1. Why LSTM? — Shortcomings of RNN

A standard **Recurrent Neural Network (RNN)** updates its hidden state hth_tht at each time step using:

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

While this works for short sequences, it struggles with **long-term dependencies** because of the **vanishing/exploding gradient problem** during backpropagation through time (BPTT).

- **Vanishing gradients:** Gradients shrink exponentially, making it impossible for the model to update weights related to earlier time steps.
- **Exploding gradients:** Gradients grow uncontrollably, leading to unstable training.

As a result, RNNs **forget early information** in long sequences (e.g., predicting the last word of a long sentence where early context matters).

2. LSTM Architecture

The **Long Short-Term Memory (LSTM)** network solves this by introducing a **cell state** C_t and **gates** to control information flow.

An LSTM unit consists of:

1. **Forget Gate** (f_t) — decides which information to discard from the cell state.
2. **Input Gate** (i_t) — decides which new information to store.
3. **Candidate Cell State** ($\tilde{C}_t$) — new content that could be added to the cell state.
4. **Output Gate** (o_t) — decides what to output as the hidden state.

Equations for LSTM

At time step t:

Forget Gate:

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

Input Gate:

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

Candidate Cell State:

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

Update Cell State:

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

Output Gate:

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

Hidden State:

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

Where:

- σ = sigmoid activation (outputs values between 0 and 1, representing "how much to allow"
- $\tanh$ = hyperbolic tangent activation (outputs between -1 and 1)
- $\odot$ = element-wise multiplication
- W and b = weights and biases learned during training

3. Example: Predicting the Next Word in a Sentence

Sentence: "The cat sat on the ..."

Step-by-step in LSTM:

1. **Input (x_t)**: The word at position t is converted to an embedding vector.
2. **Forget Gate (f_t)**: If the early words ("The cat") are still important, f_t outputs values near 1 for relevant features; otherwise near 0.
3. **Input Gate (i_t)**: Decides if "sat on" is important to store.
4. **Candidate State ($\tilde{C}_t$)**: Encodes possible new meaning from the current word.
5. **Cell State (C_t)**: Merges old context with new important information.
6. **Output Gate (o_t)**: Produces hidden state h_t, which will help predict the next word ("mat").

4. Problems with LSTM

While LSTM solves many RNN issues, it still has:

- **High computational cost:** More parameters due to multiple gates.
- **Long training time:** Complex structure increases training duration.
- **Not optimal for extremely long sequences:** Still limited by memory size.
- **Difficult parallelization:** Sequential processing limits GPU efficiency.