

Recurrent Neural Networks II

Deep Dive: Long Short-Term Memory (LSTM)

Prof. D.Sc. BARSEKH-ONJI Aboud

Facultad de Ingeniería
Universidad Anáhuac México

January 15, 2026

Agenda

1 Introduction to LSTM

2 LSTM Architecture

3 The 4 Gates: Mathematical Deep Dive

4 Computational Complexity

5 Network Analysis & Fine Tuning

6 Applications and Implementation

The Problem with Standard RNNs

Short-Term Memory

Standard Recurrent Networks (RNNs) suffer from the ****Vanishing Gradient Problem****.

- As the gap between relevant information and the current prediction grows, RNNs lose the ability to learn connections.
- They have difficulty retaining information over long sequences.

The LSTM Solution

Proposed by Hochreiter & Schmidhuber (1997).

- Explicitly designed to avoid the long-term dependency problem.
- Remembering information for long periods is their default behavior, not something they struggle to learn.

Agenda

1 Introduction to LSTM

2 LSTM Architecture

3 The 4 Gates: Mathematical Deep Dive

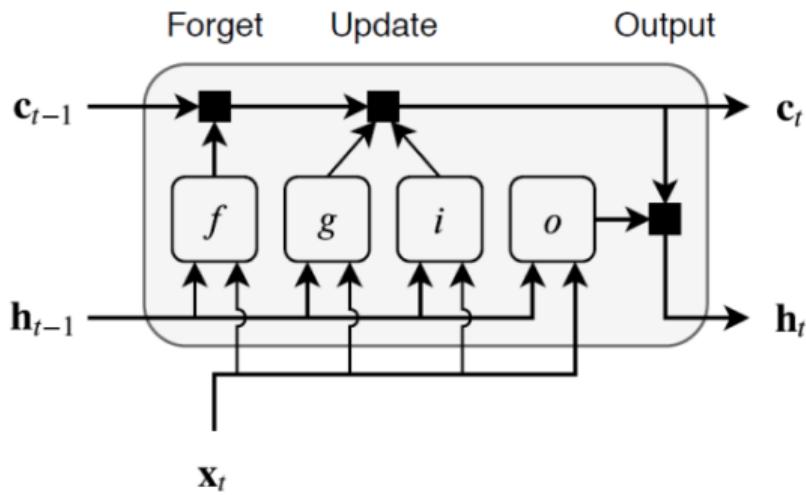
4 Computational Complexity

5 Network Analysis & Fine Tuning

6 Applications and Implementation

The Core Structure

Unlike the repeating module in a standard RNN (a single tanh layer), the LSTM contains four interacting layers (gates).



Key Concepts: States and Inputs

Looking at the diagram, we identify the primary vectors:

- 1 **Input Vector (x_t):** The new information entering the network at time step t .
- 2 **Previous Hidden State (h_{t-1}):** The output of the LSTM from the previous step (Short-term memory).
- 3 **Cell State (c_t):** The internal memory "highway". It runs straight down the entire chain with only minor linear interactions, allowing gradients to flow unchanged (solving vanishing gradient).

Agenda

- 1** Introduction to LSTM
- 2** LSTM Architecture
- 3** The 4 Gates: Mathematical Deep Dive
- 4** Computational Complexity
- 5** Network Analysis & Fine Tuning
- 6** Applications and Implementation

Understanding the Symbols

The figure introduces four distinct blocks (gates). Each block represents a Neural Network layer with its own weights (W) and bias (b).

Symbol	Name	Activation	Role
f	Forget Gate	Sigmoid (σ)	Decides what to delete.
g	Update Gate	Tanh	Creates new candidates.
i	Input Gate	Sigmoid (σ)	Decides importance of g .
o	Output Gate	Sigmoid (σ)	Filters the output \mathbf{h}_t .

Step 1: The Forget Gate (f)

Decision: "What do we throw away?"

The gate looks at \mathbf{h}_{t-1} and \mathbf{x}_t , and outputs a number between 0 and 1 for each number in the cell state \mathbf{c}_{t-1} .

$$\mathbf{f}_t = \sigma(W_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + b_f) \quad (1)$$

Symbol in Diagram:

f



Control of History

- **1:** "Keep this completely."
- **0:** "Get rid of this completely."

Step 2: The Input & Update Gates (i, g)

Decision: "What new info do we store?"

This step has two parts appearing in the diagram as i and g :

1. Input Gate (i):

$$\mathbf{i}_t = \sigma(W_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + b_i) \quad (2)$$

Decides *which values* we'll update.

2. Candidate Update (g):

$$\mathbf{g}_t = \tanh(W_g \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + b_g) \quad (3)$$

Creates a vector of *new candidate values* to be added to the state.

Step 3: Updating the Cell State (\mathbf{c}_t)

The Core Operation

We now update the old cell state, \mathbf{c}_{t-1} , into the new cell state \mathbf{c}_t .

In the diagram, observe the "Merge" line at the top:

$$\mathbf{c}_t = \left(\underbrace{\mathbf{f}_t \odot \mathbf{c}_{t-1}}_{\text{Forget old memory}} \right) + \left(\underbrace{\mathbf{i}_t \odot \mathbf{g}_t}_{\text{Add new scaled info}} \right) \quad (4)$$

*Note: The symbol \odot denotes element-wise multiplication (Hadamard product), represented by the black squares in the diagram.

Step 4: The Output Gate (o)

Finally, we need to decide what we're going to output. This output will be based on our cell state, but a filtered version.

1. Calculate Gate Activation:

$$\mathbf{o}_t = \sigma(W_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + b_o) \quad (5)$$

2. Calculate Hidden State: Push cell state through tanh (to push values between -1 and 1) and multiply by output gate.

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (6)$$

Symbol in Diagram:

 → Output

Outputs both \mathbf{h}_t (for prediction) and passes it to the next step.

Agenda

- 1 Introduction to LSTM
- 2 LSTM Architecture
- 3 The 4 Gates: Mathematical Deep Dive
- 4 Computational Complexity
- 5 Network Analysis & Fine Tuning
- 6 Applications and Implementation

Parameter Complexity

Why does an LSTM have so many learnable parameters in MATLAB?

The Factor of 4

Since an LSTM has 4 separate "Neural Networks" inside (f, i, g, o), the total number of weights is:

$$\text{NumParameters} \approx 4 \times [(n \times m) + (n^2) + n] \quad (7)$$

Where:

- n : Number of Hidden Units.
- m : Input dimension size.

This explains why Deep Network Analyzer shows "Total learnables: 68.1k" for a relatively small network.

Agenda

- 1 Introduction to LSTM
- 2 LSTM Architecture
- 3 The 4 Gates: Mathematical Deep Dive
- 4 Computational Complexity
- 5 Network Analysis & Fine Tuning
- 6 Applications and Implementation

Validating Complexity with Network Analyzer

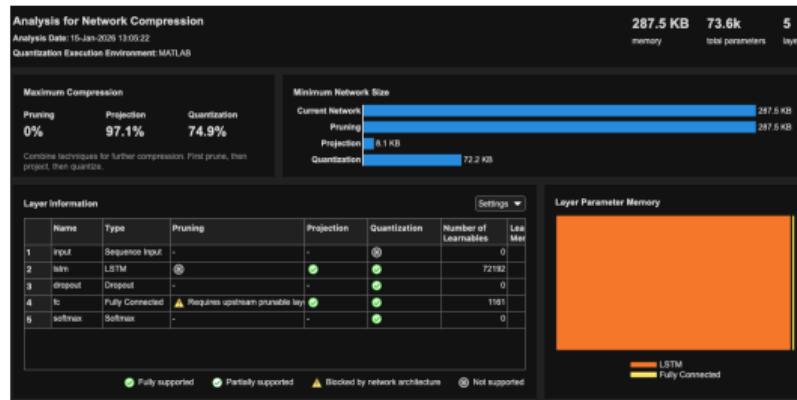


Figure 2: Deep Network Designer Report: 73.3k Learnables.

Applying the Formula

In the provided screenshot: Input (m) = 12, Hidden (n) = 128.

LSTM Parameters Calculation:

$$P \approx 4 \times ((12 \times 128) + (128^2) + 128)$$

$$P \approx 4 \times (1,536 + 16,384 + 128)$$

$$P \approx 72,192 \text{ weights}$$

Final Count: Adding the Fully Connected

The Softmax Layer

From Logits to Probabilities

In the architecture diagram, the LSTM feeds a Fully Connected (FC) layer, which outputs raw scores called **logits**. These can be negative or unbounded (e.g., 2.5, -0.1, 5.0).

The **Softmax Layer** squashes these values to form a valid probability distribution:

$$P(y = j | \mathbf{z}) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad (8)$$

- **Rule 1:** All outputs sum to exactly 1 (100%).
- **Rule 2:** Competitive nature (if Class A goes up, B and C must go down).
- **Usage:** Essential for Multi-class Classification loss calculation.

Initialization Strategies

Choosing the right initialization is critical for convergence in LSTMs. MATLAB offers several options in the dropdown menu:

- **Glorot (Xavier):** The default for LSTMs.
 - optimized for symmetric activations like **Tanh** and **Sigmoid** (the internal gates of LSTM).
 - Keeps variance constant across layers.
- **He:**
 - Optimized for **ReLU** layers. usually avoided for the internal gates of RNNs but used in the FC layers.
- **Orthogonal:**
 - Specifically useful for RNNs/LSTMs to prevent vanishing gradients during very long sequences.

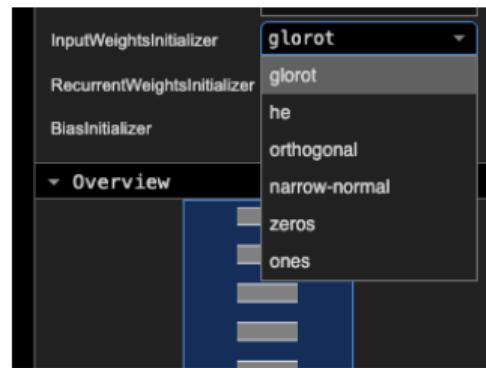


Figure 3: Weight Initializers in MATLAB.

Deployment & Compression

Resource Constraints

LSTMs are memory intensive. The analyzer shows **287.5 KB** for a small network, but deep LSTMs can grow to Gigabytes.

MATLAB's Compression Tools (Pruning, Quantization) help deploy to embedded systems:

Method	Concept	Trade-off
Pruning	Removing "weak" connections	Can reduce accuracy if aggressive.
Quantization	Float32 → Int8	4x Smaller, faster inference.
Projection	Compressing feature maps	Requires retraining.

Agenda

- 1 Introduction to LSTM
- 2 LSTM Architecture
- 3 The 4 Gates: Mathematical Deep Dive
- 4 Computational Complexity
- 5 Network Analysis & Fine Tuning
- 6 Applications and Implementation

Real-World Applications

LSTMs are the state-of-the-art for sequence problems (before Transformers took over NLP):

- 1 **Time Series Classification:** Activity recognition (Acc/Gyro), ECG arrhythmia detection.
- 2 **Forecasting:** Stock prediction, energy load demand.
- 3 **Anomaly Detection:** Predictive maintenance in machinery (detecting patterns that deviate from normal history).
- 4 **Control Systems:** Modeling dynamic systems where state depends on deep history.

MATLAB Implementation

Constructing the network discussed in theory:

Listing 1: Defining LSTM Architecture

```
1 % Sequence Input: 3 Channels (sensors)
2 inputSize = 3;
3 numHiddenUnits = 128; % Generates 128x4 internal weights
4 numClasses = 4;
5
6 layers = [ ...
7     sequenceInputLayer(inputSize)
8     lstmLayer(numHiddenUnits, 'OutputMode', 'last') % Use 'sequence' for
9         seq2seq
10    dropoutLayer(0.5)
11    fullyConnectedLayer(numClasses)
12    softmaxLayer
13    classificationLayer];
```

Conclusion

- **Structure:** LSTMs separate the memory stream (\mathbf{c}_t) from the processing stream (\mathbf{h}_t).
- **Gates:** Through mechanisms f, g, i, o , the network learns purely by gradient descent *what* to remember and *what* to ignore.
- **Result:** They solve the vanishing gradient problem, enabling the modeling of complex temporal dependencies over thousands of time steps.