Week 4: Recurrent Neural Networks

Memory Matters: RNNs for Sequential Prediction

Next-Word Prediction Course

Learning Objectives

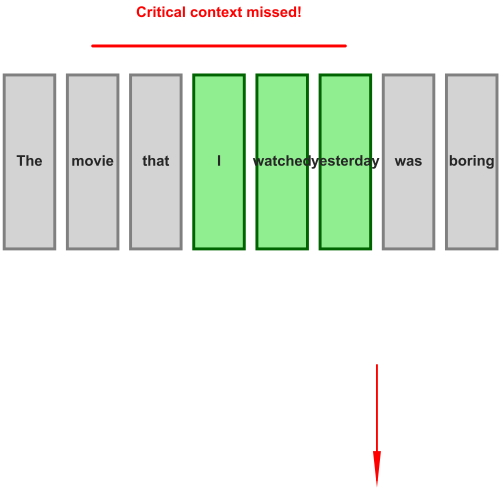
By the end of this week, you will understand:

- How RNNs maintain **memory** across sequences
- The concept of **hidden states** as context representation
- **Backpropagation through time** (BPTT) for training
- The **vanishing gradient** problem and its implications
- Why RNNs are better than n-grams for **long-range dependencies**

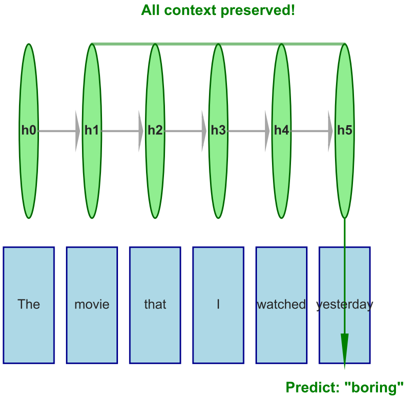
Key Innovation: Networks that remember their past to predict the future!

The Problem: Sequential Memory in Language

Feedforward Network Limitation



RNN Solution



Why RNNs Matter for Language

Limitations of Feedforward Networks:

1. **No Memory**: Each prediction independent
2. **Fixed Context**: Cannot handle variable-length sequences
3. **No State**: Cannot track discourse or dialogue
4. **Redundant Parameters**: Separate weights for each position

The RNN Solution:

- **Hidden State**: Maintains summary of past
- **Parameter Sharing**: Same weights across time
- **Dynamic Context**: Adapts to sequence length
- **Sequential Processing**: Natural for language

Real-World Impact: Smart Text Prediction

Example: Email Autocomplete

- Context: "Thank you for your..."
- Feedforward: Only sees fixed window
- RNN: Remembers entire email thread
- Better predictions from fuller context

Early Applications (1990s-2000s):

- Speech recognition systems
- Handwriting recognition
- Early machine translation
- Text-to-speech synthesis

Key Insight: Language is inherently sequential - our models should be too!

Historical Context: The Evolution to RNNs

Timeline of Sequential Models:

- **1986**: Jordan Networks - Output fed back as input
- **1990**: Elman Networks - Hidden state recurrence
- **1997**: LSTM proposed to solve gradient issues
- **2000s**: RNNs for language modeling take off
- **2010s**: Deep RNNs achieve state-of-the-art

Key Pioneers:

- Jeffrey Elman: Simple recurrent networks
- Michael Jordan: Alternative recurrent architecture
- Yoshua Bengio: Gradient flow analysis
- Jürgen Schmidhuber: Long short-term memory

Evolution: From Feedforward to Recurrent

Feedforward Limitations:

- Fixed input size
- No temporal dynamics
- Position-dependent weights
- Cannot model sequences

Architectural Evolution:

1. Feedforward: $y = f(Wx + b)$
2. Time-delay: $y_t = f(Wx_t + Ux_{t-1} + b)$
3. Recurrent: $h_t = f(Wx_t + Uh_{t-1} + b)$

Recurrent Innovations:

- Variable sequence length
- Hidden state evolution
- Weight sharing across time
- Natural sequence modeling

Limitations: Why We Need Recurrence

N-gram Models:

- Fixed context window
- Exponential parameter growth
- No parameter sharing
- Cannot generalize patterns

Feedforward Neural LMs:

- Still fixed context size
- No state between predictions
- Cannot handle variable length
- Positional parameters wasteful

What We Need:

- **Unbounded context** in principle
- **Parameter efficiency** through sharing
- **State maintenance** across predictions
- **Sequential inductive bias**

Core Concept: Hidden State as Memory

The Hidden State h_t:

- Summarizes history up to time t
- Updated at each time step
- Passed to next time step
- Used for prediction

Conceptual View:

Time	Input	Hidden State Contains
$t = 1$	"The"	{start-of-sentence}
$t = 2$	"cat"	{definite article + subject}
$t = 3$	"sat"	{subject + past action}
$t = 4$	"on"	{complete subject-verb phrase}

Key Property: h_t is a **learned representation** of relevant history

Core Concept: Parameter Sharing Across Time

Same Weights Everywhere:

- W_{xh}: Input-to-hidden (same at all times)
- W_{hh}: Hidden-to-hidden (same at all times)
- W_{hy}: Hidden-to-output (same at all times)

Benefits:

1. **Generalization**: Pattern learned at position 5 works at position 50
2. **Efficiency**: $O(H^2)$ parameters, not $O(TH^2)$
3. **Inductive Bias**: Assumes time-invariant dynamics

Example: Learning "not" negation

- Sees: "I do **not** like..."
- Learns: "not" $\rightarrow$ negation
- Generalizes: "They will **not** come..." (different position)

Mathematics: RNN Forward Pass

Core RNN Equations:

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h) \tag{1}$$
$$y_t = W_{hy} h_t + b_y \tag{2}$$
$$\hat{p}_t = \text{softmax}(y_t) \tag{3}$$

Where:

- $x_t \in \mathbb{R}^{|V|}$: One-hot input at time t
- $h_t \in \mathbb{R}^{H}$: Hidden state (typically $H = 128 - 512$)
- $y_t \in \mathbb{R}^{|V|}$: Output scores
- $\hat{p}_t$: Probability distribution over vocabulary

Initial State: $h_0 = \vec{0}$ or learned parameter

Hidden State Evolution:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

Unrolling the Recursion:

$$h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1 + b_h) \tag{4}$$
$$h_2 = \tanh(W_{hh}h_1 + W_{xh}x_2 + b_h) \tag{5}$$
$$h_3 = \tanh(W_{hh}h_2 + W_{xh}x_3 + b_h) \tag{6}$$

Key Insight: h_t depends on **entire history** $\{x_1, ..., x_t\}$
Information Flow:

- Previous state: $W_{hh}h_{t-1}$ (memory)
- Current input: $W_{xh}x_t$ (new information)
- Nonlinearity: tanh (enables complex functions)

Mathematics: Backpropagation Through Time

Loss Function:

$$L = -\sum_{t=1}^{T} \log p(w_t | w_{<t}) = -\sum_{t=1}^{T} \log \hat{p}_t[w_t]$$

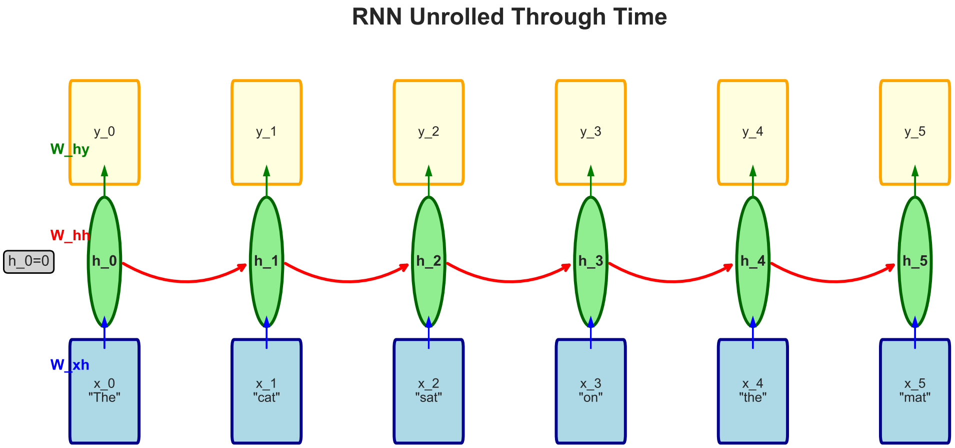
BPTT Algorithm:

1. Forward pass: Compute all h_t and $\hat{p}_t$
2. Compute loss at each time step
3. Backward pass: Accumulate gradients backwards
4. Update weights using total gradient

Gradient Flow:

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_{hh}}$$

Challenge: Gradients pass through many tanh layers!



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

Intuition: Why Gradients Vanish

The Vanishing Gradient Problem:
During backpropagation:

$$\frac{\partial h_t}{\partial h_{t-k}} = \prod_{i=1}^{k} W_{hh}^T \cdot \text{diag}(\tanh'(h_{t-i+1}))$$

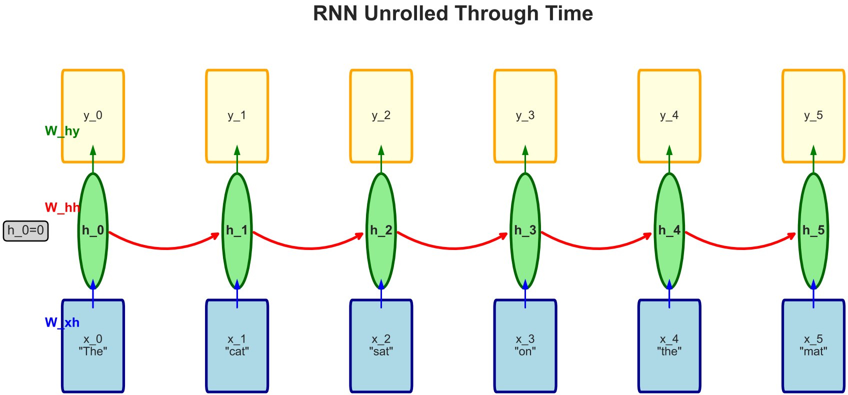
Why Gradients Vanish:

1. $\tanh'(x) \in [0, 1]$ (derivative bounded)
2. Multiple by values < 1 repeatedly
3. Exponential decay: $(0.9)^{10} = 0.35$, $(0.9)^{100} = 0.000027$
4. Long-term dependencies get no gradient!

Consequences:

- RNN "forgets" after 5-10 steps
- Cannot learn long-range patterns
- Biased toward recent context

From Recursion to Computation Graph:



$$\boxed{h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)}$$

Same weights (W_xh, W_hh, W_hy) used at every time step!

Implementation: Training with BPTT

Truncated BPTT (Practical Approach):

1. Process sequence in chunks (e.g., 35 tokens)
2. Carry hidden state forward
3. Only backpropagate within chunk
4. Approximates full BPTT efficiently

Pseudocode:

Input: Sequence $X = (x_1, ..., x_T)$, chunk_size K
$h_0 \leftarrow$ initialize();
for $i = 0$ *to* $T/K - 1$ **do**
  chunk $\leftarrow X[i \cdot K : (i+1) \cdot K]$;
  $h_{i \cdot K}, losses \leftarrow$ forward($chunk, h_{i \cdot K}$);
  gradients $\leftarrow$ backward($losses$);
  update_weights(gradients);
  $h_{(i+1) \cdot K} \leftarrow h_{i \cdot K}$.detach();
end

Implementation: Handling Variable Lengths

Challenge: Sequences have different lengths

Solutions:

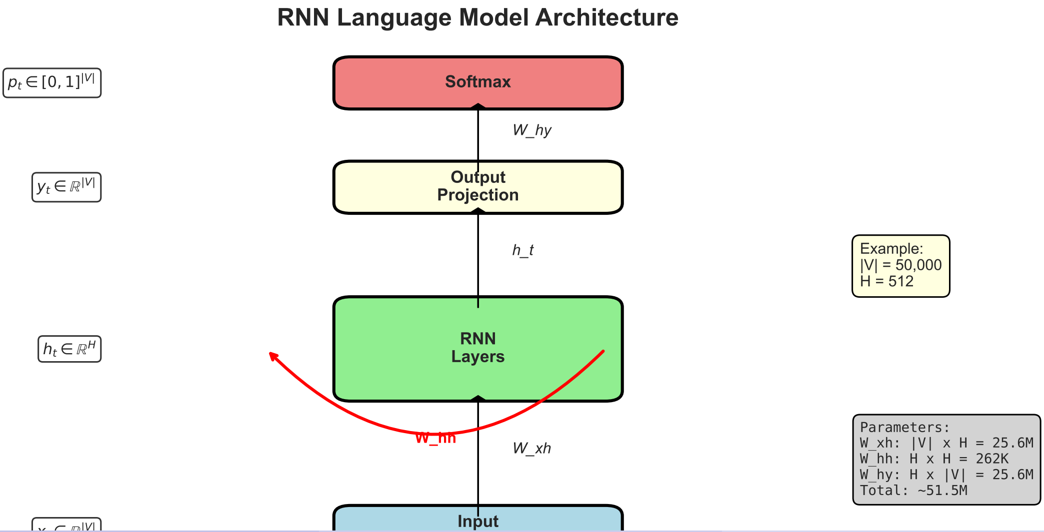
1. **Padding**: Add special ¡PAD¿ tokens
2. **Masking**: Ignore padded positions in loss
3. **Packing**: Process efficiently in batches
4. **Dynamic Batching**: Group similar lengths

Example Batch:

- "The cat sat"
- "A very long sentence with many words"
- "Short"

After Padding (max_len=7):

- "The cat sat `<PAD>` `<PAD>` `<PAD>` `<PAD>`"
- "A very long sentence with many words"
- "Short `<PAD>` `<PAD>` `<PAD>` `<PAD>` `<PAD>` `<PAD>`"


RNN Language Model Architecture
$p_t \in [0,1]^{|V|}$
Softmax
W_hy
$y_t \in \mathbb{R}^{|V|}$
Output
Projection
h_t
Example:
|V| = 50,000
H = 512
$h_t \in \mathbb{R}^H$
RNN
Layers
W_hh
W_xh
Parameters:
W_xh: |V| x H = 25.6M
W_hh: H x H = 262K
W_hy: H x |V| = 25.6M
Total: ~51.5M
Input

Complexity Analysis

Time Complexity:

- Forward pass: $O(T \cdot H^2)$ where T = sequence length, H = hidden size
- Backward pass: $O(T \cdot H^2)$ (same as forward)
- Total per sequence: $O(T \cdot H^2)$

Space Complexity:

- Parameters: $O(H^2 + H \cdot |V|)$
- Activations: $O(T \cdot H)$ (must store all hidden states)
- Gradients: $O(H^2 + H \cdot |V|)$

Comparison:

Model	Parameters	Computation
N-gram	$O(\|V\|^n)$	$O(1)$
Feedforward	$O(n \cdot H + H \cdot \|V\|)$	$O(H^2)$
RNN	$O(H^2 + H \cdot \|V\|)$	$O(T \cdot H^2)$

Application: Text Generation

Character-Level RNN Example:

- Train on Shakespeare
- Generate character-by-character
- Learns spelling, words, grammar, style!

Sample Output:

> *"KING LEAR:*
> *Thou hast been a knave's mind in the world,*
> *And therefore I have seen the day of the death*
> *That thou hast speak to me."*

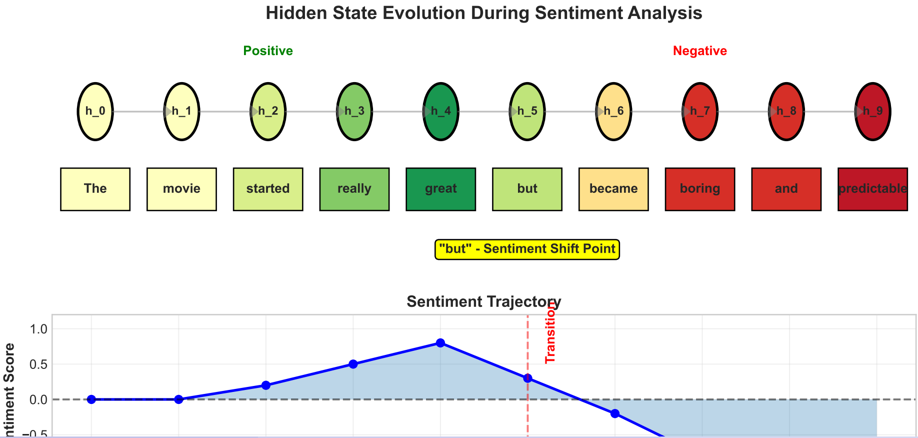
What RNN Learned:

- Character sequences forming words
- Word boundaries and punctuation
- Grammar patterns
- Shakespeare's style

Sequential Sentiment Modeling:

- Input: "The movie started great but became boring"
- RNN tracks sentiment evolution
- Final hidden state → classification

Hidden State Evolution:



Application: Named Entity Recognition

Sequential Labeling with RNNs:
Input: "Apple Inc. was founded by Steve Jobs"

Word:	Apple	Inc.	was	founded	by	Steve	Jobs
Label:	B-ORG	I-ORG	O	O	O	B-PER	I-PER

Why RNNs Excel:

- Context determines entity type
- "Apple" could be fruit or company
- RNN uses surrounding words
- Maintains entity boundaries

BiRNN Enhancement:

- Forward RNN: left context
- Backward RNN: right context
- Combine for full context

Case Study: Early Dialogue Systems

RNN-based Chatbots (circa 2015):

Architecture:

- Encoder RNN: Process input
- Decoder RNN: Generate response
- Hidden state bridges them
- Trained on conversation pairs

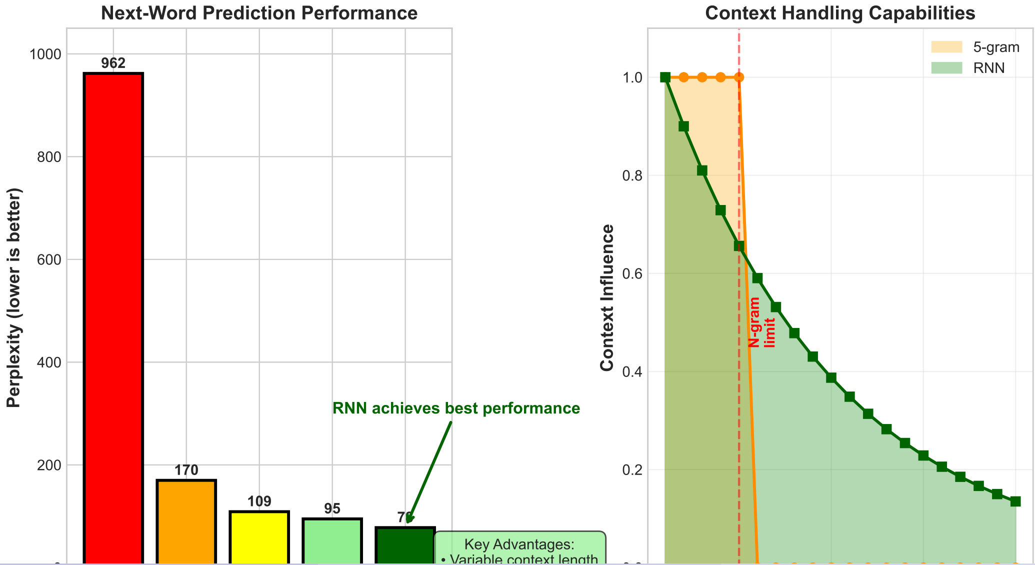
Limitations Discovered:

- Poor long conversation memory
- Generic responses
- No true understanding
- Gradient vanishing limits context

Example Dialogue:

- Human: "How are you?"
- Bot: "I'm doing well, thanks!"
- Human: "What's the weather?"
- Bot: "I don't have access to that."

Performance: RNN vs Previous Methods



Key Takeaways

1. **Sequential Memory**
   - RNNs maintain hidden state across time
   - Theoretically unlimited context
2. **Parameter Sharing**
   - Same weights used at all time steps
   - Enables generalization across positions
3. **Gradient Challenges**
   - Vanishing gradients limit effective context
   - Typically 5-10 words in practice
4. **Natural for Language**
   - Processes text left-to-right
   - Hidden state summarizes context

Model Comparison: Evolution of Context

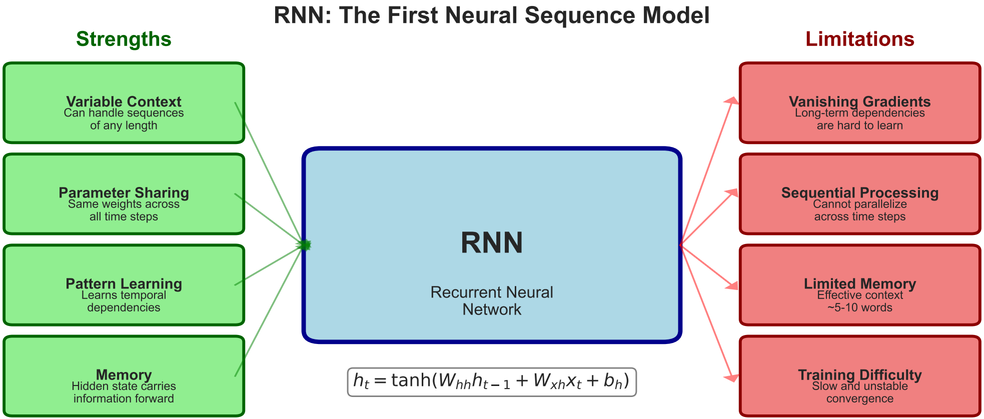
Model	Context	Parameters	Gradient Flow
N-gram	Fixed ($n-1$)	Exponential in n	N/A
Feedforward	Fixed window	Linear in window	Direct
RNN	Unlimited*	Constant	Through time

***In Practice:**

- Theoretical: Unlimited context
- Practical: 5-10 words due to gradient vanishing
- Still better than fixed window
- Motivated next innovation: LSTM/GRU

Key Trade-off: Memory vs Gradient Flow

Visual Summary: RNN Capabilities



Next Week: LSTM/GRU - Solving the vanishing gradient problem

Further Reading

Foundational Papers:

- Elman (1990): "Finding Structure in Time"
- Bengio et al. (1994): "Learning Long-term Dependencies with Gradient Descent is Difficult"
- Mikolov et al. (2010): "Recurrent Neural Network based Language Model"

Practical Resources:

- Karpathy (2015): "The Unreasonable Effectiveness of RNNs"
- Olah (2015): "Understanding LSTM Networks" (blog)
- PyTorch/TensorFlow RNN tutorials

Key Insight: RNNs opened the door, but gradients held them back

Next Week: Solving the Memory Problem

LSTM/GRU - Engineering Better Memory:

- How do we maintain gradients over 100+ steps?
- What are "gates" and how do they help?
- Can we learn what to remember and forget?

Preview:

- LSTM: 4 gates to control information flow
- GRU: Simplified but effective variant
- Gradient highways for long-range learning
- State-of-the-art until attention mechanisms

The Journey: N-grams → Neural → RNN → LSTM → Transformers