LSTM Primer: Next Word Prediction

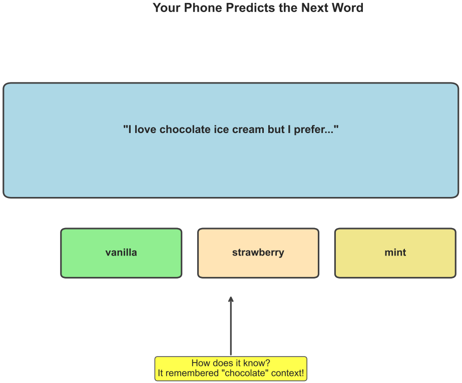
From Autocomplete to Modern Language Models

BSc Level Introduction

September 27, 2025

The Autocomplete Challenge

You're typing on your phone. . .



Why It's Hard:

- Context can be long
- Words far back still matter
- Grammar rules complex
- Meaning changes with context

Example Challenges:

- "I love chocolate. But I prefer ___"
- Need to remember "love/prefer" pattern
- Forget specific "chocolate"
- Context = 7 words back!

This is a sequence modeling problem

The Goal:

- Predict what word comes next

What We've Tried So Far: N-grams

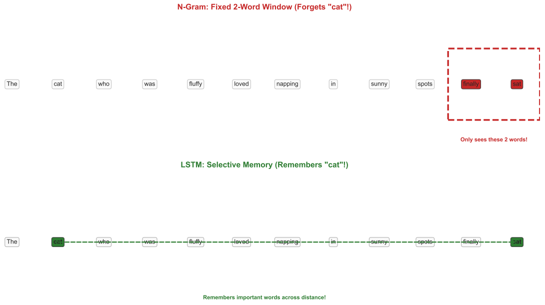
Simple Baseline: Count Word Pairs

How N-grams Work:

- Look at previous 1-2 words only
- Count what usually comes next
- Pick most common continuation

Example:

- Saw "I love" 100 times
- Followed by "you": 60 times
- Followed by "chocolate": 30 times
- Followed by "pizza": 10 times
- Predict: "you" (most common)



Fatal Limitations:

- Only sees 1-2 words back
- Can't handle long context
- No real understanding
- Millions of word combinations

The Memory Problem

Reading a Book: What Do You Remember?

Human Memory:

- You read 200 pages
- Don't remember every word
- Remember: Main plot
- Remember: Key characters
- Forget: Minor details
- Forget: Exact sentences

The Insight:

- Memory is **selective**
- Keep important information
- Discard irrelevant details
- Update as story progresses

What We Need in AI:

- Remember relevant past words
- Forget irrelevant words
- Decide what's important NOW
- Update memory as we read

Example:

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

- Remember: "Paris" (8 words back)
- Forget: "grew up" (not needed now)
- Predict: "French"

We need controllable memory

Recurrent Neural Networks: A Loop of Memory

The RNN Idea:

- Hidden state = memory
- Update memory at each word
- Use memory to predict next word
- Memory flows through time

The Math:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$
$$y_t = \text{softmax}(W_y h_t)$$

- h_t = memory at time t
- x_t = current word
- h_{t-1} = previous memory
- y_t = prediction

The Problem:

When we train (backpropagation):

- Gradient flows backward through time
- Multiplied by same weight matrix
- After 10 steps: $0.9^{10} = 0.35$
- After 20 steps: $0.9^{20} = 0.12$
- After 50 steps: $0.9^{50} = 0.005$

Vanishing Gradient:

- Signal gets weaker each step
- Can't learn from distant past
- Memory effectively only 5-10 words
- Same problem as n-grams!

We need a gradient highway

Why RNNs Forget Long-Term Context

What Happens During Training:

Step 1: Make prediction

- Forward pass through network
- Get prediction error

Step 2: Compute gradient

- How much to adjust weights?
- Flow gradient backward in time

Step 3: Problem appears

- Gradient multiplied at each step
- $\frac{\partial h_t}{\partial h_{t-1}} \approx 0.9$
- After n steps: 0.9^n

The Numbers:

Steps Back	Gradient Strength
1	0.90 (90%)
5	0.59 (59%)
10	0.35 (35%)
20	0.12 (12%)
50	0.005 (0.5%)

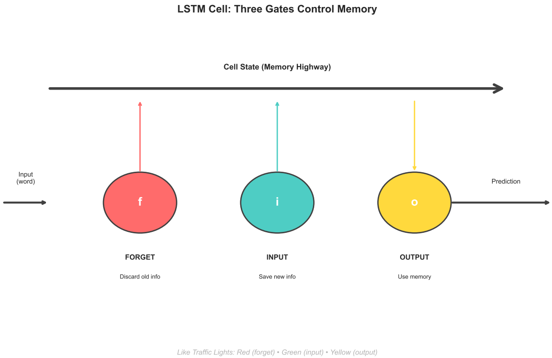
The Impact:

- Word 50 steps back: Almost no learning
- Network can't learn long-term patterns
- Effectively limited to 5-10 words
- Need 50-100+ word context!

Solution: Create a direct path for gradients

LSTM Solution: Selective Memory with Gates

Long Short-Term Memory: Three Gates Control Information Flow



Traffic Light Analogy:

- Red Gate (Forget): What to remove from memory

How It Works:

Forget Gate: (0 to 1)

- 0.0 = Completely forget
- 1.0 = Keep everything
- Example: 0.9 at period = forget old topic

Input Gate: (0 to 1)

- 0.0 = Ignore new word
- 1.0 = Fully store
- Example: 0.95 on "Paris" = remember!

Output Gate: (0 to 1)

- 0.0 = Hide memory
- 1.0 = Reveal everything
- Example: 0.8 when predicting = use memory

Inside the LSTM: Gate Equations

From Intuition to Mathematics

The Four Equations:

1. Forget Gate:

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

2. Input Gate:

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

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

3. Update Cell State:

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

4. Output Gate:

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

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

Concrete Example:
Input: "I love"

Values at "love":

- $f_t = 0.3$ (forget 70% of "I")
- $i_t = 0.9$ (strongly store "love")
- $\tilde{C}_t = [0.8, -0.3, 0.5, \ldots]$ (new info)
- $C_t = 0.3 \cdot C_{t-1} + 0.9 \cdot \tilde{C}_t$
- $o_t = 0.7$ (reveal 70% of memory)
- $h_t = 0.7 \cdot \tanh(C_t)$

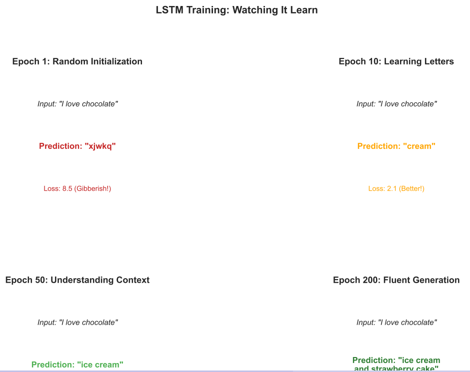
Key Insight:

- σ = sigmoid (0 to 1)
- $\odot$ = element-wise multiply
- C_t = cell state (long-term memory)
- h_t = hidden state (short-term output)

All gates learned automatically during training

Training Progression: Watching It Learn

How LSTM Improves Over Time



What's Happening:

Epoch 1 (Random):

- Gates untrained
- Random predictions
- No pattern learning
- Loss: 8.5

Epoch 10 (Bigrams):

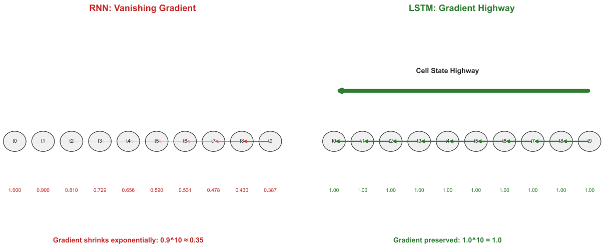
- Learns immediate pairs
- "I" → "love" pattern
- Still struggles with context
- Loss: 4.2

Epoch 50 (Context):

- Gates start working
- Remembers further back

Why LSTMs Work: The Gradient Highway

Comparing RNN vs LSTM Gradient Flow



RNN Problem:

- Gradient: $0.9^{10} = 0.35$
- Multiplied at every step
- Exponential decay
- Can't learn long-term

LSTM Solution:

- Gradient: $1.0^{10} = 1.0$
- Addition in cell state
- No multiplication!
- Learns 50+ steps back

The Key Equation:

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

- Addition (+) not multiplication

Modern Applications & Summary

From Theory to Real-World Impact

Where LSTMs Are Used:

- **Autocomplete**: Phone keyboards
- **Translation**: Google Translate (2016-2019)
- **Speech Recognition**: Siri, Alexa
- **Sentiment Analysis**: Product reviews
- **Text Generation**: Creative writing
- **Music Generation**: Compose melodies
- **Video Captioning**: Describe videos

PyTorch Implementation:

```
import torch.nn as nn
lstm = nn.LSTM(input_size=100,
  hidden_size=256, num_layers=2)
output, (h_n, c_n) = lstm(input)
```

Key Takeaways:

1. The Problem:

- Need to remember long context
- RNNs couldn't learn beyond 5-10 words
- Vanishing gradient problem

2. The Solution:

- Three gates control memory
- Cell state = gradient highway
- Additive updates preserve gradients

3. The Impact:

- 50-100+ word context
- Breakthrough in NLP (2015-2018)
- Foundation for modern transformers

What's Next: