Recurrent Neural Networks (RNNs)
Understanding Sequential Data Processing

Why Do We Need RNNs?

Traditional Neural Networks:

- Process fixed-size inputs
- No memory of previous inputs
- Each prediction is independent

Problem: Many tasks involve sequences!

- Text: Words depend on previous words
- Speech: Sounds form words over time
- Video: Frames are connected

> **Intuition: Key Idea**
>
> We need networks that can "remember" previous inputs to understand context.

Examples of Sequential Data:

1. **Names:** "Joh" $\rightarrow$ "n"
2. **Sentences:** "The cat is ..."
3. **Stock prices:** Yesterday $\rightarrow$ Today
4. **Weather:** Past week $\rightarrow$ Tomorrow

Where Are RNNs Used? Real-World Applications

|	
Text Generation	Write stories, poetry, code *Example: ChatGPT, GitHub Copilot*
Machine Translation	Translate between languages *Example: Google Translate*
Speech Recognition	Convert speech to text *Example: Siri, Alexa*
Sentiment Analysis	Analyze emotions in text *Example: Product reviews, social media*
Time Series Prediction	Predict stock prices, weather *Example: Financial forecasting*

RNNs power billions of daily interactions:
- 100+ billion Google Translate requests/day
- Millions of voice assistant queries
- Content generation in social media

RNN Unrolled Through Time

Key Idea: Same RNN cell used at each time step
(Shared parameters: W_x, W_h, b)
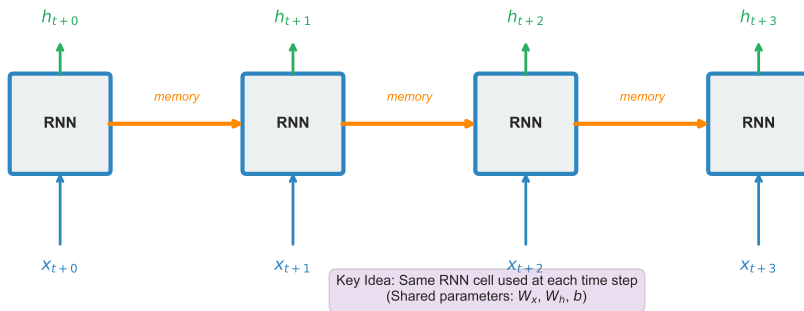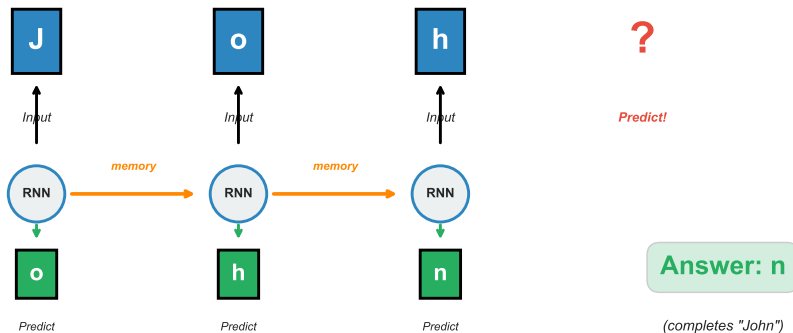
How RNNs Work:

1. Take current input x_t
2. Combine with previous memory h_{t-1}
3. Produce output h_t
4. Pass memory to next step

Key Innovation:

- Same weights at each time step
- Memory flows through time
- Can process any sequence length

Example: Predicting Names Character by Character

Task: Given "Joh", predict "n"

RNN Process:

RNN Equations:

Hidden state update:

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

Output:

$$y_t = \text{softmax}(W_y h_t + b_y)$$

What each part means:

- x_t: Current input
- h_{t-1}: Previous memory
- W_x, W_h: Weight matrices
- tanh: Activation function (-1 to 1)
- y_t: Output prediction

Intuition: Think of it like

A student taking notes:
1. Read new information (x_t)
2. Check your notes (h_{t-1})
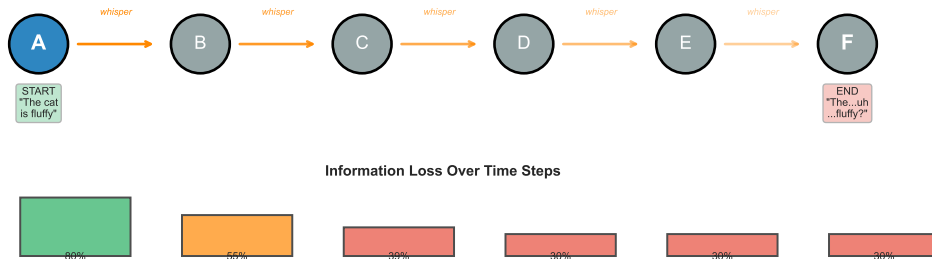3. Update your notes (h_t)
4. Answer question (y_t)

Key Parameters:

- Input size: 100-300
- Hidden size: 128-512
- Layers: 1-3

Problem: Vanishing Gradient (Memory Fading)

The Memory Problem: Telephone Game Analogy

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

Information Loss Over Time Steps

The Telephone Game Problem:
- Message gets weaker as it passes
- Early information gets lost
- RNN forgets long-term context

Why This Happens:
- Gradients get multiplied many times
- Small numbers × small numbers → tiny
- Long sequences = many multiplications

Example 2: Sentiment Analysis

Task: Classify movie review as positive/negative

Input Sentence:
"This movie is absolutely fantastic!"

RNN Processing:

1. Process "This" $\to h_1$
2. Process "movie" $\to h_2$
3. Process "is" $\to h_3$
4. Process "absolutely" $\to h_4$
5. Process "fantastic" $\to h_5$

Final hidden state h_5 contains sentiment info

$\to$ Classify: **Positive (98%)**

PyTorch Code Example:

```python
import torch.nn as nn

class SentimentRNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.embedding = nn.Embedding(
            num_embeddings=10000,
            embedding_dim=128
        )
        self.rnn = nn.RNN(
            input_size=128,
            hidden_size=256,
            num_layers=2
        )
        self.fc = nn.Linear(256, 2)

    def forward(self, x):
        # x: [seq_len, batch]
        embedded = self.embedding(x)
        output, hidden = self.rnn(embedded)
        # Use final hidden state
        prediction = self.fc(hidden[-1])
        return prediction
```
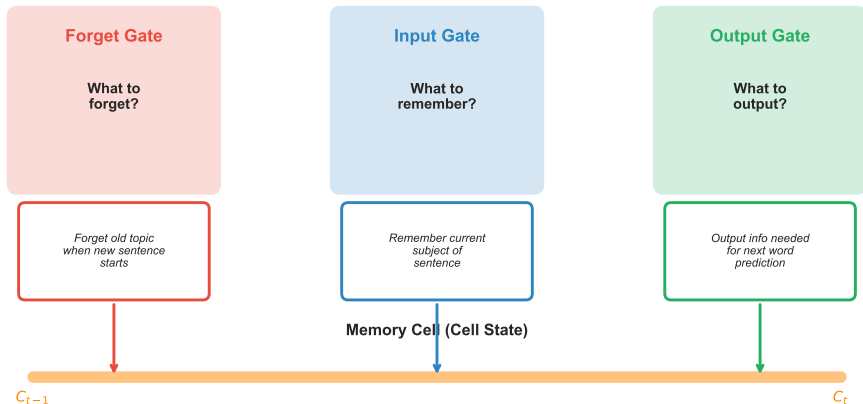
Output: [Negative, Positive]

LSTM Solution: Three Smart Gates

Forget Gate	Input Gate	Output Gate
What to forget?	What to remember?	What to output?
Forget old topic when new sentence starts	*Remember current subject of sentence*	*Output info needed for next word prediction*

Memory Cell (Cell State)

C_{t-1} C_t

Key: Gates control information flow to solve vanishing gradient problem

Example 3: Text Generation

Task: Generate text character by character

Training Data: Shakespeare's works

How it works:

1. Start with seed text: "To be"
2. LSTM predicts next character: " "
3. Add to sequence: "To be "
4. Predict next: "o"
5. Continue: "To be or"
6. Keep generating. . .

Generated Output:

"To be or not to be, that is the question"

LSTM learned:

- English grammar
- Shakespeare's style
- Common phrases

Simple Generation Code:

```python
import torch
import torch.nn as nn

class CharRNN(nn.Module):
    def __init__(self, vocab_size):
        super().__init__()
        self.lstm = nn.LSTM(
            input_size=vocab_size,
            hidden_size=512,
            num_layers=2
        )
        self.fc = nn.Linear(512, vocab_size)

    def generate(self, start_text, length=100):
        hidden = None
        chars = [start_text]

        for _ in range(length):
            # Encode current char
            x = encode(chars[-1])
            # LSTM prediction
            out, hidden = self.lstm(x, hidden)
            # Sample next char
            next_char = sample(self.fc(out))
            chars.append(next_char)

        return ''.join(chars)

# Generate text
model.generate("To␣be", length=50)
```

Summary: What We Learned

Key Concepts:

1. **Sequential Data:** Many tasks involve sequences
2. **RNN:** Networks with memory
3. **Hidden State:** Carries information forward
4. **Vanishing Gradient:** Memory fades over time
5. **LSTM:** Uses gates to preserve memory

Three Examples We Saw:

- Name prediction (supervised)
- Sentiment analysis (classification)
- Text generation (autoregressive)

Real World: Where RNNs Are Used

- **ChatGPT:** Text generation
- **Google Translate:** Language translation
- **Siri/Alexa:** Speech recognition
- **YouTube:** Video captioning
- **Gmail:** Smart compose

Next Steps:

- Week 4: Sequence-to-Sequence models
- Week 5: Transformers (attention mechanism)
- Lab: Implement your own LSTM

Questions?