Week 3: Teaching Networks to Remember

Discovering RNNs, LSTMs, and the Memory Problem
Pre-Lab Exercise (No Programming Required)
INSTRUCTOR VERSION WITH ANSWER KEY

NLP Course 2025

Time: 40 minutes
Objective: Discover why networks need memory and how gates solve the vanishing gradient problem.

Teaching Note

This handout builds intuition for RNNs through discovery. Students don't know these architectures yet. Let them struggle with the memory problem before revealing solutions. The "aha!" moment when they realize they've invented LSTM gates is powerful.

Part 1: The Context Problem (10 minutes)

Why Order Matters

Task 1: Same words, different meanings
Rearrange these words to create two sentences with opposite meanings:

{dog, bites, man}

Sentence 1: *Dog bites man (common event)*
Sentence 2: *Man bites dog (newsworthy event)*
Teaching Note: This simple example shows that word order fundamentally changes meaning - something Word2Vec cannot capture.

Task 2: Context tracking
Read this sentence word by word and track what you remember:
"The student who studied hard..."

1. After "The": I expect *a noun/person/thing*
2. After "student": I remember *the subject is "student" (singular)*
3. After "who": I expect *a description/action about the student*
4. After "studied hard": I'm waiting for *what happened to the student / main verb*

Task 3: What information are you maintaining?
List three things your brain tracks while reading:

- *Subject of the sentence (who/what)*
- *Tense and number agreement (singular/plural, past/present)*
- *Incomplete dependencies (waiting for main verb)*

Teaching Note: Students should realize they naturally maintain multiple types of information - this motivates why simple feedforward networks fail.

Think About It

Word embeddings (Week 2) treat "dog bites man" and "man bites dog" as identical - they have the same word vectors. How can we teach networks that order matters?

Part 2: Building Memory - Discovering Hidden States (10 minutes)

Designing Memory for Networks

Scenario: You're teaching a computer to read "The cat sat on the mat" word by word.
Task 1: Design your memory system
The computer can only see one word at a time. How would you help it remember previous words?
Your design: *Keep a summary/compressed representation of what we've seen so far. Update this summary with each new word. This summary should influence how we process the next word.*
Teaching Note: Look for students who suggest: keeping all previous words (too much memory), keeping just the last word (not enough context), or some form of summary (correct intuition).
Task 2: Memory updates
Fill in how memory should update at each step:

Current Word	Previous Memory	New Memory Should Contain
"The"	(empty)	*"The" / article / expect noun*
"cat"	"The"	*"The cat" / subject identified*
"sat"	"The cat"	*"The cat sat" / subject + action*

Task 3: Mathematical pattern
The pattern you discovered can be written as:
New Memory = Function(Current Word, Previous Memory)
Or mathematically: $h_t = f(x_t, h_{t-1})$
What are the inputs to this function?

- Input 1: *Current word (x_t)*
- Input 2: *Previous memory/hidden state (h_{t-1})*

What is the output? *Updated memory/hidden state (h_t) that combines both inputs*

Discovery Moment

Congratulations! You've just invented the core idea of RNNs - using hidden states (h_t) to maintain memory of previous inputs!

Teaching Note: This is the key conceptual breakthrough. Make sure everyone understands that h_t is a compressed representation of the entire history up to time t.

Part 3: When Memory Fails - The Vanishing Gradient Problem (10 minutes)

Long-Distance Dependencies

Task 1: The forgetting problem

Try to complete this sentence: "The keys that I left on the table in the kitchen yesterday before going to work *are* missing."

What word did you need to remember? *"keys" (plural)*

How many words back was it? *13-14 words*

Teaching Note: This demonstrates that agreement (are vs is) requires long-distance memory. Simple RNNs fail at this distance.

Task 2: Memory decay simulation

Imagine your memory fades by 10% at each word. Starting with strength 1.0:

Step	Calculation	Memory Strength
Start	1.0	1.0
After 1 word	1.0×0.9	0.9
After 2 words	0.9×0.9	*0.81*
After 5 words	0.9^5	*0.59*
After 10 words	0.9^{10}	*0.35*
After 20 words	0.9^{20}	*0.12*

Task 3: The problem

At what point does memory become effectively zero (¡ 0.01)? *After 44 words (*$0.9^{44} \approx 0.01$*)*

This is why RNNs can't handle long sequences - the gradient (learning signal) vanishes!

Teaching Note: Key insight: This isn't just about forward pass memory - it's about the gradient during backprop. If gradient = 0, no learning happens for early words.

Think About It

If memory decays exponentially, how can we preserve important information for longer?

Part 4: The Gate Solution - Selective Memory (10 minutes)

Designing Gates

Scenario: You're managing your email inbox. You have three actions:

- Delete (forget) spam
- Save (input) important emails
- Reply to (output) urgent emails

Task 1: Gate design
For the sentence "The cat that was black sat on the mat", decide what to do with each word:

Word	**Forget?** (0=forget all, 1=keep all)	**Save?** (0=ignore, 1=save)	**Use Now?** (0=hide, 1=use)
"The"	0.5	0.3	0.2
"cat"	*0.9*	*0.9*	*0.8*
"that"	*0.7*	*0.2*	*0.1*
"was"	*0.7*	*0.2*	*0.1*
"black"	*0.8*	*0.5*	*0.3*
"sat"	*0.9*	*0.9*	*0.9*

Note: "cat" and "sat" are crucial (high values), while "that was" are less important (lower values)
Teaching Note: There's no single correct answer - the key is that students understand selective importance. Main words (cat, sat) should have higher values than function words (that, was).
Task 2: Gate benefits
How do gates help with the vanishing gradient problem?
If forget gate = 1.0, then memory decay = 1.0^{20} = *1.0 (no decay!)*
This creates a "highway" for gradients!
Task 3: Memory update with gates
New formula with gates:

$$\text{New Memory} = (\text{Forget Gate} \times \text{Old Memory}) + (\text{Input Gate} \times \text{New Info})$$

Write this mathematically using your notation:
$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$
Teaching Note: This is the actual LSTM cell state equation! Students have derived it from first principles.

Discovery Moment

You've just invented LSTM! The three gates you designed (Forget, Input/Save, Output/Use) are exactly how LSTMs work!

Part 5: Comparing Memory Systems (10 minutes)

RNN vs LSTM vs GRU

Task 1: Complete the comparison
Based on your discoveries, fill in this table:

Aspect	**Simple RNN** (No gates)	**LSTM** (3 gates)	**GRU** (2 gates)
How it handles memory	Overwrites completely	*Selective forget/remember*	Selective update
Maximum sequence length	10-20 words	*100+ words*	100 words
Gradient flow	Exponential decay	*Preserved via cell state highway*	Good preservation
Best for	*Short sequences, simple patterns*	Long documents	Quick training

Task 2: Application matching
Which architecture would you choose for:

1. Predicting the next character in a name: *Simple RNN (short sequence)*
2. Summarizing a 10-page document: *LSTM (long-range dependencies)*
3. Real-time speech recognition on phone: *GRU (good performance, faster)*
4. Analyzing sentiment in tweets: *GRU or simple RNN (short text)*

Task 3: Design your own architecture
If you could add a fourth gate to LSTM, what would it do?
Gate name: *Example: "Attention gate" or "Importance gate"*
Function: *Example: "Identifies which parts of memory are most relevant for current prediction"*
When would it activate (0 or 1)? *Example: "High (near 1) when seeing question words, low for filler words"*
Teaching Note: This is open-ended - look for creative thinking. Some students might independently invent attention mechanisms!

Synthesis Questions

1. **Why can't we just make the memory infinitely large?**

   Computational cost grows with memory size. More parameters = harder to train, risk of overfitting. Also, infinite memory would remember irrelevant details instead of learning to extract important features.

2. **The forget gate seems counterintuitive - why would we want to forget?**

   Forgetting irrelevant information is crucial. Example: After processing "The cat sat. The dog ran." we need to forget "cat" when focusing on "dog". Selective forgetting = selective attention.

3. **How is an LSTM like a computer's RAM?**

Both have: 1) Read/write operations (gates), 2) Selective memory updates, 3) Ability to preserve information long-term, 4) Limited capacity requiring intelligent management.

4. **Could we have more than 3 gates? What would be the trade-off?**

   Yes, but more gates = more parameters = slower training and risk of overfitting. The 3-gate design balances expressiveness with trainability. (Note: Attention mechanisms essentially add more sophisticated gating.)

Instructor Notes

Key Teaching Points

1. **Build intuition before math**: Let students discover the need for memory naturally
2. **Vanishing gradient is not intuitive**: Use the decay calculation to make it concrete
3. **Gates as decisions**: Frame gates as binary decisions (keep/forget) before introducing continuous values
4. **LSTM complexity**: Don't worry if students don't fully grasp LSTM - the key is understanding why gates help

Common Misconceptions

- **Teaching Note:** RNNs don't "remember everything" - they compress history into fixed-size hidden state
- **Teaching Note:** Vanishing gradient affects learning, not just forward pass
- **Teaching Note:** LSTM doesn't solve vanishing gradient completely, just mitigates it
- **Teaching Note:** More gates isn't always better - GRU often outperforms LSTM

Extension Activities

For advanced students:

- Implement RNN from scratch in NumPy
- Visualize hidden states during text processing
- Experiment with gradient clipping
- Compare LSTM vs GRU on same task

Assessment Ideas

- Can student explain why order matters in language?
- Can they describe the vanishing gradient problem?
- Do they understand how gates create gradient highways?
- Can they choose appropriate architecture for a given task?

End of Instructor Version