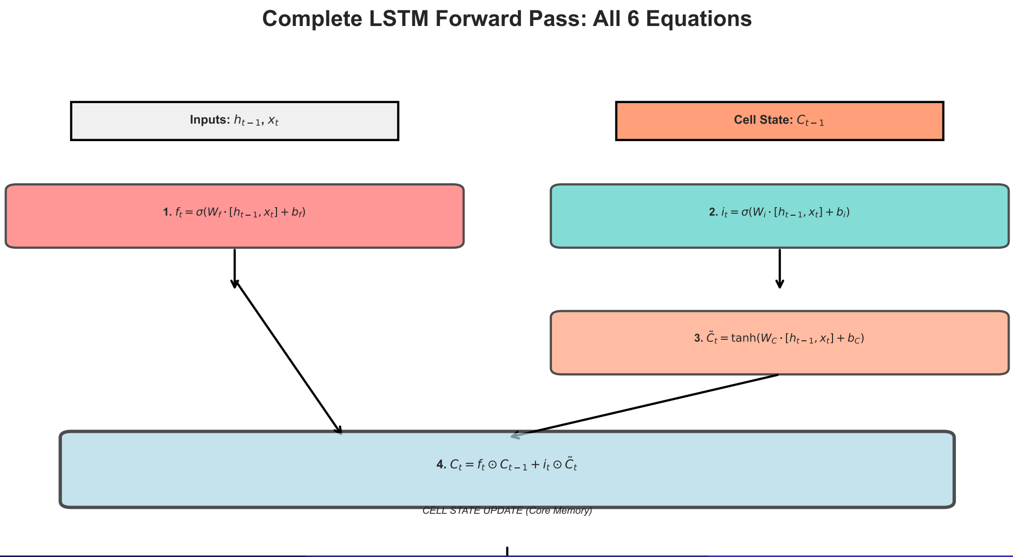
A2: C - Different roles
C_t stores unfiltered long-term memory (highway). h_t is filtered output for predictions.

Q3: What's the key operation that prevents vanishing gradients?

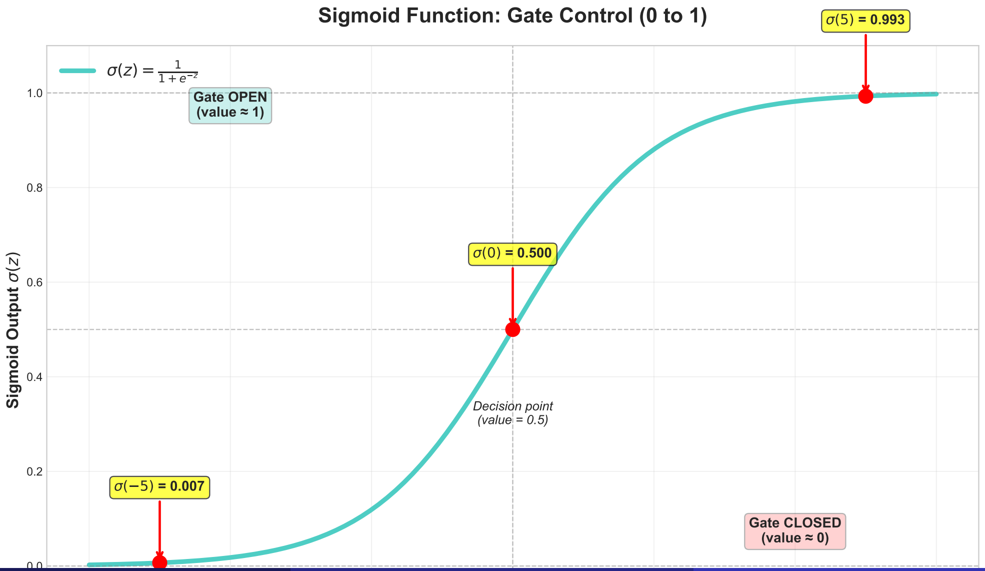
A) Sigmoid activation
B) Additive cell state update
C) Element-wise multiplication
D) Tanh normalization

A3: B - Addition
$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ uses addition, creating gradient highway!

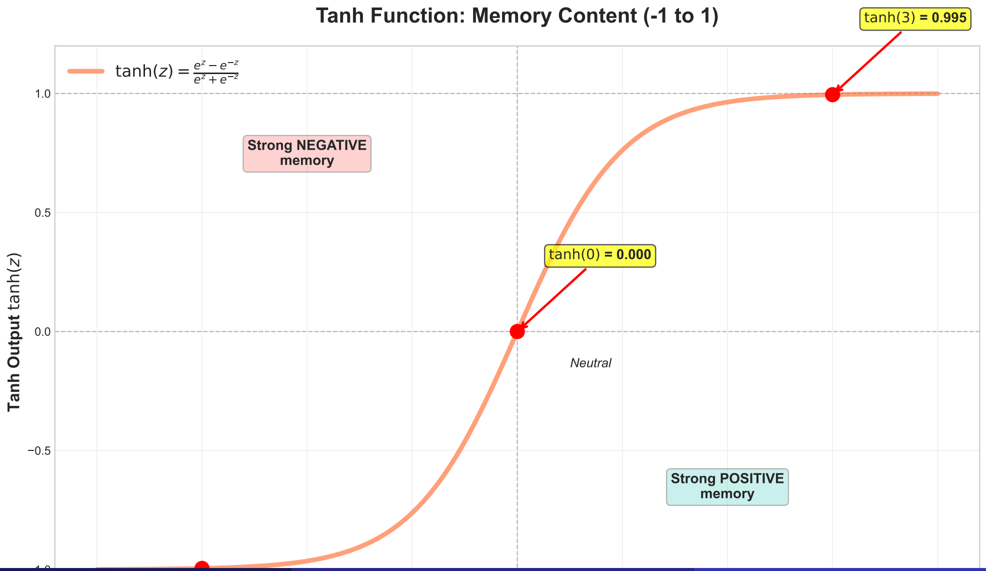
Complete Forward Pass: All 6 Equations as Flowchart


Complete LSTM Forward Pass: All 6 Equations
Inputs: h_{t-1}, x_t
Cell State: C_{t-1}
1. $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
2. $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
3. $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
4. $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$
CELL STATE UPDATE (Core Memory)

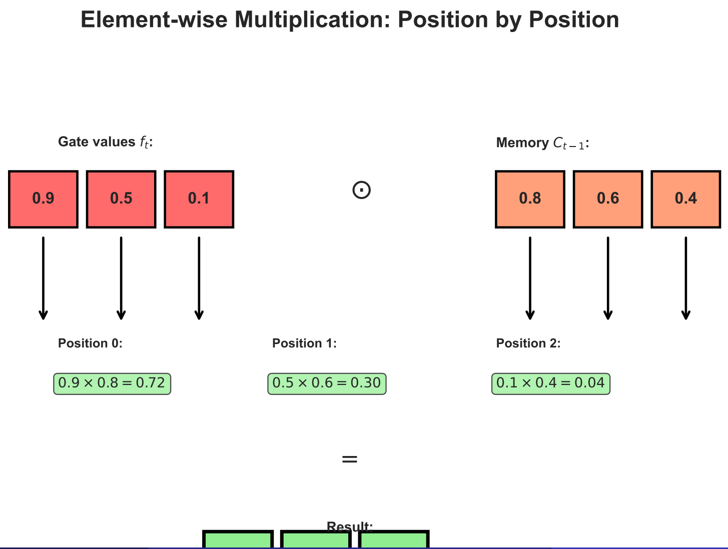
Sigmoid Function: Gate Control Mechanism



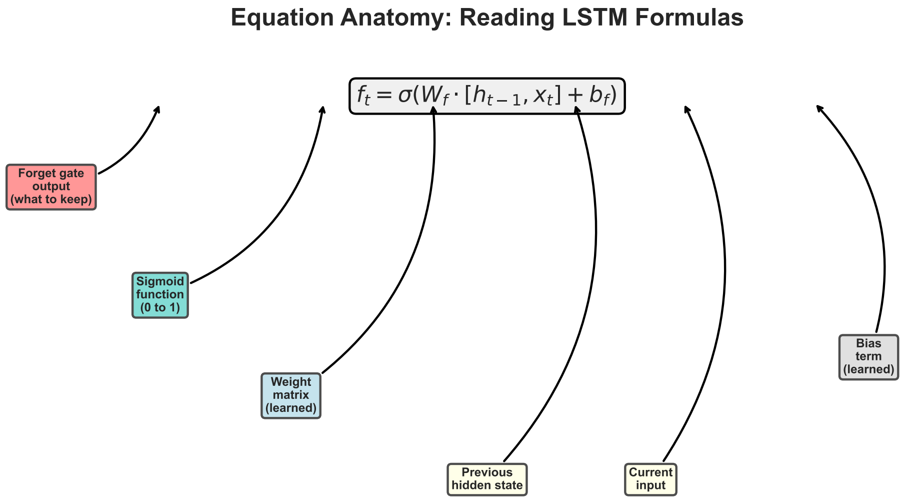
Tanh Function: Memory Content Normalization



Element-wise Multiplication: How Gates Control Memory



Equation Anatomy: Reading LSTM Formulas


Equation Anatomy: Reading LSTM Formulas
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
Forget gate output (what to keep)
Sigmoid function (0 to 1)
Weight matrix (learned)
Previous hidden state
Current input
Bias term (learned)

Numerical Walkthrough: Real Numbers Through LSTM

Step 1: Inputs

- $h_{t-1} = [0.8, 0.6]$
- $x_t = [1.0, 0.2]$
- $C_{t-1} = [0.9, 0.7]$

Step 2: Compute Gates
(Assume weights trained)

- $f_t = \sigma([1.2, -0.5]) = [0.77, 0.38]$
- $i_t = \sigma([1.5, -0.2]) = [0.82, 0.45]$
- $o_t = \sigma([2.0, 0.5]) = [0.88, 0.62]$
- $\tilde{C}_t = \tanh([2.0, -0.3]) = [0.96, -0.29]$

Step 3: Update Cell State

$$\begin{aligned} C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ &= [0.77, 0.38] \odot [0.9, 0.7] \\ &\quad + [0.82, 0.45] \odot [0.96, -0.29] \\ &= [0.69, 0.27] + [0.79, -0.13] \\ &= [1.48, 0.14] \end{aligned}$$

Step 4: Compute Output

$$\begin{aligned} h_t &= o_t \odot \tanh(C_t) \\ &= [0.88, 0.62] \odot \tanh([1.48, 0.14]) \\ &= [0.88, 0.62] \odot [0.90, 0.14] \\ &= [0.79, 0.09] \end{aligned}$$

In Plain English: Watch numbers flow: Forget gate keeps 77% of first memory value, Input gate adds strong new info (0.79), Output gate reveals 88% of normalized cell state to output.

Checkpoint 3: Understanding the Math

Q1: What does $\sigma(-3)$ output?

A) -3
B) Close to 0
C) 0.5
D) Close to 1

Q2: What is $[0.5, 0.8] \odot [2.0, 3.0]$?

A) $[2.5, 3.8]$
B) $[1.0, 2.4]$
C) $[1.5, 2.2]$
D) $[0.25, 0.27]$

A1: B - Close to 0
$\sigma(-3) = 0.047 \approx 0$. Large negative input $\longrightarrow$ gate closed!

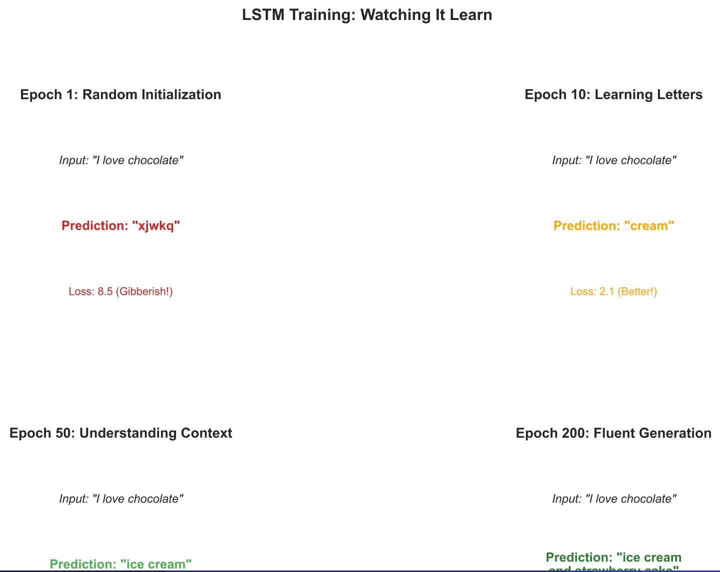
A2: B - [1.0, 2.4]
Element-wise: $[0.5 \times 2.0, 0.8 \times 3.0] = [1.0, 2.4]$

Q3: Why use Tanh for $\tilde{C}_t$ but Sigmoid for gates?

A) Tanh faster
B) Tanh allows negative values,
Sigmoid for 0-1 control
C) Historical reasons
D) Random choice

A3: B - Different purposes
Gates need 0-1 (off/on). Memory needs positive AND negative evidence (-1 to +1)

Training Progression: Watching LSTM Learn



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"

Epoch 200: Fluent Generation

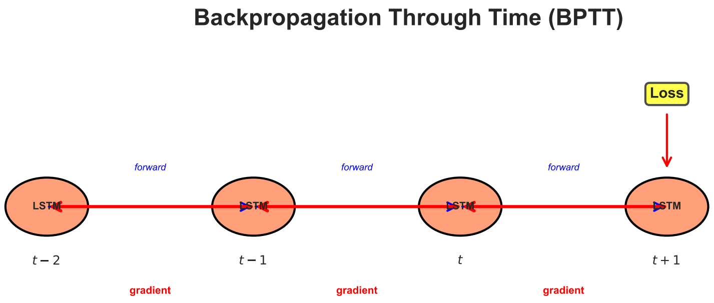
Input: "I love chocolate"

Prediction: "ice cream and strawberry cake"

Training Recipe: Step-by-Step LSTM Training

1. **Data Preparation**
   - Tokenize text into sequences
   - Create input-target pairs
   - Batch sequences of same length
   - Pad if needed
2. **Model Setup**
   - Initialize weight matrices W_f, W_i, W_o, W_C
   - Initialize bias vectors b_f, b_i, b_o, b_C
   - Choose hidden size (e.g., 128, 256, 512)
   - Choose number of layers (1-3 typical)
3. **Forward Pass**
   - Process sequence word by word
   - Update cell state C_t at each step
   - Collect outputs h_t for predictions
4. **Loss Computation**
   - Compare predictions to targets
   - Use cross-entropy loss
   - Average over sequence
5. **Backward Pass (BPTT)**
   - Compute gradients backward through time
   - Cell state acts as gradient highway
   - No vanishing gradient problem!
6. **Weight Update**
   - Apply optimizer (Adam recommended)
   - Learning rate: 10^{-3} to 10^{-4}
   - Gradient clipping: max norm 5.0
7. **Iterate**
   - Repeat for multiple epochs (10-50)
   - Monitor validation loss

BPTT: Backpropagation Through Time

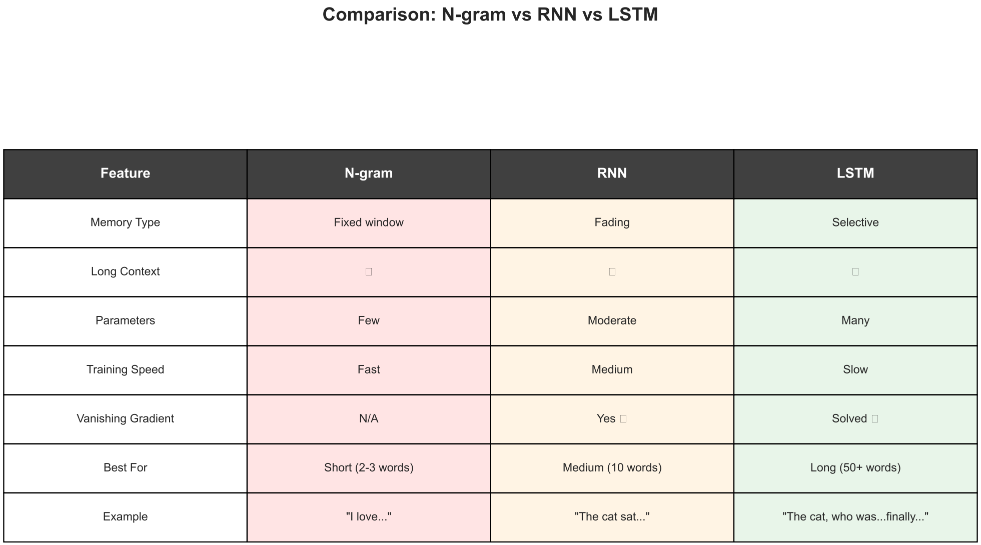


Forward Pass (blue): Compute outputs from left to right

Backward Pass (red): Propagate gradients from right to left

LSTM cell state acts as gradient highway - prevents vanishing!

Model Comparison: When to Use LSTM



Comparison: N-gram vs RNN vs LSTM

Feature	N-gram	RNN	LSTM
Memory Type	Fixed window	Fading	Selective
Long Context	□	□	□
Parameters	Few	Moderate	Many
Training Speed	Fast	Medium	Slow
Vanishing Gradient	N/A	Yes □	Solved □
Best For	Short (2-3 words)	Medium (10 words)	Long (50+ words)
Example	"I love..."	"The cat sat..."	"The cat, who was...finally..."

Applications: Where LSTMs Excel

Natural Language Processing:

- Machine translation
- Text generation
- Sentiment analysis
- Named entity recognition
- Question answering

Time Series:

- Stock price prediction
- Weather forecasting
- Energy demand prediction
- Anomaly detection

Speech and Audio:

- Speech recognition
- Music generation
- Voice synthesis
- Audio classification

Video Analysis:

- Action recognition
- Video captioning
- Motion prediction
- Event detection

Key Advantage:
LSTM excels when **context from distant past** matters for current prediction!

Checkpoint 4: Training and Applications

Q1: What is BPTT?

A) Backward Pass Through Training
B) Backpropagation Through Time
C) Batch Processing Training Technique
D) Bi-directional Processing Through Time

Q2: Why use gradient clipping in LSTM training?

A) Speed up training
B) Prevent exploding gradients
C) Reduce memory usage
D) Improve accuracy

A1: B - Backpropagation Through Time
BPTT unrolls sequence through time and backpropagates errors from future to past.

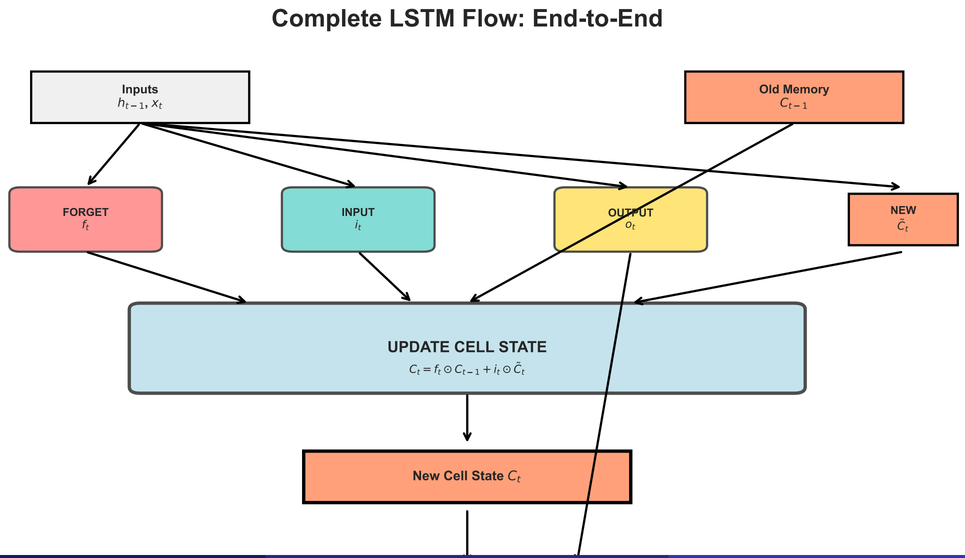
A2: B - Prevent explosion
While LSTM solves vanishing, gradients can still explode. Clipping caps maximum gradient norm.

Q3: When should you NOT use LSTM?

A) Machine translation
B) Short 2-3 word context
C) Speech recognition
D) Time series prediction

A3: B - Short context
For 2-3 word context, simpler models (N-gram, simple RNN) are faster

Complete LSTM Flow: Visual Summary



Quick Reference: Essential LSTM Equations

The Six Core Equations:

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

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

3. Candidate Memory:
$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

4. **Cell State Update:**
$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$

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

6. Hidden State:
$h_t = o_t \odot \tanh(C_t)$

Key Takeaways:

1. **Problem:** N-grams limited window, RNNs vanishing gradient
2. **Solution:** Three gates + cell state highway
3. **Forget gate:** Decides what to erase (0-1)
4. **Input gate:** Decides what to store (0-1)
5. **Output gate:** Decides what to reveal (0-1)
6. **Cell state:** Long-term memory highway (additive update)
7. **Hidden state:** Short-term filtered output
8. **Why it works:** Addition in C_t update creates gradient highway
9. **Use when:** Long-distance dependencies matter
10. **Avoid when:** Short context or parallel processing needed

Remember: Cell state is the **memory highway**, gates control the **traffic**!