Recurrent Neural Networks

Understanding Sequential Data Processing

Sina Moghimi, Autumn 2024

МФТИ

Outline

- Sequential Data Fundamentals
- Temporal Dependencies
- Basic RNN Structure
- Memory Mechanism
- RNN Computation Process
- Memory Challenges
- Advanced RNN Variants
- Practical Applications

Sequential Data Fundamentals

- **Sequential data r**efers to any data where the order of elements is **important**
  - In another word, each data point relates to its neighbors. The sequence can be based on time, space, or any logical progression.
- Key Characteristics:
  - **Order Matters:** The sequence cannot be shuffled without altering its meaning
  - **Dependencies:** Data points often depend on their predecessors or successors
  - Examples
    - **Time-based:** Stock prices, weather data
    - **Event-based:** User interactions, system logs
    - **Spatial-based:** DNA sequences, text in natural language processing

МФТИ

Types of Sequential Data

- **Time-Series Data**
  - Data points are collected at successive time intervals
  - Examples: Temperature readings, financial data, or heart rate monitoring
- **Text Data**
  - Sequential words or characters in natural language
  - Examples: Sentences, books, programming code
- **Audio and Speech**
  - Sequences of sound waves or spoken words over time
  - Examples: Music, voice recordings
- **Video Data**
  - Frames in a video sequence, where each frame depends on the previous and subsequent ones
  - Examples: Movies, surveillance footage
- **Event Logs**
  - Sequences of events recorded in a system
  - Examples: Website clickstreams, server logs
- **Biological Sequences**
  - Sequential arrangements in biological data
  - Examples: DNA sequences, protein structures

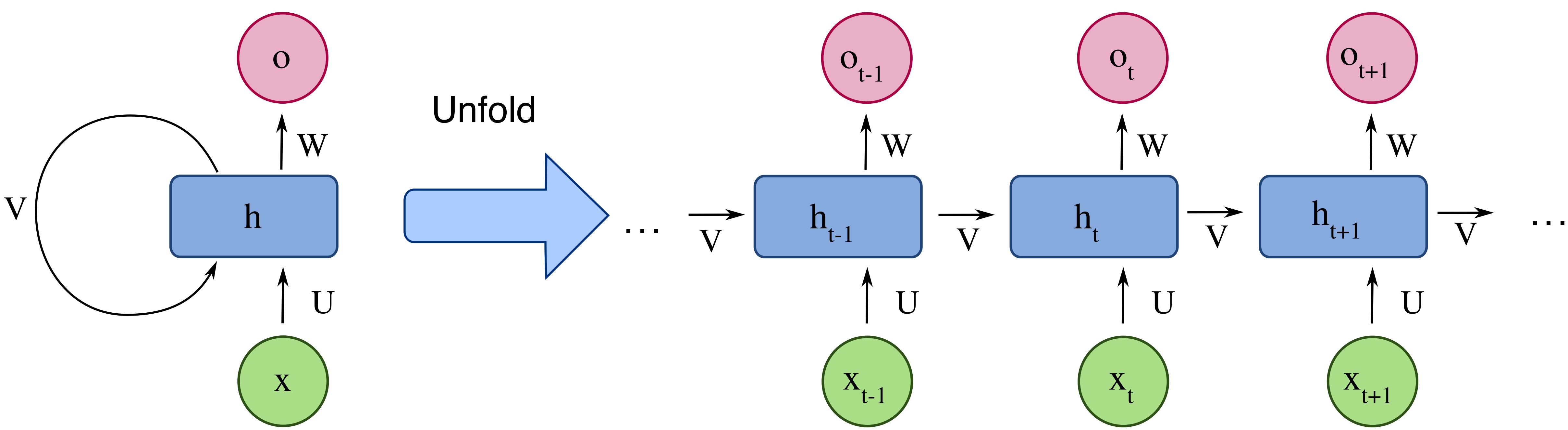
Challenges in Time-Series

- **Handling Long-Term Dependencies**
  - Sequential data often has dependencies across long intervals, which can be challenging to capture
  - Example: Predicting future stock prices based on trends spanning weeks or months
- **Irregular Sampling**
  - Data points may not be evenly spaced, leading to difficulties in analysis
  - Example: Medical data recorded at irregular intervals
- **Noise and Missing Data**
  - Sequential data is often noisy or incomplete, requiring robust preprocessing
  - Example: Missing sensor readings in IoT data
- **Temporal Patterns**
  - Temporal variations and seasonality can complicate modeling
  - Example: Sales data may vary by day, week, or season
- **Data Dimensionality**
  - Sequential data may be multidimensional, making analysis computationally intensive
  - Example: Video data combines temporal and spatial dimensions
- **Overfitting**
  - Models may memorize rather than generalize patterns, especially with limited data
- **Computational Complexity**
  - Sequential data often requires recurrent or attention-based models, which can be computationally expensive
- **Interpretability**
  - Understanding the relationships and patterns within sequential data can be challenging, especially for complex models

МФТИ

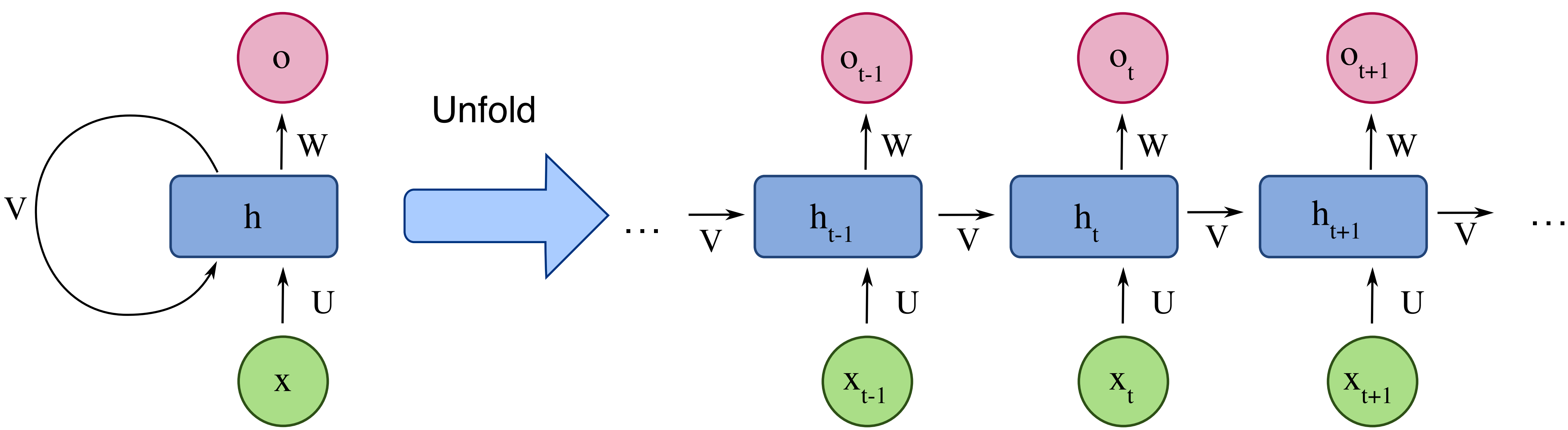
RNNs Structure

- Unlike MLP and CNNs, in **RNNs,** the nodes in each layer, keep some information from previous clones of themselves as well
- **Core Components**
  - **Recurrent Connections**
  - **Hidden State**
  - **Input Layer**
  - **Output Layer**



RNNs Structure

- **Hidden State**
  - A vector that stores information from previous time steps, allowing the network to **remember** earlier inputs
  - It is shared across all time steps, meaning the same weight matrix is used to compute the hidden state at each step
- **Recurrent Connections**
  - A matrix used to propagate the hidden state from one time step to the next, capturing the sequential nature of the data
- **Input Layer**
  - At each time step, the model receives an input, often represented as a vector
- **Output Layer**
  - The output can either be at each time step (for **sequence-to-sequence tasks**) or only after processing the entire sequence (for **sequence-to-label tasks**)



Information Flow

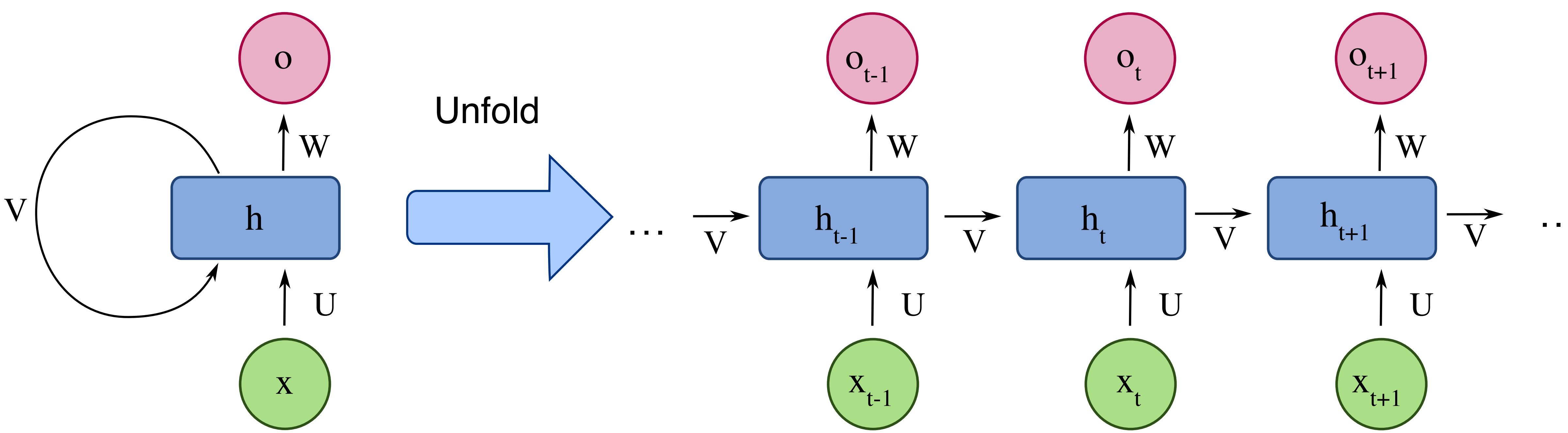
- **At Time Step *t (Forward Pass)***
  - **Input** x_t is the current input vector (e.g., a word in a sentence, a sensor reading, etc.)
  - **Previous Hidden State** h_{t-1} is the hidden state from the previous time step
  - **Recurrent Updated** is to update the hidden state based on the current input and previous hidden state

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

  - **Output** o_t **at time step *t*,** which can be based on the current hidden state

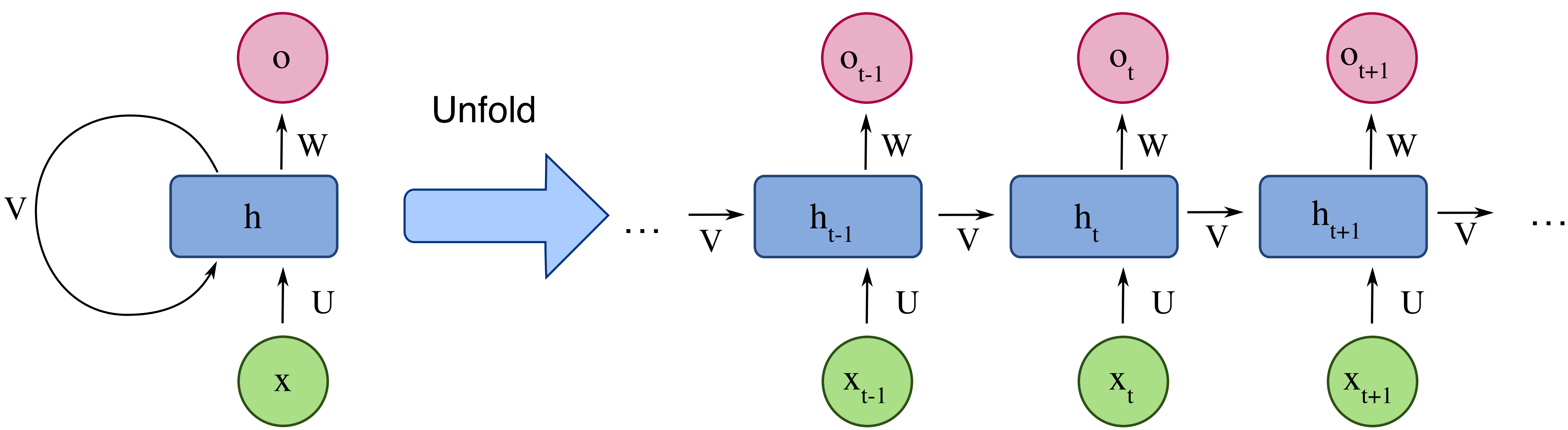
$$o_t = W_{ho}h_t + c$$

  - Where
    - W_{xh} is the weight matrix, connecting the input x_t to the hidden state - W_{hh} is the weight matrix for recurrent connections (from h_{t-1} to h_t) - W_{hy} is the weight matrix from hidden state to the output - b, c are bias terms.



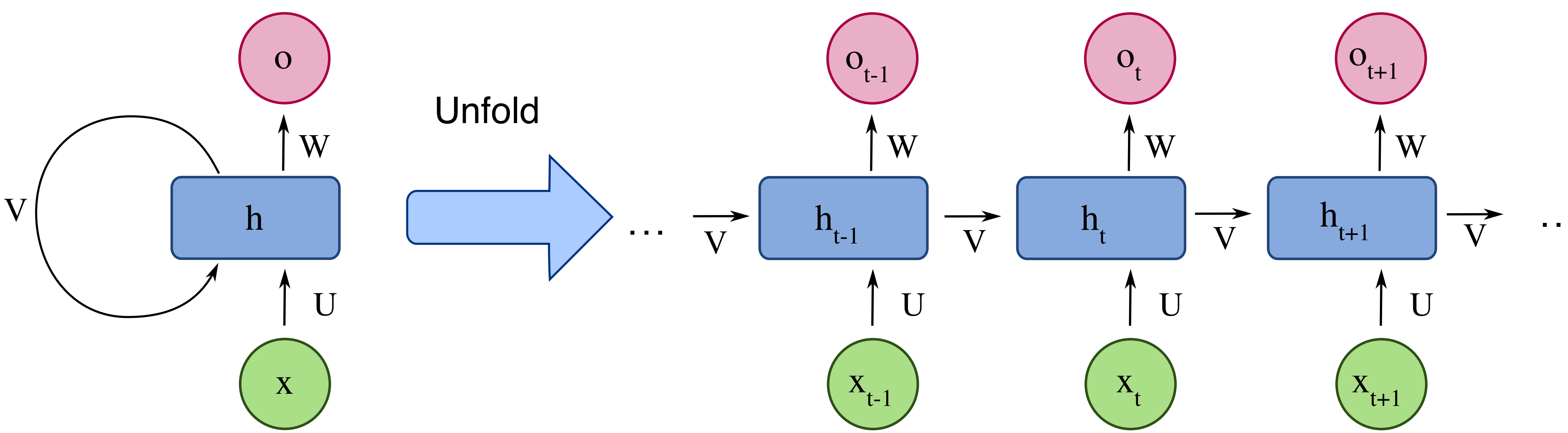
Information Flow

- **Backpropagation Trough Time (BPTT)**
  - During training, RNNs use BPTT to compute gradients across all time steps
  - The loss is calculated over the sequence, and gradients are propagated backward through the recurrent connections



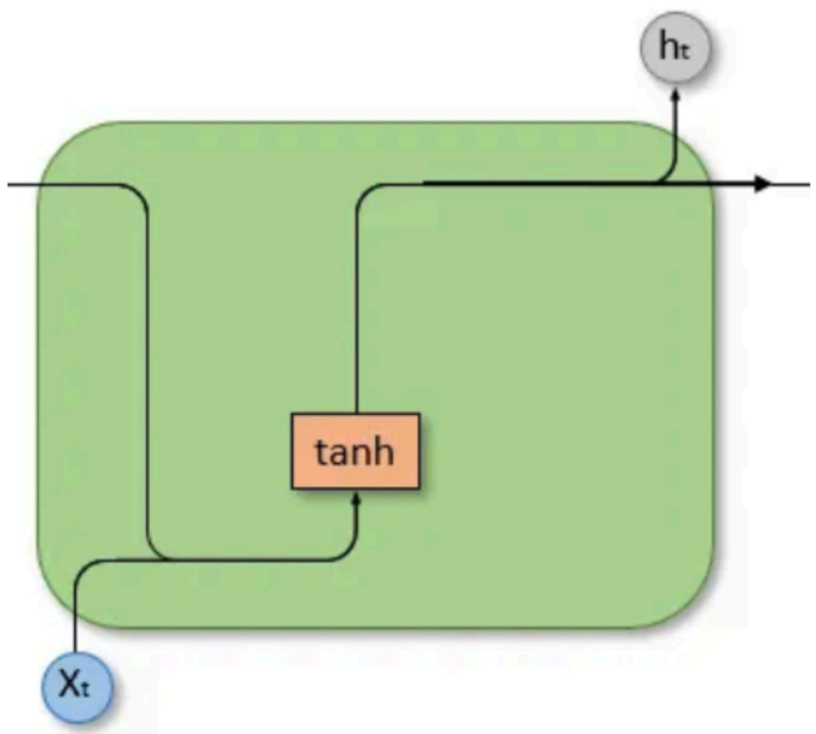
Long-Term Dependency Challenge

- **Vanishing Gradients**
  - During backpropagation, gradients become increasingly small as they are propagated through many time steps
  - Very old data become less important
- **Exploding Gradients**
  - Conversely, gradients may grow exponentially large during backpropagation, destabilizing the training process
  - Can be solved by **gradient clipping**
- **Forgetting Mechanism**
  - Due to overwriting the hidden state, RNNs forget critical information from earlier time steps

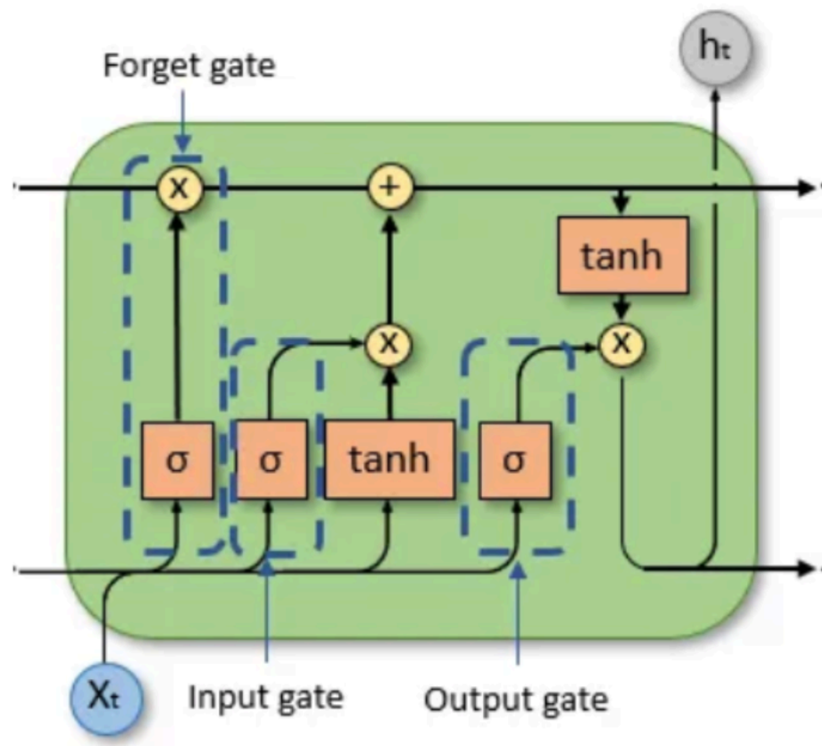


Advanced Variants

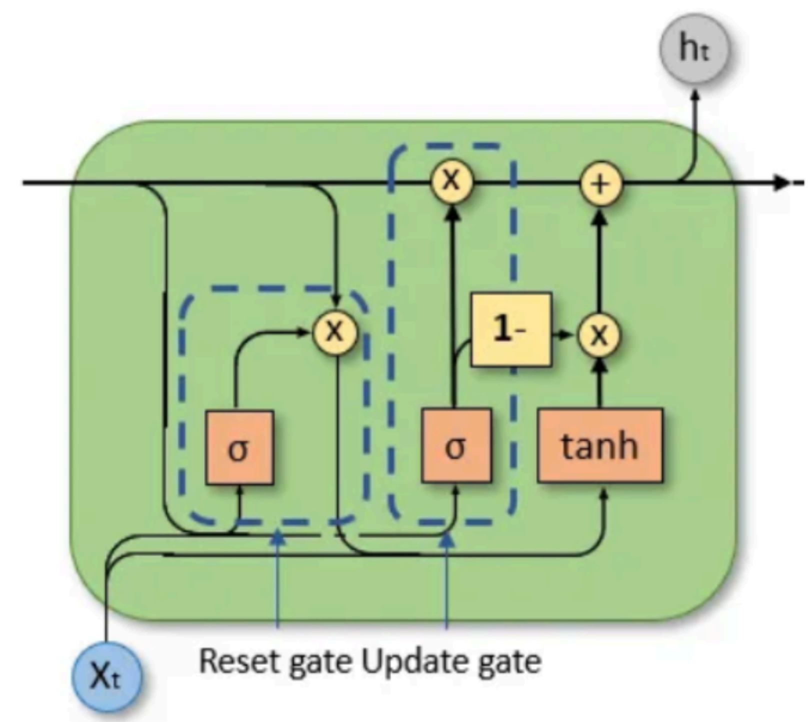
RNN



LSTM



GRU



https://aiml.com/compare-the-different-sequence-models-rnn-lstm-gru-and-transformers/

Comparison

	RNN	LSTM	GRU
Architecture	Simple structure with recurrent connections	Incorporates memory cells and gating mechanisms (input, forget, output gates)	Combines memory mechanisms with fewer gates (update and reset)
Gating Mechanism	None	Input gate, forget gate, and output gate	Update gate and reset gate
Memory Mechanism	Relies only on hidden state for memory	Adds a cell state to store long-term dependencies	Combines hidden state and memory functions into fewer components
Ability to handle Long-Term data	Poor (suffers from vanishing gradients)	Excellent (designed to mitigate vanishing gradient problems)	Good (handles long-term dependencies effectively, similar to LSTM)
Training Efficiency	Fast, but struggles with capturing long-term patterns	Slower due to more parameters and computations	Faster than LSTM due to fewer gates and parameters
Pros	Simplicity, fewer computational requirements	Robust handling of sequential data with long-term dependencies	Efficient with fewer parameters while retaining strong performance
Cons	Struggles with vanishing gradients, poor memory	Higher computational cost and complexity	May lose detailed information due to fewer parameters than LSTM

Practical Applications

- Natural Language Processing
- Speech Recognition
- Stock Market Analysis
- Self-Driving Cars Control
- Surveillance Video Analysis