

Recurrent Neural Networks

Discovery-Based Introduction to Sequential Learning

Imagine You're Building a Text Prediction System

Your Challenge:

Given text: "The weather today is very"

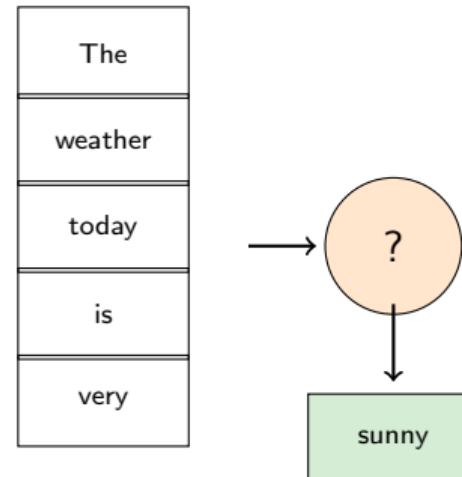
Your system must predict: "sunny" or "nice"

Design Questions:

- ① How much previous text should you look at?
- ② What if the text is 100 words long?
- ③ How do you remember important words from earlier?
- ④ What makes "weather" more important than "is"?

Key Insight: To predict well, you need to *remember* what came before. But how much? And for how long?

Your Design:



What should go in the "?" box?

Next slide reveals: What neural networks are actually trying to do

What Is This About?

The Core Task: Predict the Next Word

The Prediction Challenge:

Example 1: Short Context

Input: "My name is"

Prediction: "**John**" or "**Sarah**"

(Need to remember: 2-3 words)

Example 2: Medium Context

Input: "The cat that ate the fish"

Prediction: "**was**" (singular verb!)

(Need to remember: "cat" is singular, 5 words back)

Example 3: Long Context

Input: "In 1492, Columbus sailed across the Atlantic Ocean and discovered America. This voyage changed"

Prediction: "**history**" or "**everything**"

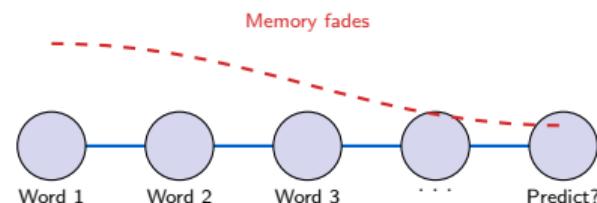
(Need to remember: entire story, 15+ words)

Intuition: Why This Matters

If we can predict the next word:

- **Autocomplete:** "Did you mean...?"
- **Translation:** Convert word-by-word
- **Chatbots:** Generate responses
- **Search:** Understand queries

The Memory Problem:



Question: How far back can we remember?

Why Context Matters: Real Examples

Example 1: Grammar Rules

"The **cat** is sleeping"

→ Predict: "purring" (cat action)

"The **dog** is sleeping"

→ Predict: "snoring" (dog action)

Insight: Need to remember the subject!

Example 2: Names

"J" → "o" (could be "Jo")

"Jo" → "h" (building "Joh")

"Joh" → "n" (common: "John")

Insight: Each prediction depends on ALL previous characters!

Example 3: Sentiment

"This movie is" → "amazing"

"This movie is not" → "amazing"

Insight: One word ("not") flips entire meaning!

Sequential Patterns Everywhere:

Where Are RNNs Used? Real-World Applications



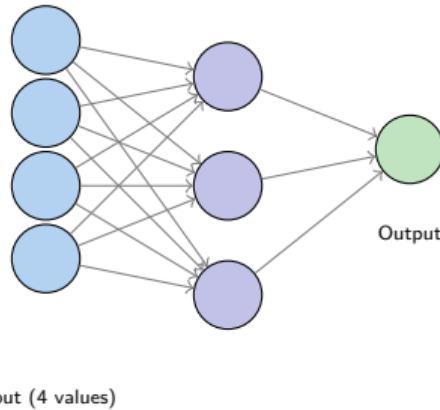
Checkpoint: Understanding

Question: What do all these examples have in common?

Answer: The *order* matters! You cannot shuffle the words/letters/frames and get the same meaning.

Traditional Neural Networks: The Fixed-Size Problem

How Traditional Networks Work:



Key Limitation:

- Fixed input size (e.g., 4 numbers)
- No memory of previous inputs
- Each prediction independent

Concrete Example:

Input: [0.2, 0.5, 0.1, 0.8] → Output: 0.6

Input: [0.3, 0.4, 0.2, 0.7] → Output: 0.5

Worked Example: Text Prediction FAILS

Task: Predict next word after "The cat is"

Attempt 1: Use last 2 words only

Input: "cat is" → Predict: "sleeping"?

Problem: What if sentence was "The dog and the cat is"?
Now we need 5 words!

Attempt 2: Use last 10 words

Input: "----- The cat is"

(Pad with blanks if sentence is shorter)

Problems:

- What if sentence is 50 words long?
- Wasted space for short sentences
- Still a fixed limit!

Real World: The Real Problem

Natural language has **no fixed length**:

- "Hi" (1 word)
- Entire novels (100,000+ words)

The Long-Range Dependency Challenge

Challenge: Distant Information Matters

Example 1: Subject-Verb Agreement

"The **cat** that ate the fish **was** hungry"

To predict "was" (singular), need to remember "cat" from 5 words ago

Example 2: Pronoun Resolution

"Sarah went to the store. She bought milk."

"She" refers to "Sarah" from previous sentence

Example 3: Story Coherence

"In 1492, Columbus discovered America."

(50 words of other content)

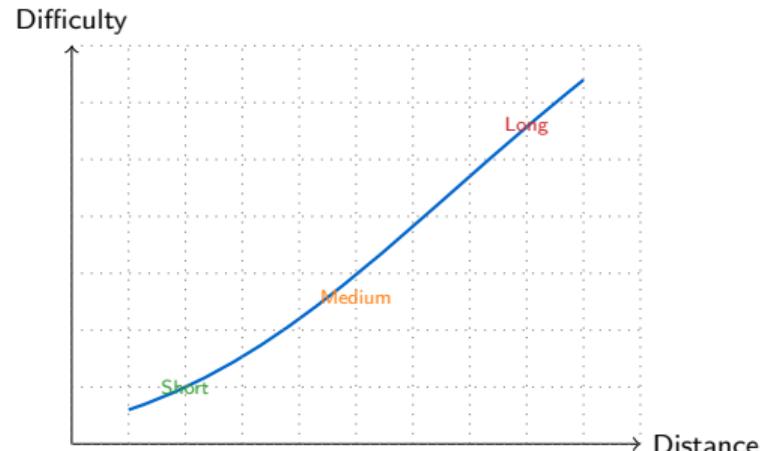
"This **voyage** changed history."

"Voyage" connects to "1492" 50+ words earlier

The Distance Problem:

- Near: 1-5 words back (**Easy**)
- Medium: 5-20 words back (**Harder**)
- Far: 20+ words back (**Very Hard**)

Quantifying Memory Needs:



Real Statistics:

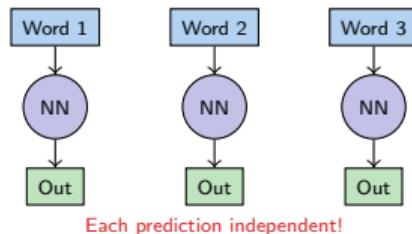
- Average sentence: 15-20 words
- Paragraph: 50-100 words
- Document: 1000+ words

The Key Insight: Memory Flowing Through Time

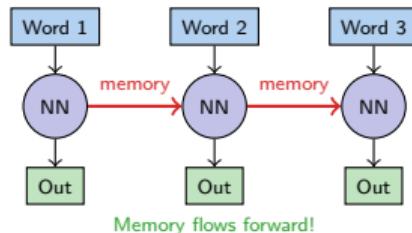
The Breakthrough Idea:

What if the network could *pass information to itself*?

Traditional Network:



Recurrent Network:



Key Property: Same network, reused at each time step,
but with memory!

Intuition: Think of It Like This

Traditional Network: Goldfish memory

- Sees word
- Makes prediction
- Forgets everything
- Sees next word (no context!)

Recurrent Network: Taking notes

- Sees word
- Checks previous notes (memory)
- Updates notes with new info
- Makes prediction with full context
- Passes notes to next step

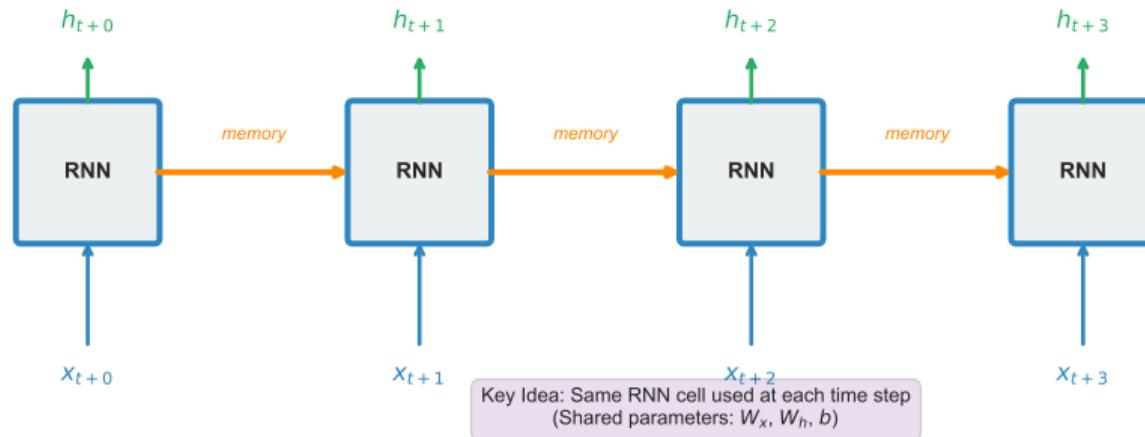
The Memory Vector:

Hidden state h_t = "notes" at time t



RNN Architecture: How It Works

RNN Unrolled Through Time



The RNN Equations:

At each time step t :

1. Update hidden state:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$

2. Produce output:

Concrete Numerical Example:

Predict next character after "ca"

Step 1: Process "c"

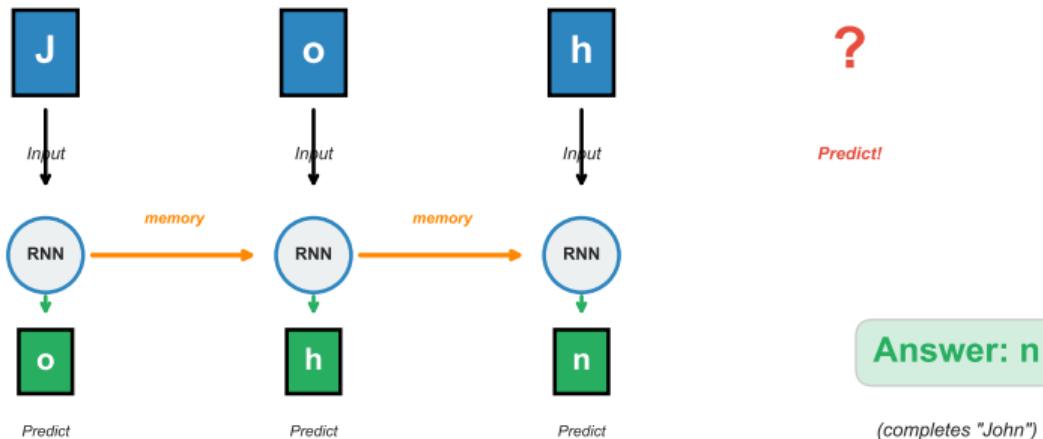
$$x_1 = [1, 0, 0] \text{ (one-hot encoding for "c")}$$

$$h_0 = [0, 0] \text{ (initial memory: zeros)}$$

$$h_1 = \tanh(W_x x_1 + W_h h_0 + b) = [0.5, -0.2]$$

Worked Example: Name Prediction

Example: Predicting Names Character by Character



Task: Given “Joh”, predict “n”

Step-by-Step RNN Process:

Time 1: See “J”

Input: “J” $\rightarrow h_1 = [0.2, 0.1, -0.1]$

Prediction: “o” (70%) “h” (20%) “?” (10%)

What's Happening:

- ① RNN sees “J” \rightarrow starts building context
- ② RNN sees “o” \rightarrow narrows to “Jo-” names
- ③ RNN sees “h” \rightarrow high confidence “John”

Success! RNNs Work for Short Sequences

Task: Sentiment Analysis

Classify movie review as positive/negative

Example Sentence:

"This movie is absolutely fantastic!"

RNN Processing:

- ① "This" → $h_1 = [0.1, 0.0, \dots]$
- ② "movie" → $h_2 = [0.2, 0.1, \dots]$
- ③ "is" → $h_3 = [0.2, 0.1, \dots]$
- ④ "absolutely" → $h_4 = [0.3, 0.5, \dots]$
- ⑤ "fantastic" → $h_5 = [0.8, 0.9, \dots]$

Final hidden state h_5 contains sentiment info

Classifier: sentiment = sigmoid($W \cdot h_5$)

Output: **Positive (98%)**

Why It Works:

- Sentence is short (5 words)
- Strong sentiment words ("fantastic")
- RNN captures entire meaning in h_5

More Success Stories:

1. Stock Price Prediction (short term)

Input: Last 10 days of prices

Output: Tomorrow's price

Accuracy: **85%**

2. Name Completion

Input: "Sar"

Output: "ah" (Sarah)

Accuracy: **92%**

3. Short Text Generation

Input: "The weather is"

Output: "very nice today"

Accuracy: **88%**

Real World: RNNs in Production

By 2010-2015, RNNs were used for:

- Speech recognition (Siri, Alexa)
- Simple chatbots
- Auto-complete in keyboards

But Then... The Failure Pattern Emerges

The Long Sentence Test:

"The **cat** that ate the fish and slept on the couch all day was hungry."

Expected: Predict "was" (singular verb matching "cat")

RNN Prediction: "were" (plural - WRONG!)

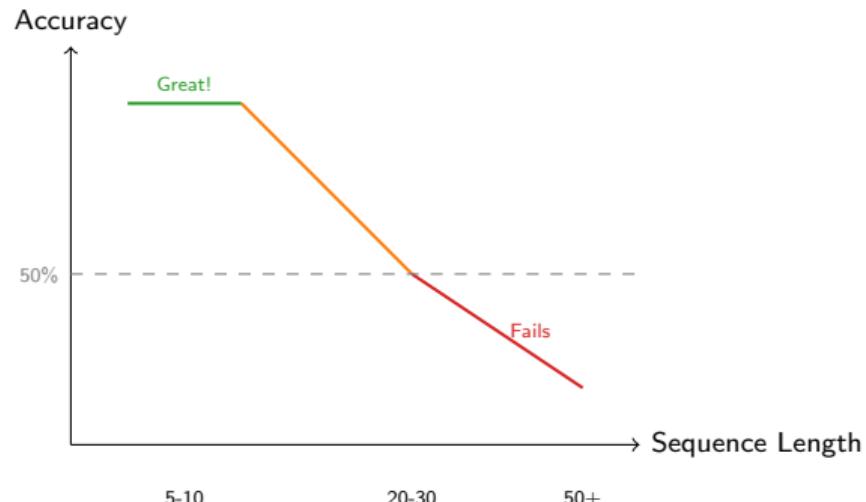
What Happened?

- ① RNN sees "cat" at position 2
- ② Processes 8 more words
- ③ By position 11 ("was"), has forgotten "cat" was singular!
- ④ Remembers "fish" and "couch" (plural-sounding)
- ⑤ Predicts wrong verb form

More Failures:

- Long paragraphs: Forgets topic
- Stories: Loses plot thread
- Conversations: Forgets context

The Pattern:



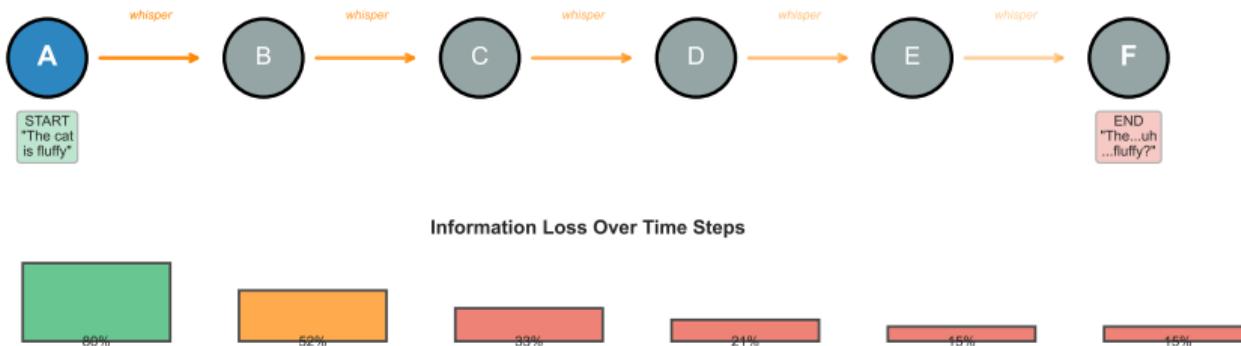
Measured Performance:

Sequence Length	Accuracy
5-10 words	90%
15-20 words	70%
30+ words	45%

Diagnosis: The Vanishing Gradient Problem

The Memory Problem: Telephone Game Analogy

Problem: Earlier information gets WEAKER as it travels through time
Solution: LSTM uses gates to preserve important information



The Telephone Game Analogy:

- ① Person 1: "The blue cat is sleeping"
- ② Person 2: "The lu cat is sleeping"
- ③ Person 3: "The cat is sleeping"
- ④ Person 4: "A cat sleeping"

Why Gradients Vanish:

Problem: Training uses gradient descent
To update early weights, need gradient to flow backward through ALL time steps

Tanh Derivative:

$$\tanh'(x) < 1 \text{ (usually 0.1 to 0.5)}$$

Pause: How Do YOU Remember Long Stories?

Thought Experiment:

You read a 500-word article about Columbus's 1492 voyage.

Next day, someone asks: "What was the article about?"

What do you remember?

- "1492" (important date)
- "Columbus" (main person)
- "Atlantic Ocean" (key location)
- "discovered America" (main event)
- NOT every single word!

Human Memory Strategy:

- ① **Selective:** Keep important info, forget details
- ② **Protected:** Important facts stay in memory
- ③ **Updated:** Add new important info, remove old

You don't remember every word - you remember *what matters!*

RNN vs Human Memory:

	RNN	Human
Selective?	No	Yes
Protected?	No	Yes
Updated?	Sort of	Yes

The Key Insight:

RNN problem: $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$
Everything gets mixed together and fades!

What we need:

- ① **Selective Gate:** Decide what to remember
- ② **Protected Memory:** Keep important info safe
- ③ **Update Control:** Add new info carefully

Intuition: The Hypothesis

What if we add **explicit control** over memory?

- Gate 1: "Should I forget this?"
- Gate 2: "Should I add this new info?"

The Hypothesis: Protect Important Memories

The Problem with RNN:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$

Multiplication at each step → gradient vanishes

The Breakthrough Idea:

What if we use ADDITION instead?

$$C_t = C_{t-1} + \text{new info}$$

Addition preserves gradients → memory lasts!

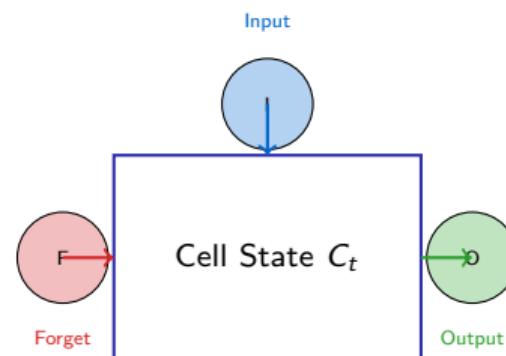
But we need control:

- **What to forget:** Remove outdated info
- **What to add:** Add new important info
- **What to output:** Show relevant info

Solution: Three “gates” (valves) controlling memory flow!

Conceptual Design:

Imagine a protected memory cell C_t :

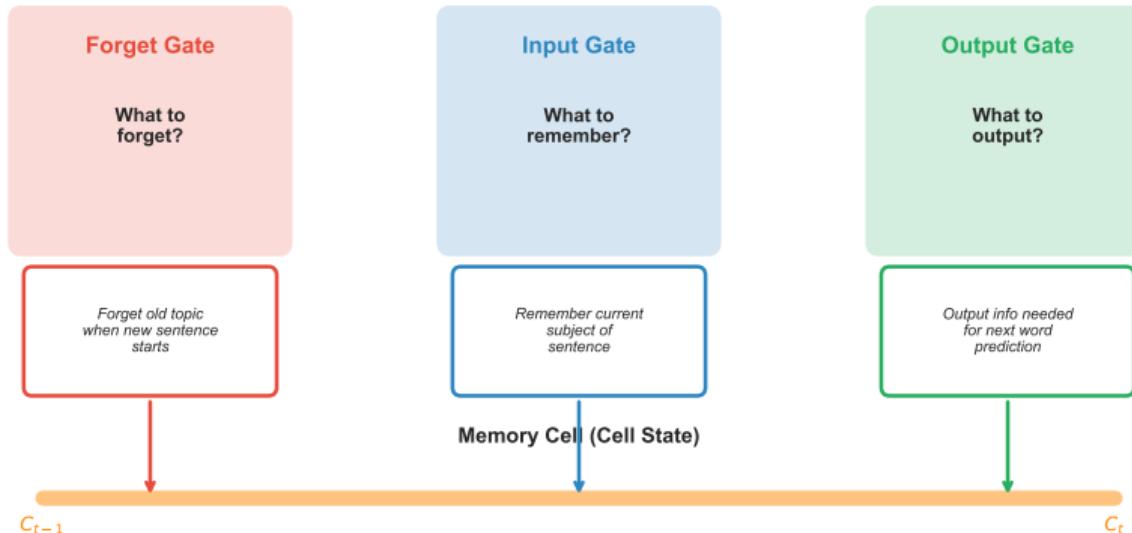


How Gates Work:

- ➊ **Forget Gate:** “Remove outdated info”
 $f_t = \sigma(\dots)$ (0 = forget all, 1 = keep all)
- ➋ **Input Gate:** “Add new important info”
 $i_t = \sigma(\dots)$ (0 = ignore, 1 = add fully)
- ➌ **Output Gate:** “Show relevant info”
 $o_t = \sigma(\dots)$ (0 = hide, 1 = show)

Preview: LSTM (Long Short-Term Memory)

LSTM Solution: Three Smart Gates



The LSTM Solution:

Two memory streams:

Checkpoint: Breakthrough Comparison

What You Learned Today (RNN):

- ① **The Challenge:** Variable-length sequences
- ② **RNN Insight:** Memory through recurrence
- ③ **RNN Success:** Works for short sequences (5-10)
- ④ **RNN Failure:** Vanishing gradient kills long memory
- ⑤ **Root Cause:** Repeated multiplication
- ⑥ **Human Insight:** Selective, protected memory
- ⑦ **LSTM Idea:** Gates + Addition = Long memory

Real World: Where RNNs Still Used

Despite limitations, RNNs work well for:

- **Short sequences:** 5-15 steps
- **Real-time audio:** Low latency needed
- **Simple patterns:** Limited context required
- **Embedded systems:** Memory constraints

What's Next (LSTM Slides):

- ① **Water Tank Analogy:** Concrete intuition for gates
- ② **Gate Mechanics:** How forget/input/output gates work
- ③ **Worked Examples:** Step-by-step LSTM processing
- ④ **Vector Mathematics:** Full equations explained
- ⑤ **Training Details:** BPTT through LSTM
- ⑥ **Applications:** Where LSTMs excel
- ⑦ **Implementation:** PyTorch code

Intuition: The Learning Journey

Today: Discovered WHY we need better memory (Vanishing gradient problem)

LSTM Slides: Learn HOW gates solve it (Discovery-based approach)

Lab: Build your own LSTM (Hands-on implementation)

Where RNNs Work Well Today

Successful Applications:

1. Short-Sequence Tasks

- **Name prediction:** “Joh” → “n”
- **Next character:** Autocomplete (5-10 chars)
- **Sentiment:** Short reviews (1-2 sentences)
- **Simple classification:** Fixed-length input

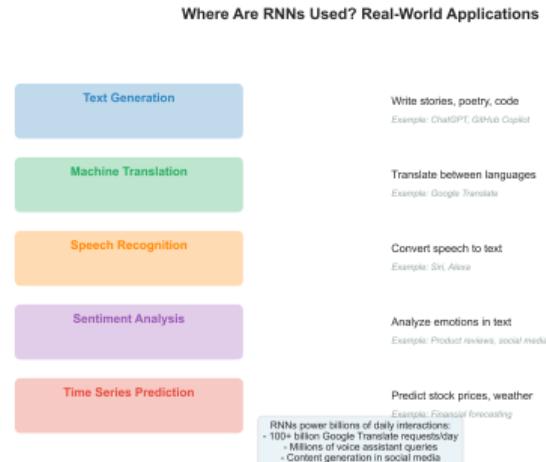
2. Real-Time Systems

- **Voice recognition:** Process audio frame-by-frame
- **Gesture detection:** Short movement sequences
- **Sensor data:** Last 10 readings for prediction

3. When Speed Matters

- RNN: Simpler, faster than LSTM
- Fewer parameters to train
- Good for resource-constrained devices

Application Comparison:



Limitations Summary:

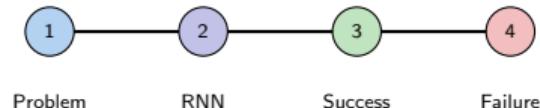
Task	Max Length
Character prediction	10-15
Sentiment (sentence)	15-20
Short translation	10-15
Paragraph understanding	FAILS

Summary: What We Learned About RNNs

Key Concepts:

- ① **Sequential Data:** Order matters (text, speech, time series)
- ② **Traditional Networks Fail:** Fixed-size input, no memory
- ③ **RNN Innovation:** Recurrent connections create memory
$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$
- ④ **Hidden State:** Carries context forward through time
- ⑤ **Success:** Works well for short sequences (5-20 steps)
- ⑥ **Failure:** Vanishing gradient for long sequences
- ⑦ **Root Cause:** Repeated multiplication fades gradients
gradient $\propto 0.5^{20} \approx 0$
- ⑧ **Solution Preview:** LSTM uses addition + gates

The Discovery Journey:



Worked Examples Covered:

- **Name prediction:** “Joh” → “n”
- **Sentiment analysis:** Movie reviews
- **Long sentences:** Subject-verb agreement fails

Mathematical Insights:

- Recurrence creates memory
- Tanh activation bounds values
- Backpropagation through time (BPTT)
- Gradient vanishing from repeated multiplication

Checkpoint: Final Check

Next Steps: From RNN to LSTM

Learning Path:

① Today (RNN): [Completed]

- Understand sequential data challenge
- Learn RNN architecture
- Discover vanishing gradient problem

② Next (LSTM): Continue reading

- Discovery-based water tank analogy
- Learn three gates (forget, input, output)
- See how addition solves vanishing gradient
- Worked examples with actual numbers

③ Lab Session: Hands-on implementation

- Build RNN from scratch
- Implement LSTM
- Train on real text data
- Compare performance

④ Week 4 (Seq2seq): Advanced architectures

- Encoder-decoder models
- Attention mechanism
- Machine translation

Resources:

Slides:

- LSTM slides (next in sequence)
- Week 4: Seq2seq models
- Week 5: Transformers

Labs:

- Week 3 Lab: RNN/LSTM implementation
- Shakespeare generation notebook
- Sentiment classification exercise

Code Examples:

- PyTorch RNN tutorial
- Character-level generation
- Name prediction model

Key Questions for Next Session:

- ① How do gates actually work? (Sigmoid functions)
- ② Why does addition preserve gradients? (Math proof)