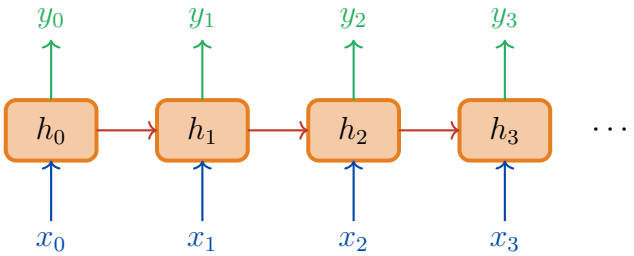
Deep Learning for Perception

Lecture 07: Recurrent Neural Networks



RNN: Processing Sequential Data Through Time

Topics Covered in This Lecture:

- Sequential Data
- RNN Architecture
- RNN Forward Pass
- Backpropagation Through Time
- Vanishing Gradients Problem
- LSTM (Long Short-Term Memory)
- GRU (Gated Recurrent Unit)
- RNN Applications

Instructor: Aqsa Younas

Department of Computer Science
FAST-NU, CFD Campus

Contents

Advance Organizer — What You'll Learn

Learning Objectives: By the end of this lecture, you will be able to:

1. **Explain** why sequential data requires special architectures
2. **Calculate** RNN forward pass step-by-step
3. **Describe** the vanishing gradient problem in RNNs
4. **Compute** LSTM gate operations
5. **Compare** LSTM and GRU architectures
6. **Calculate** parameter counts for RNN variants
7. **Apply** RNNs to sequence modeling problems

Prior Knowledge Required:

- Feedforward neural networks
- Backpropagation
- Activation functions (tanh, sigmoid)

1 Machine Learning for Sequential Data

Why It Matters

Many real-world data types are **sequences** where order matters and elements depend on each other. Standard feedforward networks cannot handle such data effectively.

1.1 What is Sequential Data?

Definition

Sequential Data is data where:

1. Elements occur in a **specific order**
2. Elements **depend on each other** (temporal/contextual dependencies)
3. Sequences can have **variable length**

Examples:

- Text (sentences, documents)
- Speech and audio
- Time series (stock prices, weather, sensor data)
- Video (sequence of frames)
- DNA sequences

1.2 Why Feedforward Networks Fail

Problems with Feedforward Networks for Sequences

1. **Fixed input size:** Cannot handle variable-length sequences
2. **No memory:** Each input processed independently
3. **No parameter sharing:** Weights learned separately for each position
4. **Too many parameters:** Long sequences = huge input layer

1.3 Sequence-to-Sequence Tasks

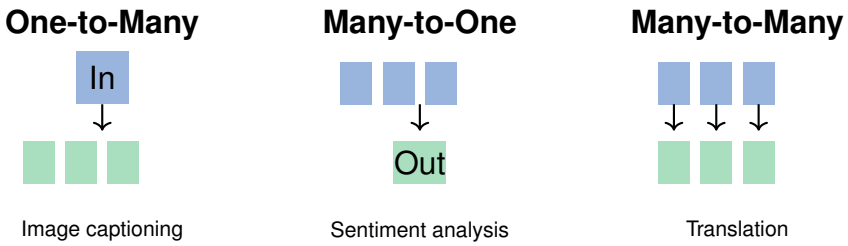


Figure 1: Types of sequence-to-sequence tasks

2 Recurrent Neural Networks (RNNs)

Why It Matters

RNNs introduce the concept of **memory** through recurrent connections. The hidden state carries information from previous time steps, enabling the network to model sequential dependencies.

2.1 RNN Core Idea

Definition

A **Recurrent Neural Network (RNN)** processes sequences by:

1. Maintaining a **hidden state** h_t that captures information about the past
2. At each time step, combining the **current input** x_t with the **previous hidden state** h_{t-1}
3. Using the **same weights** across all time steps (weight sharing)

Key equation:

$$h_t = f(h_{t-1}, x_t)$$

Key Formula

RNN Forward Pass Equations:
Hidden state update:

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

Output:

$$\boxed{y_t = W_{hy} \cdot h_t + b_y}$$

Where:

- $x_t \in \mathbb{R}^d$ = Input at time t
- $h_t \in \mathbb{R}^n$ = Hidden state at time t
- $W_{xh} \in \mathbb{R}^{n \times d}$ = Input-to-hidden weights
- $W_{hh} \in \mathbb{R}^{n \times n}$ = Hidden-to-hidden weights
- $W_{hy} \in \mathbb{R}^{k \times n}$ = Hidden-to-output weights

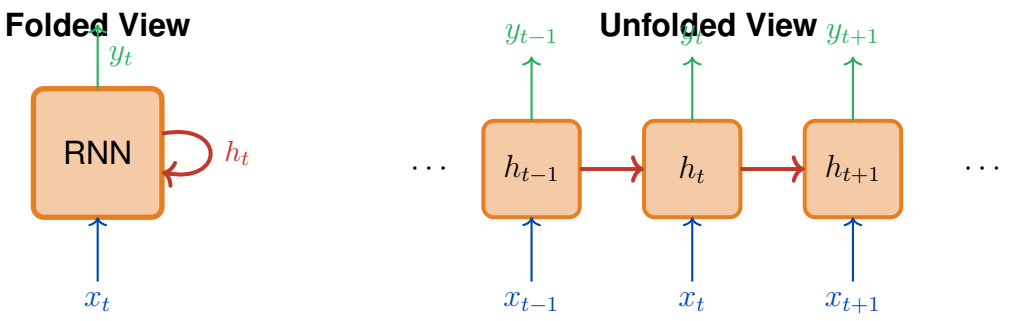


Figure 2: RNN: Folded (left) and Unfolded through time (right)

Analogy — Think of It Like This

Think of RNN as reading a book:
When you read a sentence, you don't start fresh at each word. You carry the **context** (hidden state) from previous words. The word "bank" means something different after "river" vs after "money."
The RNN's hidden state is like your mental context while reading!

2.2 RNN Advantages

Memory Hook — Remember This!

Why RNNs Work for Sequences:

1. **Weight sharing:** Same parameters for all time steps $\rightarrow$ fewer parameters
2. **Memory:** Hidden state captures past information
3. **Variable length:** Can process any sequence length by unfolding more/fewer steps
4. **End-to-end training:** Learn features and predictions jointly

Solved Example 1: RNN Forward Pass

Given Data:
Processing the word "hello" character by character.

- Vocabulary: $\{h = 0, e = 1, l = 2, o = 3\}$ (size 4)
- Hidden dimension: 3
- Initial hidden state: $h_0 = [0, 0, 0]$

Weight matrices:

$$W_{xh} = \begin{bmatrix} 0.5 & 0.1 & -0.2 & 0.3 \\ -0.1 & 0.4 & 0.2 & -0.1 \\ 0.2 & -0.3 & 0.5 & 0.1 \end{bmatrix}, \quad W_{hh} = \begin{bmatrix} 0.1 & -0.2 & 0.1 \\ 0.2 & 0.1 & -0.1 \\ -0.1 & 0.2 & 0.1 \end{bmatrix}$$

Task: Calculate the hidden state after processing 'h' (first character).

Solution:
Step 1: One-hot encode 'h'

$$x_0 = [1, 0, 0, 0]^T \quad \text{(index 0 in vocabulary)}$$

Step 2: Calculate input contribution

$$W_{xh} \cdot x_0 = \begin{bmatrix} 0.5 \\ -0.1 \\ 0.2 \end{bmatrix}$$

Step 3: Calculate recurrent contribution

$$W_{hh} \cdot h_{-1} = W_{hh} \cdot [0, 0, 0]^T = [0, 0, 0]^T$$

Step 4: Sum and apply tanh

$$z_0 = W_{xh} \cdot x_0 + W_{hh} \cdot h_{-1} = [0.5, -0.1, 0.2]^T$$

$$h_0 = \tanh(z_0) = [\tanh(0.5), \tanh(-0.1), \tanh(0.2)]$$

$$h_0 = [0.462, -0.100, 0.197]$$

Answer: $h_0 = [0.462, -0.100, 0.197]$
Continuing for the full sequence "hello":

t	**Char**	h_t
0	'h'	[0.462, -0.100, 0.197]
1	'e'	[0.184, 0.432, -0.333]
2	'l'	[-0.293, 0.303, 0.489]
3	'l'	[-0.237, 0.122, 0.564]
4	'o'	[0.299, -0.189, 0.202]

3 The Vanishing Gradient Problem

Why It Matters

RNNs struggle to learn **long-term dependencies** because gradients shrink exponentially as they flow backward through time. This is the key limitation that LSTM and GRU were designed to solve.

Definition

Vanishing Gradient Problem:
During backpropagation through time (BPTT), gradients are multiplied by the weight matrix at each time step:

$$\frac{\partial L}{\partial h_0} = \frac{\partial L}{\partial h_T} \cdot \prod_{t=1}^{T} \frac{\partial h_t}{\partial h_{t-1}}$$

Since $\frac{\partial h_t}{\partial h_{t-1}} = \text{diag}(\tanh'(z_t)) \cdot W_{hh}$, and $|\tanh'(x)| \leq 1$:

- If $\|W_{hh}\| < 1$: Gradients $\to 0$ (vanishing)
- If $\|W_{hh}\| > 1$: Gradients $\to \infty$ (exploding)

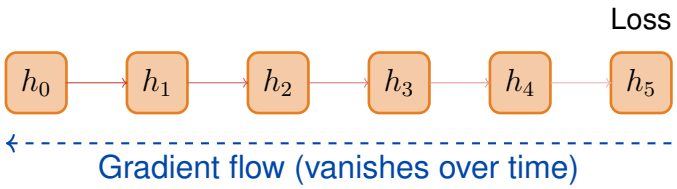


Figure 3: Vanishing gradients: Information from early time steps fades

Analogy — Think of It Like This

Snow Plow Analogy:
Imagine a snow plow clearing a road. As it moves forward, the road behind it starts getting covered with snow again. By the time it's far down the road, the beginning of the path has completely disappeared.
Similarly, in RNNs, information from early time steps "fades away" as the sequence gets longer.

4 Long Short-Term Memory (LSTM)

Why It Matters

LSTM introduces **gates** that control information flow, allowing the network to learn when to remember, forget, and output information. This solves the vanishing gradient problem.

Historical Note

LSTM was introduced by Hochreiter & Schmidhuber in 1997. It remained relatively obscure until the deep learning revolution, when it became the dominant architecture for sequence modeling (2014-2017).

4.1 LSTM Architecture

Definition

LSTM has two state vectors:

1. **Cell state** c_t: Long-term memory (the "conveyor belt")
2. **Hidden state** h_t: Short-term memory / output

Three gates control information flow:

1. **Forget gate** f_t: What to remove from cell state
2. **Input gate** i_t: What new information to add
3. **Output gate** o_t: What to output from cell state

Key Formula

LSTM Equations:
Gates (all use sigmoid σ for values in $[0, 1]$):

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(Forget gate)} \quad (1)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(Input gate)} \quad (2)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(Output gate)} \quad (3)$$

Candidate cell state:

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Cell state update:

$$\boxed{c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t}$$

Hidden state (output):

$$\boxed{h_t = o_t \odot \tanh(c_t)}$$

Where $\odot$ denotes element-wise multiplication.

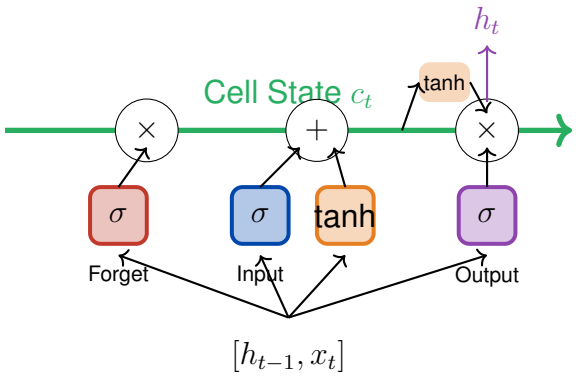


Figure 4: LSTM cell: Gates control information flow through the cell state

Solved Example 2: LSTM Gate Calculation

Given Data:

- Input: $x_t = [0.5, 0.3]$
- Previous hidden state: $h_{t-1} = [0.1, -0.1]$
- Previous cell state: $c_{t-1} = [0.2, 0.3]$
- Concatenated input: $[x_t, h_{t-1}] = [0.5, 0.3, 0.1, -0.1]$

After computing gates (using weight matrices):

- Forget gate: $f_t = [0.542, 0.500]$
- Input gate: $i_t = [0.527, 0.515]$
- Candidate: $\tilde{c}_t = [-0.020, 0.090]$
- Output gate: $o_t = [0.520, 0.527]$

Task: Calculate the new cell state c_t and hidden state h_t.

Solution:

Step 1: Cell state update

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$\begin{aligned} c_t &= [0.542, 0.500] \odot [0.2, 0.3] + [0.527, 0.515] \odot [-0.020, 0.090] \\ &= [0.1084, 0.150] + [-0.0105, 0.0464] \\ &= [0.098, 0.196] \end{aligned}$$

Step 2: Hidden state output

$$h_t = o_t \odot \tanh(c_t)$$

$$\begin{aligned} \tanh(c_t) &= \tanh([0.098, 0.196]) = [0.098, 0.194] \\ h_t &= [0.520, 0.527] \odot [0.098, 0.194] \\ &= [0.051, 0.102] \end{aligned}$$

Answer:

- New cell state: $c_t = [0.098, 0.196]$
- New hidden state: $h_t = [0.051, 0.102]$

Interpretation:

- Forget gate ≈ 0.5: Keeping about half of old memory
- Input gate ≈ 0.5: Adding moderate new information
- Output gate ≈ 0.5: Outputting about half of cell state

5 Gated Recurrent Unit (GRU)

Why It Matters

GRU is a simplified version of LSTM with fewer gates and parameters. It often performs comparably to LSTM while being faster to train.

Definition

GRU simplifies LSTM by:

1. Merging cell state and hidden state into one
2. Using only **two gates** (vs three in LSTM):
   - **Update gate** z_t: Controls how much past to keep
   - **Reset gate** r_t: Controls how much past to forget when computing candidate

Key Formula

GRU Equations:
Gates:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \quad \text{(Update gate)} \tag{4}$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \quad \text{(Reset gate)} \tag{5}$$

Candidate hidden state:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

Hidden state update:

$$\boxed{h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t}$$

Intuition:

- $z_t = 1$: Completely replace old state with new candidate
- $z_t = 0$: Keep old state, ignore new input

Solved Example 3: GRU Calculation

Given Data:

- Input: $x_t = [0.5, 0.3]$
- Previous hidden state: $h_{t-1} = [0.1, -0.1]$

After computing gates:

- Update gate: $z_t = [0.542, 0.500]$
- Reset gate: $r_t = [0.527, 0.515]$
- Candidate: $\tilde{h}_t = [-0.015, 0.109]$

Solution:
Hidden state update:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Step 1: Calculate $(1 - z_t) \odot h_{t-1}$

$$(1 - [0.542, 0.500]) \odot [0.1, -0.1] = [0.458, 0.500] \odot [0.1, -0.1] = [0.046, -0.050]$$

Step 2: Calculate $z_t \odot \tilde{h}_t$

$$[0.542, 0.500] \odot [-0.015, 0.109] = [-0.008, 0.054]$$

Step 3: Sum

$$h_t = [0.046, -0.050] + [-0.008, 0.054] = [0.038, 0.004]$$

Answer: $h_t = [0.038, 0.004]$
The update gate $z \approx 0.5$ means roughly equal mix of old and new information.

5.1 LSTM vs GRU Comparison

Aspect	LSTM	GRU
Gates	3 (forget, input, output)	2 (update, reset)
States	2 (cell state, hidden state)	1 (hidden state only)
Parameters	4× RNN	3× RNN
Training speed	Slower	Faster
Long sequences	Often better	Comparable
When to use	Complex long-term dependencies	Simpler tasks, limited data

Table 1: LSTM vs GRU comparison

Solved Example 4: Parameter Count Comparison

Given Data:

- Input dimension: $d = 100$
- Hidden dimension: $n = 256$

Task: Calculate parameter count for RNN, LSTM, and GRU (one layer, excluding output).

Solution:
RNN Parameters:

- W_{xh}: $d \times n = 100 \times 256 = 25,600$
- W_{hh}: $n \times n = 256 \times 256 = 65,536$
- b_h: $n = 256$

Total RNN: $25,600 + 65,536 + 256 = \mathbf{91,392}$
LSTM Parameters (4 gates, each with W_x, W_h, b):

$$4 \times (d \times n + n \times n + n) = 4 \times 91,392 = \mathbf{365,568}$$

GRU Parameters (3 gates):

$$3 \times (d \times n + n \times n + n) = 3 \times 91,392 = \mathbf{274,176}$$

Parameter Count Summary:

Model	Parameters	Ratio to RNN
RNN	91,392	1.0×
GRU	274,176	3.0×
LSTM	365,568	4.0×

LSTM has 4× the parameters of a basic RNN because it has 4 sets of weights (for 3 gates + candidate).

6 Summary

Key Takeaways

1. Sequential Data

- Order matters, elements depend on each other
- Feedforward networks cannot handle variable-length sequences

2. RNN Core Equations

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

- Hidden state carries memory across time steps
- Same weights shared across all time steps

3. Vanishing Gradients

- Gradients shrink exponentially through time
- RNNs struggle with long-term dependencies

4. LSTM

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad h_t = o_t \odot \tanh(c_t)$$

- 3 gates: forget, input, output
- Cell state enables long-term memory

5. GRU

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

- 2 gates: update, reset
- Simpler, often comparable performance to LSTM

Self-Test — Check Your Understanding

Quick Quiz:

1. How many gates does LSTM have? What are they?
2. If $z_t = 0$ in GRU, what happens to the hidden state?
3. For input dim=50, hidden dim=100, how many parameters does one LSTM layer have?

Answers:

1. 3 gates: Forget (f), Input (i), Output (o)
2. $h_t = h_{t-1}$ (old state is completely preserved)
3. $4 \times (50 \times 100 + 100 \times 100 + 100) = 4 \times 15,100 = 60,400$

7 Glossary

Term	Definition
RNN	Recurrent Neural Network, processes sequences with hidden state
Hidden State	Vector carrying information from previous time steps
BPTT	Backpropagation Through Time, training algorithm for RNNs
LSTM	Long Short-Term Memory, gated RNN for long-term dependencies
Cell State	LSTM's long-term memory, flows with minimal modification
Forget Gate	LSTM gate that decides what to remove from cell state
Input Gate	LSTM gate that decides what new information to store
Output Gate	LSTM gate that decides what to output
GRU	Gated Recurrent Unit, simplified LSTM with 2 gates
Update Gate	GRU gate controlling blend of old and new state
Reset Gate	GRU gate controlling how much past to forget