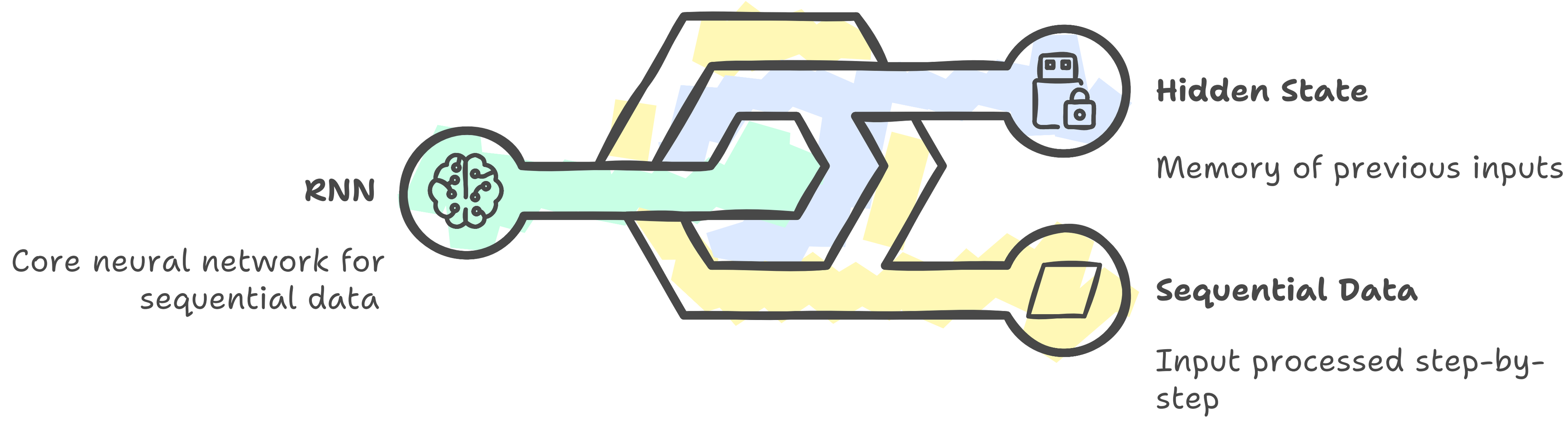
Recurrent Neural Networks

RNN Structure



Recurrent Neural Networks (RNN)

Architecture

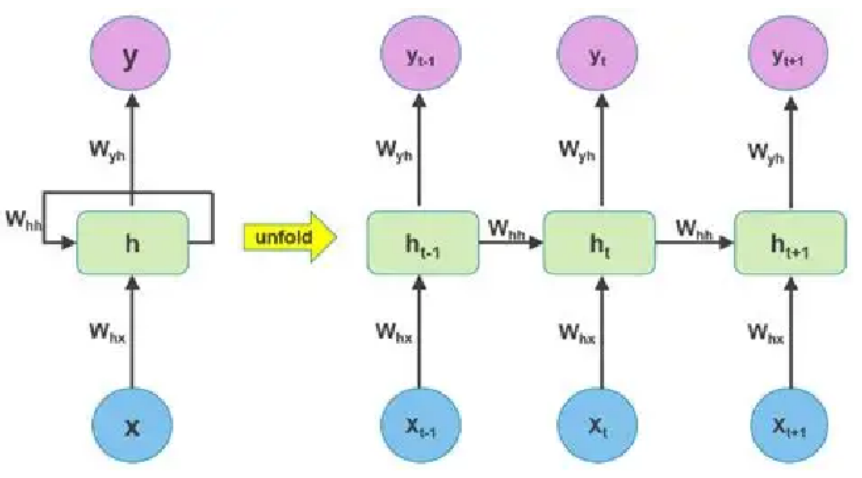
An RNN is a type of neural network designed for sequential data. It processes input one step at a time, maintaining a hidden state ht that acts as a memory of previous inputs.

Mathematical Formulation

At each time step t, the hidden state is updated as $h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h)$

where:
- h_t → Hidden state at time t
- x_t → Input at time t
- W_h, W_x → Weight matrices
- b_h → Bias term
- tanh → Activation function

The output at each time step is: $y_t = W_y h_t + b_y$



Challenges in RNN

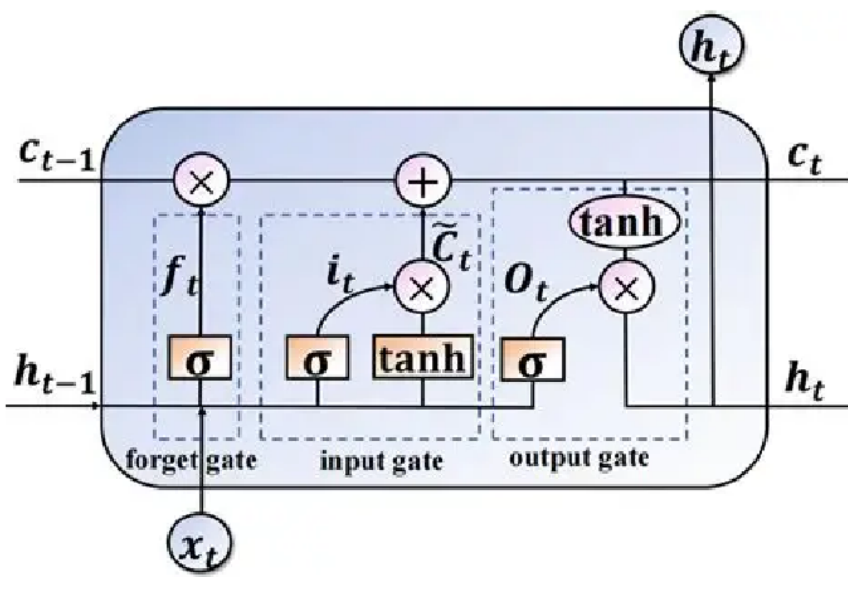
Vanishing Gradient Problem: Gradients become very small for long sequences, making earlier layers hard to update.
Exploding Gradient Problem: Large gradients can cause unstable training.
Short-Term Memory: RNNs struggle to retain long-term dependencies.

Long Short-Term Memory (LSTM)

LSTMs introduce memory cells with gates to regulate information flow.

LSTM Cell Components

- C_t → Cell state - Stores long-term information.
- f_t → Forget gate - Decides what information to discard.
- i_t → Input gate - Decides what new information to store.
- o_t → Output gate - Determines the next hidden state.



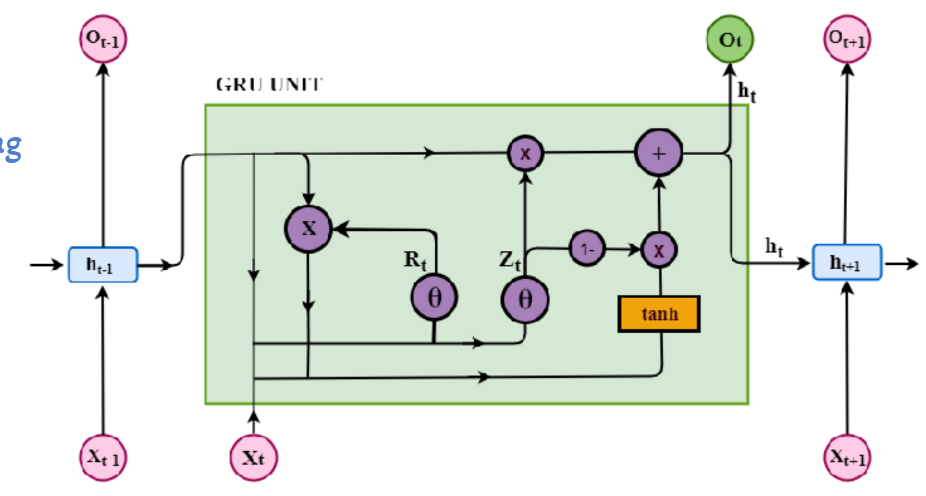
Mathematical Formulation:

- Forget Gate: $f_t = \sigma(W_f [h_{t-1}, x_t] + b_f)$
- Input Gate: $i_t = \sigma(W_i [h_{t-1}, x_t] + b_i)$
- Candidate Cell: $\hat{C}_t = \tanh(W_C [h_{t-1}, x_t] + b_C)$
- Cell State: $C_t = f_t C_{t-1} + i_t \hat{C}_t$
- Output Gate: $o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$
- Hidden State: $h_t = o_t \tanh(C_t)$

Why LSTMs Work Well?

Gates regulate memory flow, preventing vanishing gradients.
Can learn long-term dependencies by selectively forgetting or updating memory.

Gated Recurrent Unit (GRU)

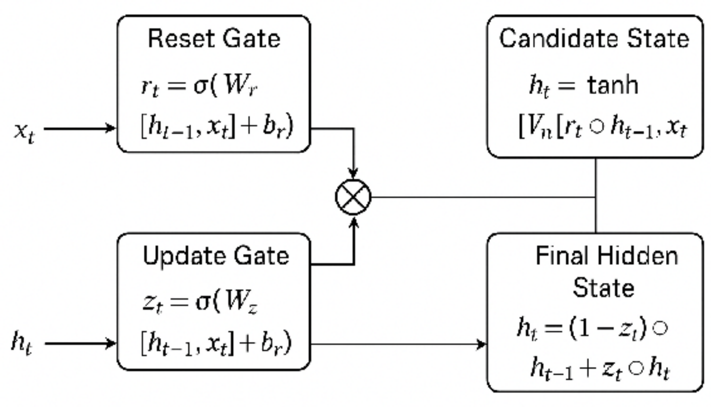


GRU is a simplified version of LSTM that reduces computational complexity while maintaining similar performance.

- GRU Cell Components
  - Reset Gate: r_t → Controls how much past information to forget.
  - Update Gate: z_t → Decides how much of the past state should be carried forward.

Mathematical Formulation

- Reset Gate: $r_t = \sigma(W_r [h_{t-1}, x_t] + b_r)$
- Update Gate: $z_t = \sigma(W_z [h_{t-1}, x_t] + b_z)$
- Candidate State: $\hat{h}_t = \tanh(W_h [r_t \odot h_{t-1}, x_t] + b_h)$
- Final Hidden State: $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$



Why Use GRU?

Fewer parameters than LSTM, making it faster.
Performs similarly to LSTM on many tasks.
Works well when training data is limited.

Gradient Computation in RNNs (BPTT)

Backpropagation Through Time (BPTT) is the process of computing gradients in an unrolled RNN. The key challenge is that hidden states are connected across time, requiring gradients to flow through multiple time steps.

1. Revisiting RNN Equations

For an RNN with input x_t, hidden state h_t, and output y_t, the forward equations are:

Hidden State Update $h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h)$

Output Computation $y_t = W_y h_t + b_y$

Loss Function (The total loss across T time steps) $L = \sum_{t=1}^{T} L_t$

where L_t is the loss at time step t, typically:

MSE (regression): $L_t = ½ (y_t - \hat{y}_t)^2$
Cross-Entropy (classification): $L_t = - \sum_i y_{it} \log(\hat{y}_{it})$

2. Gradient Computation in BPTT

We want: $\partial L/\partial W_h$, $\partial L/\partial W_x$, $\partial L/\partial W_y$

Step 1: Output Gradients $\partial L_t/\partial y_t = \partial L_t/\partial \hat{y}_t$? $\partial \hat{y}_t/\partial y_t$

Since $y_t = W_y h_t + b_y$
$\partial L_t/\partial W_y = \delta_t h_t^T$; where $\delta_t = \partial L_t/\partial y_t$

Step 2: Hidden State Gradients

$\partial L/\partial h_t = \delta_t W_y^T + (\partial L/\partial h_{t+1}$? $W_h^T)$? $\partial h_{t+1}/\partial h_t$

Since ;
$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h)$,
the derivative is: $\partial h_t/\partial h_{t-1} = (1 - h_t^2) W_h$

Step 3: Weight Gradients

Hidden-to-Hidden: $\partial L/\partial W_h = \sum_t \delta_t (1 - h_t^2) h_{t-1}^T$
Input-to-Hidden: $\partial L/\partial W_x = \sum_t \delta_t (1 - h_t^2) x_t^T$

3. Challenges in BPTT

1. Vanishing Gradients
   $(1 - h_t^2) W_h^T$ multiplied repeatedly shrinks exponentially if $|W_h| < 1$.
   → Poor learning of long-term dependencies.
2. Exploding Gradients
   If $|W_h| > 1$, gradients blow up.
   → Solution: Gradient clipping.

4. Weight Update (Gradient Descent)

With learning rate η:

$W_h \leftarrow W_h - \eta\, \partial L/\partial W_h$
$W_x \leftarrow W_x - \eta\, \partial L/\partial W_x$
$W_y \leftarrow W_y - \eta\, \partial L/\partial W_y$

5. Summary

Forward Pass:	Compute h_t and y_t
Backward Pass (BPTT):	Compute $\partial L/\partial h_t$, then $\partial L/\partial W_h$, W_x, W_y
Weight Update:	Apply gradient descent