SCALER School of Technology × Priyansh Saxena

LSTMs, GRUs, and Gating Mechanisms

LSTM Cell Anatomy • Gradient Flow • GRU • Bidirectional • Deep RNNs

Priyansh Saxena

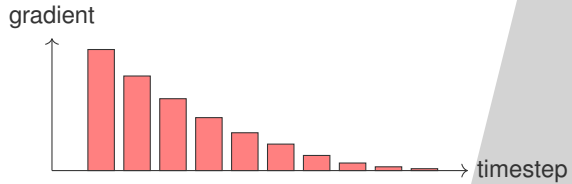
Scaler School of Technology | Deep Learning I: Neural Networks | Week 6 — Session 1/2

5 June 2026

The Problem We're Solving

Recap from Last Week

Vanilla RNNs suffer from **vanishing gradients** — can't learn dependencies beyond $\sim$20 steps.



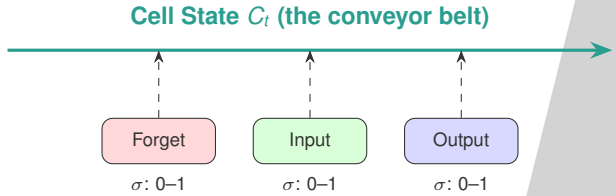
Gradients vanish $\rightarrow$ can't learn from early inputs

We need: an architecture where gradients flow **without** repeated multiplication by W_{hh}.

Today’s Roadmap

“Last week RNNs broke. This week we fix them — with gates.”

The Key Idea: A Conveyor Belt

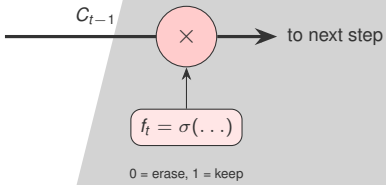


- Information rides the conveyor belt **unchanged** unless a gate intervenes
- Gates are sigmoid networks: output 0 (closed) to 1 (open)
- Only element-wise operations on the belt — **no matrix multiplication**!

The Forget Gate

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

- Sigmoid output: value between 0 and 1 **per cell dimension**
- 0 = **completely forget** this piece of information
- 1 = **completely keep** this information
- Applied element-wise to previous cell state



Example

New paragraph, new character → forget gate clears old character details.

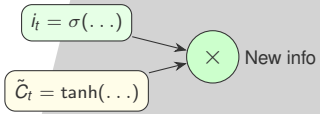
Implementation detail: b_f initialized to 1 → biased toward remembering by default.

The Input Gate

Two parts:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \qquad \text{(which cells to update)}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \qquad \text{(candidate values)}$$

- i_t: sigmoid gate decides **which dimensions** to write to
- $\tilde{C}_t$: tanh creates candidate values in $[-1, 1]$
- Together: $i_t \odot \tilde{C}_t$ = new information to add



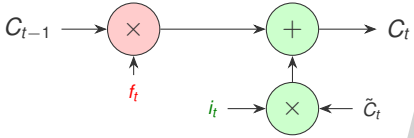
Example

"The hero is actually the villain" → input gate writes this plot twist to memory.

Updating the Cell State

The Core LSTM Equation

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

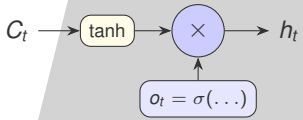


- Left: old memory $\times$ forget gate (some stuff erased)
- Right: new candidate $\times$ input gate (some stuff written)
- **This is ADDITION, not multiplication!** $\rightarrow$ gradients flow freely!

The Output Gate

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

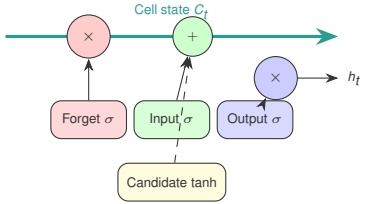
- Cell state “knows” many things
- Output gate decides **which parts to reveal** right now
- h_t is the **filtered** version of cell state
- Used for: predictions AND passed to next timestep



Analogy

You know the entire plot, but if asked “what color is the car?” you only output the relevant detail.

The Complete LSTM Cell



	RNN vs **LSTM**
Matrices	1 vs 4
Gates	0 vs 3
States	h vs $h + C$
Params	$1\times$ vs $4\times$
Range	$\sim$20 vs 100+

$4\times$ the parameters,
but handles sequences
10–$100\times$ longer!

Quick Check

1. If the forget gate outputs **all zeros**, what happens?
   - ✓ Cell state is wiped clean — total amnesia. $C_t = 0 + i_t \odot \tilde{C}_t$
2. Which component creates **candidate new values** to add?
   - ✓ The tanh candidate: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
3. How many sigmoid activations in one LSTM cell?
   - ✓ 3 — forget gate, input gate, output gate (each uses sigmoid)

Coming Up Next

Part 2: Gradient Flow, GRU, Bidirectional, and Deep RNNs

Why LSTMs Solve Vanishing Gradients

The Key Insight

$$\frac{\partial C_t}{\partial C_{t-1}} = f_t \qquad \text{(just the forget gate!)}$$

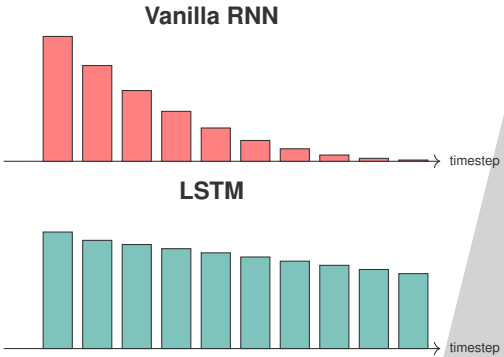
- If $f_t \approx 1$: gradient flows **perfectly** — no decay!
- No W_{hh} multiplication in the cell state path
- The forget gate **learns** when to let gradients through
- Same principle as ResNet skip connections (independently invented 18 years earlier!)

Compare

RNN: $\frac{\partial h_t}{\partial h_{t-1}} = W_{hh}^T \cdot \text{diag}(\tanh'(\cdot))$ — repeated matrix multiply $\rightarrow$ vanishes

LSTM: $\frac{\partial C_t}{\partial C_{t-1}} = f_t$ — element-wise, can stay near 1 $\rightarrow$ survives

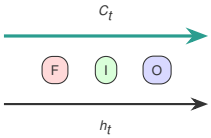
Gradient Flow: RNN vs LSTM



- RNN: gradient at timestep 1 $\approx 0.001\times$ loss gradient
- LSTM: gradient at timestep 1 $\approx 0.6\times$ loss gradient

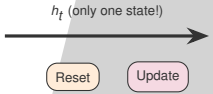
GRU: Fewer Gates, Similar Power

LSTM (3 gates)



2 states, 3 gates, 4× params

GRU (2 gates)



1 state, 2 gates, 3× params

Key Simplifications

1. Merge cell state and hidden state into **one** state
2. Use 2 gates instead of 3 (update gate does both forget + input)

GRU Equations

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \quad \text{(Update gate)}$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \quad \text{(Reset gate)}$$
$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t]) \quad \text{(Candidate)}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad \text{(Final state)}$$

Key Insight

The update gate z_t does **both** forget and input in one shot:

- $(1 - z_t)$ portion: keep from old state (like forget gate)
- z_t portion: take from new candidate (like input gate)

When $z_t = 0$: $h_t = h_{t-1}$ (perfect memory — gradient flows unchanged!)

LSTM vs GRU: When to Choose What

	LSTM	GRU
Gates	3	2
Separate cell state	Yes	No
Parameters	$4\times$ RNN	$3\times$ RNN
Training speed	Slower	$\sim$20% faster
Performance	$\approx$ similar on most tasks	

Rule of Thumb

- Start with **GRU** (simpler, faster)
- Switch to **LSTM** if: very long dependencies, GRU underperforms, or you have plenty of data/compute
- In practice: the difference is often negligible

The Limitation of Forward-Only

Problem

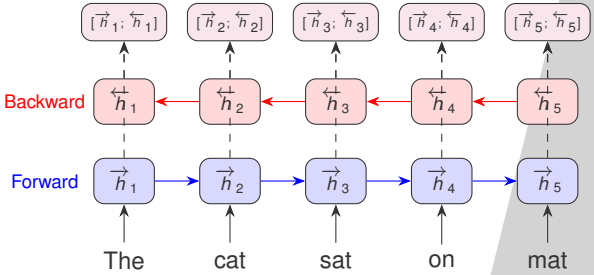
Standard RNN/LSTM only sees **past** context at each position.

Examples where future context helps

- "The _____ barked loudly" — knowing "barked" helps predict "dog"
- "Apple released a new phone" — is Apple the fruit or the company? Need future context!
- NER: "Washington visited Philadelphia" — person or place?

Key question: Do you have the full sequence before making predictions?

Bidirectional Architecture



Output at each position: concatenation of forward + backward hidden states.
Output dimension: $2 \times$ hidden_size

When to Use Bidirectional

USE when:

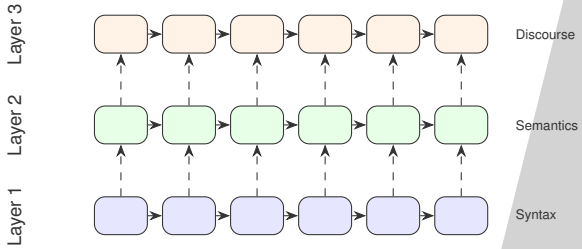
- Full sequence available first
- Text classification
- Named entity recognition
- Encoder in seq2seq
- POS tagging

DON'T USE when:

- Real-time / streaming
- Autoregressive generation
- Language modeling
- Can't see future tokens

"Bidirectional = you already have the whole input.
Unidirectional = you're generating one token at a time."

Stacking RNN Layers



- Layer 1 output $\rightarrow$ Layer 2 input (hierarchical features)
- Each layer: different level of abstraction
- PyTorch: `nn.LSTM(input_size, hidden_size, num_layers=3)`

Deep RNNs: Practical Tips

1. **Typical depth:** 2–4 layers (diminishing returns beyond)
2. **Dropout:** Apply **between** layers, not within timesteps
   - `nn.LSTM(..., dropout=0.3)` — applies between layers automatically
3. **Residual connections:** For 4+ layers, add skip connections between layers
4. **Famous examples:**
   - Google NMT (2016): 8-layer LSTM
   - Deep Speech 2 (2016): 7-layer bidirectional

The 2015–2017 Default

2-layer bidirectional LSTM with dropout $\rightarrow$ the go-to architecture for NLP before transformers.

When to Use What

Scenario	Architecture	Why
Short sequences, simple task	Vanilla RNN / GRU	Sufficient capacity
Long-range dependencies	LSTM or GRU	Gates preserve memory
Need speed / fewer params	GRU	75% of LSTM params
Full sequence available	Bidirectional	Both-direction context
Complex task, lots of data	Deep (2–4 layers)	Hierarchical features
Maximum NLP performance	Deep BiLSTM	All of the above

2024 Reality Check

Transformers have largely replaced LSTMs for NLP. But gates remain essential knowledge — transformers use similar gating concepts (residual streams, multiplicative interactions).

Final Quiz

1. Why does the LSTM cell state help gradient flow?
   - ✓ $\partial C_t / \partial C_{t-1} = f_t$ (element-wise, no matrix multiplication). If $f_t \approx 1$, gradient passes unchanged.
2. Name one scenario where you'd pick GRU over LSTM.
   - ✓ When you need faster training, fewer parameters, or LSTM shows no clear advantage on your task.
3. When should you NOT use a bidirectional RNN?
   - ✓ Real-time/streaming, autoregressive generation — any task where future tokens aren't available at prediction time.

Recap & Key Takeaways

What We Learned

- **LSTM:** 3 gates (forget, input, output) + cell state = gradient highway
- **GRU:** 2 gates (reset, update), no separate cell state, 75% params, similar performance
- **Bidirectional:** forward + backward = full context (when sequence is complete)
- **Deep/Stacked:** multiple layers = hierarchical features (2–4 layers typical)
- **All** solve vanishing gradients through **gating mechanisms**

Next: Sequence-to-Sequence and Attention

The Remaining Problem

Even LSTMs compress the **entire** input into one fixed-size vector h_T.

For translation: “The students who came from France speak French” $\rightarrow$ **one vector** $\rightarrow$ output.

That single vector is a **bottleneck!**

Coming Next Week

- Encoder-decoder architecture
- The **attention mechanism** — let the decoder look back at any input position
- “No more bottleneck — the decoder can focus on what matters.”

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

www.priyanshsaxena.com/sst-deep-learning-nn