

# Recurrent Neural Networks (RNNs) – Study Material

---

## 1. What Are RNNs?

RNNs are a class of neural networks designed for **sequential data**. Unlike traditional feedforward neural networks, RNNs maintain a **hidden state** across time steps, allowing them to remember past inputs.

### Core Idea:

The output at time  $t$  depends on the input at time  $t$  and the hidden state from time  $t-1$ .

---

## 2. Basic Architecture

For a simple RNN:

- Input:  $x_t$
- Hidden state:  $h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$
- Output:  $y_t = W_{hy}h_t + b_y$

Diagram:

CSS



Copy



Edit

```
x1 → [RNN Cell] → x2 → [RNN Cell] → x3 → ...
      ↑           ↑
    h0         h1 ...
```

## 3. Types of RNNs

Type	Description
Vanilla RNN	Basic RNN with single hidden state
LSTM	Long Short-Term Memory; handles long-term dependencies
GRU	Gated Recurrent Unit; simplified LSTM
Bi-directional	Processes input in both forward and backward directions
Deep RNN	Stack multiple RNN layers for more abstraction

## 4. Problems in Training RNNs

### 1. Vanishing Gradient:

- Gradients shrink during backpropagation.
- RNNs struggle to learn long-range dependencies.

### 2. Exploding Gradient:

- Gradients grow exponentially, causing instability.
- Solution: gradient clipping.

## 5. Long Short-Term Memory (LSTM)

LSTM introduces gates to control information flow:

- **Forget Gate**  $f_t$ : decides what to discard
- **Input Gate**  $i_t$ : decides what new info to add
- **Output Gate**  $o_t$ : controls the output

Key equations:

- $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

- $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
  - $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
  - $h_t = o_t * \tanh(C_t)$
- 

## 6. GRU (Gated Recurrent Unit)

GRUs simplify LSTMs by combining gates:

- Update Gate  $z_t$
- Reset Gate  $r_t$

Equations:

- $z_t = \sigma(W_z x_t + U_z h_{t-1})$
  - $r_t = \sigma(W_r x_t + U_r h_{t-1})$
  - $\tilde{h}_t = \tanh(W_h x_t + U_h (r_t * h_{t-1}))$
  - $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$
- 

## 7. Bidirectional RNNs

Processes the sequence in both directions:

- Forward hidden state:  $\overrightarrow{h}_t$
  - Backward hidden state:  $\overleftarrow{h}_t$
  - Output: concatenation  $h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$
- 

## 8. Applications of RNNs

Domain	Examples
NLP	Text generation, POS tagging, translation
Speech	Speech-to-text, audio classification

---

Domain	Examples
Time Series	Forecasting, anomaly detection
Video	Action recognition, captioning

## 9. RNN vs LSTM vs GRU

Feature	RNN	LSTM	GRU
Long-term memory	Weak	Strong	Strong
Complexity	Low	High	Medium
Training time	Fast	Slow	Faster than LSTM
Gate mechanism	None	Input, Forget, Output	Update, Reset

## 10. Common Interview and Quiz Concepts

- Time Unrolling
- Backpropagation Through Time (BPTT)
- Padding and Masking
- Sequence-to-Sequence Models
- Encoder-Decoder Architectures
- Attention Mechanisms with RNNs

## 11. Key Code Example (LSTM in PyTorch)

python

 Copy  Edit

```
import torch
import torch.nn as nn
class SimpleLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super(SimpleLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
```

```
batch_first=True) self.fc = nn.Linear(hidden_size, 1) def forward(self, x):  
h0 = torch.zeros(num_layers, x.size(0), hidden_size) c0 =  
torch.zeros(num_layers, x.size(0), hidden_size) out, _ = self.lstm(x, (h0,  
c0)) return self.fc(out[:, -1, :])
```

## 12. Practice Tips

- ✓ Understand what happens at each time step
- ✓ Visualize gates and cell memory in LSTM
- ✓ Compare GRU and LSTM on small datasets
- ✓ Try toy datasets like sine waves or text sequences
- ✓ Code a char-level language model from scratch

Would you like this as a **PDF handout** or **PowerPoint slides** for your students?

Explain the lstm equations in great detail

Absolutely. Let's go **step-by-step** through the **LSTM equations** and explain each component with **intuitive analogies**, **mathematical meaning**, and **how it relates to memory and sequence modeling**.

## LSTM: Long Short-Term Memory – Motivation

Traditional RNNs struggle with **long-term dependencies** because gradients **vanish or explode** during training. LSTM addresses this by introducing **gates** and a **memory cell** that can **retain or discard information** as needed.

## Key Components of an LSTM Cell

At each time step  $t$ , the LSTM cell takes:

- Input vector  $x_t$
- Previous hidden state  $h_{t-1}$
- Previous cell state  $C_{t-1}$

It outputs:

- New hidden state  $h_t$
- New cell state  $C_t$

We use 4 gates:

Gate	Purpose
Forget gate	What to forget from the previous cell state
Input gate	What new information to write
Candidate cell	Potential new memory
Output gate	What to send to the next layer or time step



## LSTM Equations in Detail

Let's now go equation by equation.

### 1. Forget Gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- **Purpose:** Decides what part of the old memory  $C_{t-1}$  to keep.
- $\sigma$  (sigmoid): Outputs values between 0 and 1
  - 0  $\rightarrow$  forget completely
  - 1  $\rightarrow$  keep completely
- **Intuition:** It's like deciding whether to keep or forget old information based on current input and past context.

## 2. Input Gate:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

- **Purpose:** Determines which values will be updated in the memory.
  - Works in tandem with the **candidate memory**  $\tilde{C}_t$ .
- 

## 3. Candidate Memory (a.k.a. Cell Input Activation):

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- **Purpose:** Proposes new content to be added to the cell state.
  - **tanh:** Outputs between -1 and 1, making it suitable for adding/subtracting values from memory.
- 

## 4. Cell State Update:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **Purpose:** The heart of LSTM — updates the memory.
  - **How it works:**
    - Forget some of the old memory:  $f_t * C_{t-1}$
    - Add new memory (controlled by input gate):  $i_t * \tilde{C}_t$
  - **This additive nature** avoids vanishing gradients and enables long-term memory.
- 

## 5. Output Gate:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

- **Purpose:** Decides what part of the memory becomes output.
-

## 6. Final Hidden State:

$$h_t = o_t * \tanh(C_t)$$

- **Purpose:** Produces the hidden state (output) from the cell.
  - Applies a tanh nonlinearity to the memory cell, scaled by the output gate.
- 

### Full Workflow Summary

At time  $t$ , using  $x_t$ ,  $h_{t-1}$ , and  $C_{t-1}$ :

1. Compute what to forget:  $f_t$
  2. Compute what to input:  $i_t$  and  $\tilde{C}_t$
  3. Update memory:  $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
  4. Compute output gate:  $o_t$
  5. Output hidden state:  $h_t = o_t * \tanh(C_t)$
- 

### Intuitive Analogy – The Selective Memory Machine

Imagine your brain is filtering information as you read a story:

- The **forget gate** is deciding, *"Should I forget what happened in Chapter 1?"*
  - The **input gate** says, *"This part of Chapter 3 is important—remember this."*
  - The **cell state** is your mental notebook.
  - The **output gate** decides what summary to give to your friend.
- 

### Parameters in Each Gate

Each gate has its own set of weights and biases:

- $W_f, W_i, W_C, W_o$ : weight matrices
- $b_f, b_i, b_C, b_o$ : bias vectors



All of them are **learned during training** using backpropagation through time (BPTT).



## Visualization

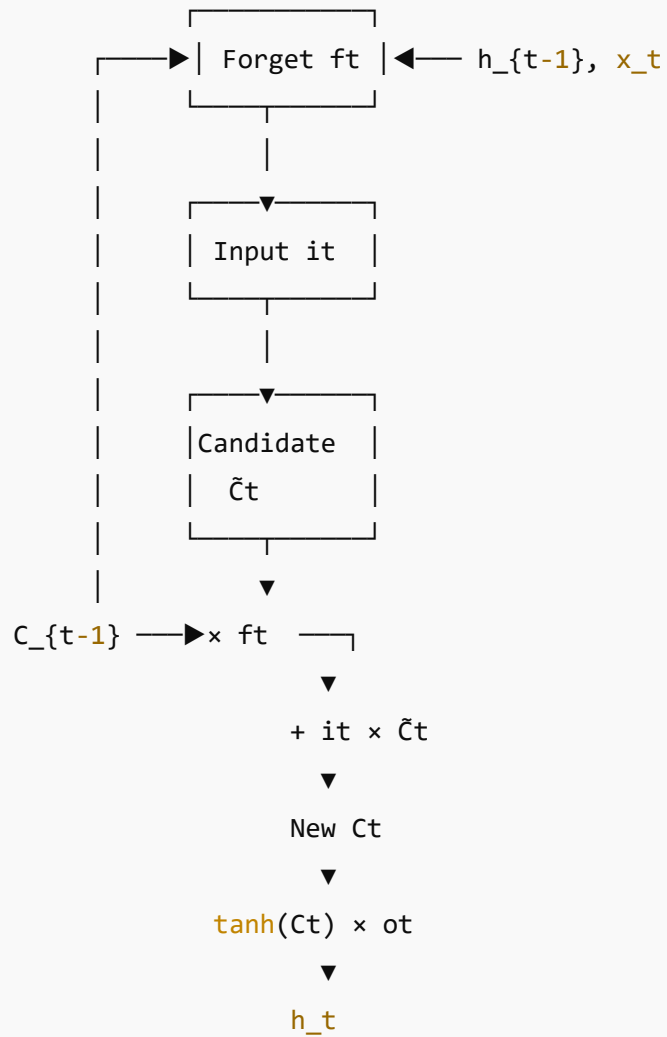
cpp



Copy



Edit



Would you like a **numerical example** where we compute each gate with dummy values?

What is the difference between cell state and hidden state


Great question! The **cell state** and **hidden state** are the two key internal states of an LSTM cell, and understanding their difference is crucial to mastering LSTM behavior.

## Summary: Hidden State vs Cell State

Feature	Hidden State ( $h_t$ )	Cell State ( $C_t$ )
Role	Output at each time step	Internal memory across time steps
Exposed to Output?	Yes — used as LSTM output and passed to next layer	No — internal memory only, not directly output
Affected by Gates	Controlled by <b>output gate</b> and <b>cell state</b>	Controlled by <b>input gate</b> , <b>forget gate</b> , and <b>candidate</b>
Used in Recurrence	Passed to the next time step	Also passed to the next time step
Nature	Short-term, modulated, viewable	Long-term, persistent memory
Equation	$h_t = o_t \cdot \tanh(C_t)$	$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$

### 1. Cell State: $C_t$


- **Main memory** of the LSTM.
- Designed to carry **long-term information**.
- Flows through time with **minor linear operations**, reducing the risk of vanishing gradients.
- Only modified by **forget gate** and **input gate**.

 **Analogy:** Like your long-term memory — it stores what you’ve learned over time and you decide what to keep or discard.

### 2. Hidden State: $h_t$

- Acts as the **visible output** of the LSTM cell.

- Carries **short-term** information — a filtered, current summary of the cell state.
- Used for:
  - Feeding into the next time step
  - Generating predictions
  - Passing to next layer in multi-layer LSTMs

 **Analogy:** Like your current thought or working memory — what you're actively thinking about, based on your past and new input.

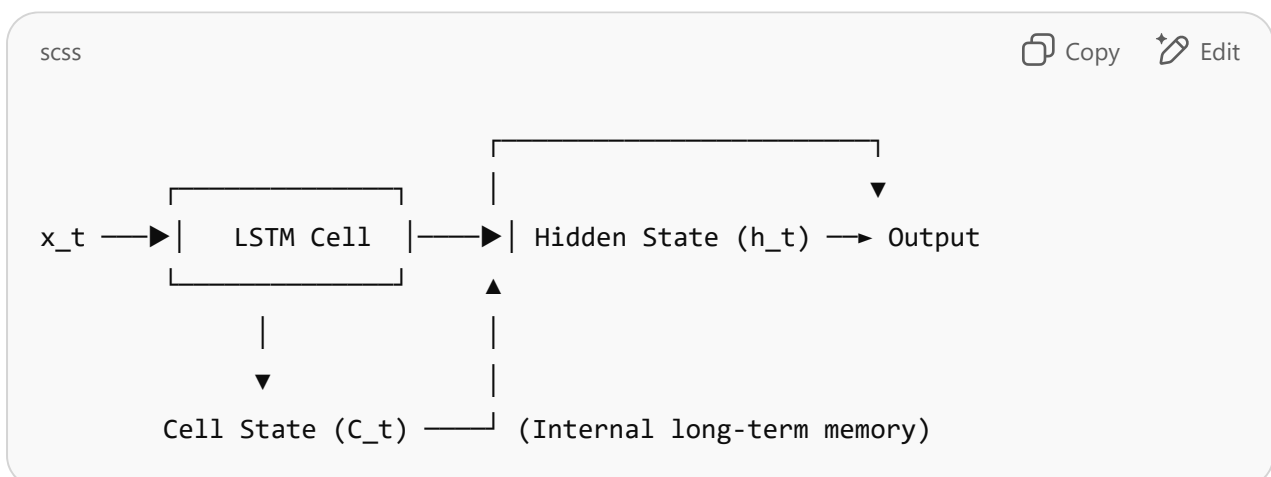
### In Action:

At time step  $t$ :

- Cell state  $C_t$ : is **updated by** the forget and input gates.
- Hidden state  $h_t$ : is **derived from**  $C_t$  and **controlled by** the output gate:

$$h_t = o_t \cdot \tanh(C_t)$$

### Visualization:



### Key Takeaway

- Cell state ( $C_t$ ) stores **persistent memory** — it's the heart of why LSTMs handle long sequences well.

- **Hidden state ( $h_t$ )** is the **immediate output** — what you expose to the next layer or use for predictions.

Would you like to see a **code-based example** printing both states across time steps?

Similarity give detailed tutorial on gru ?

Certainly! Here's a **detailed and intuitive tutorial on Gated Recurrent Units (GRUs)** — covering concepts, math, architecture, and code examples — so your students can fully understand how they work and how they differ from LSTMs.

## Gated Recurrent Units (GRU) – Detailed Tutorial

### 1. What Is a GRU?

GRU stands for **Gated Recurrent Unit**, introduced by Cho et al. in 2014. It is a simplified version of the **LSTM**, designed to solve the **vanishing gradient problem** in standard RNNs — but with fewer gates and parameters than LSTM.

### 2. GRU vs LSTM: The Big Picture

Feature	GRU	LSTM
Gates	2 (Update, Reset)	3 (Input, Forget, Output)
Memory cells	No separate cell state	Has separate cell state $C_t$
Complexity	Simpler, fewer parameters	More complex, larger memory
Computation Speed	Faster	Slower

Feature	GRU	LSTM
Performance	Comparable (depends on task)	Often slightly better on long sequences

### 3. GRU Components & Equations

At each time step  $t$ , the GRU receives:

- Input vector:  $x_t$
- Previous hidden state:  $h_{t-1}$

It computes:

- Update gate  $z_t$
- Reset gate  $r_t$
- Candidate hidden state  $\tilde{h}_t$
- New hidden state  $h_t$

### Step-by-step Equations

#### 1. Update Gate $z_t$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

- Controls how much of the **previous memory** to keep.
- Value close to 1  $\rightarrow$  keep most of the old memory.
- Value close to 0  $\rightarrow$  discard and replace with new content.

#### 2. Reset Gate $r_t$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

- Controls how much of the **past hidden state** to forget before computing the new candidate.
  - If close to 0, forget everything.
- 

### 3. Candidate Hidden State $\tilde{h}_t$

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

- The new candidate state based on the current input and **reset-modified** previous state.
  - The reset gate allows GRU to ignore irrelevant past memory.
- 

### 4. Final Hidden State $h_t$

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

- Blends the **old hidden state** and the **new candidate** based on the update gate.
  - If update gate = 1 → use only the candidate.
  - If update gate = 0 → keep the old memory.
- 

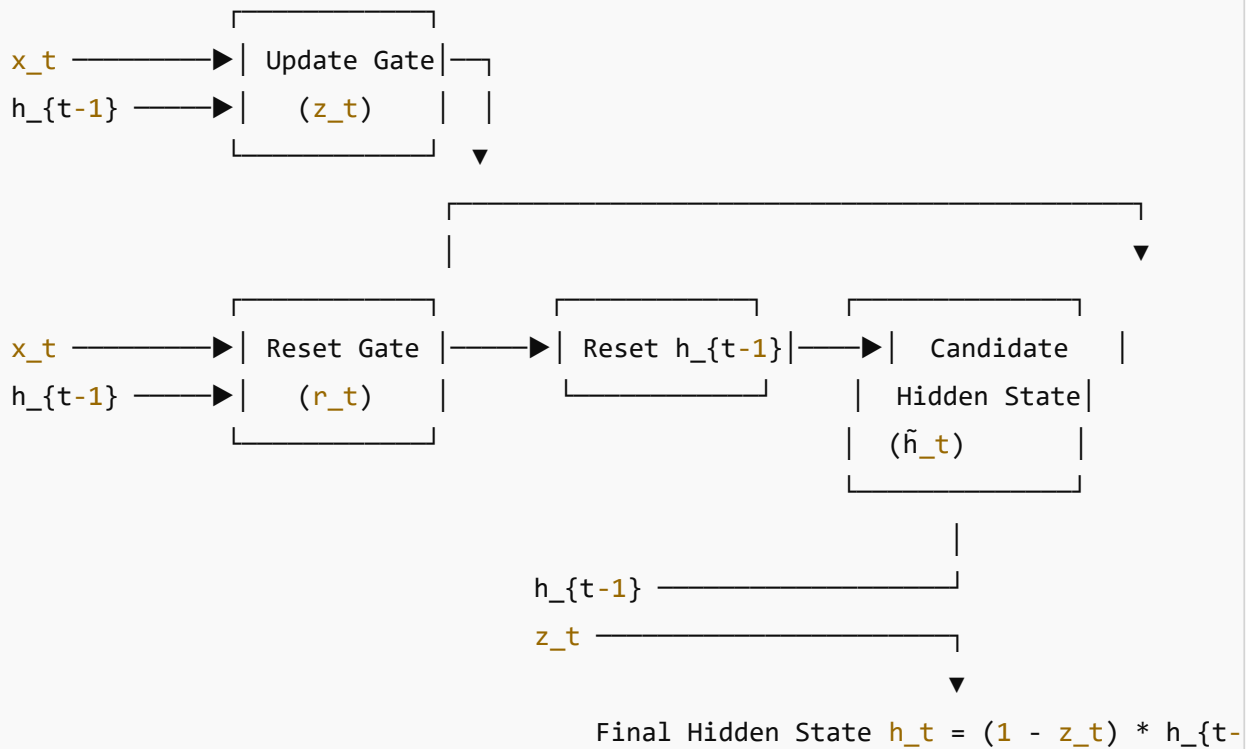
## Intuition: Selective Memory

- **Update gate  $z_t$**  → *How much should I remember from the past?*
  - **Reset gate  $r_t$**  → *How much of the past should I ignore when creating new memory?*
  - **No separate cell state** → GRU combines everything into one **hidden state**.
- 

## GRU Cell Diagram (Textual)

cpp

Copy Edit



## 4. PyTorch Code Example

python

Copy Edit

```
import torch
import torch.nn as nn

class SimpleGRU(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers=1):
        super(SimpleGRU, self).__init__()
        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, 1) # e.g., for regression

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), hidden_size) # (num_layers, batch, hidden_size)
        out, _ = self.gru(x, h0) # out: (batch, seq_len, hidden_size)
        out = self.fc(out[:, -1, :]) # use last time step's output
        return out
```



## 5. Applications of GRU

- Time series forecasting (weather, stock prices)
- NLP tasks: sentiment analysis, translation, question answering
- Speech recognition

- Video captioning
- 

## 6. When to Use GRU Over LSTM?

- When training time is a concern (GRUs are faster).
  - When memory consumption is tight (GRUs have fewer parameters).
  - For shorter sequences or real-time tasks.
  - When empirical testing shows GRU performs equally or better.
-