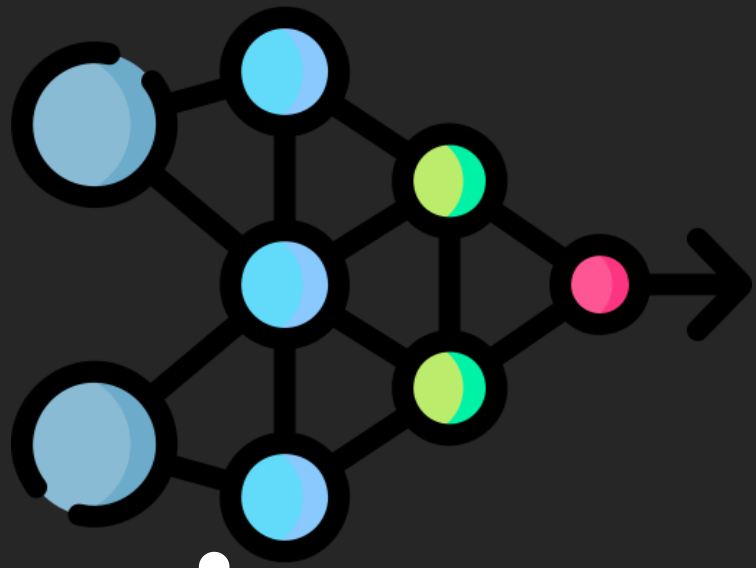


Sequence Learning

Deep Learning Models for Sequential Data

Lecture 20 – HCCDA-AI

Imran Nawar

Wajahat Ullah

17 August 2025

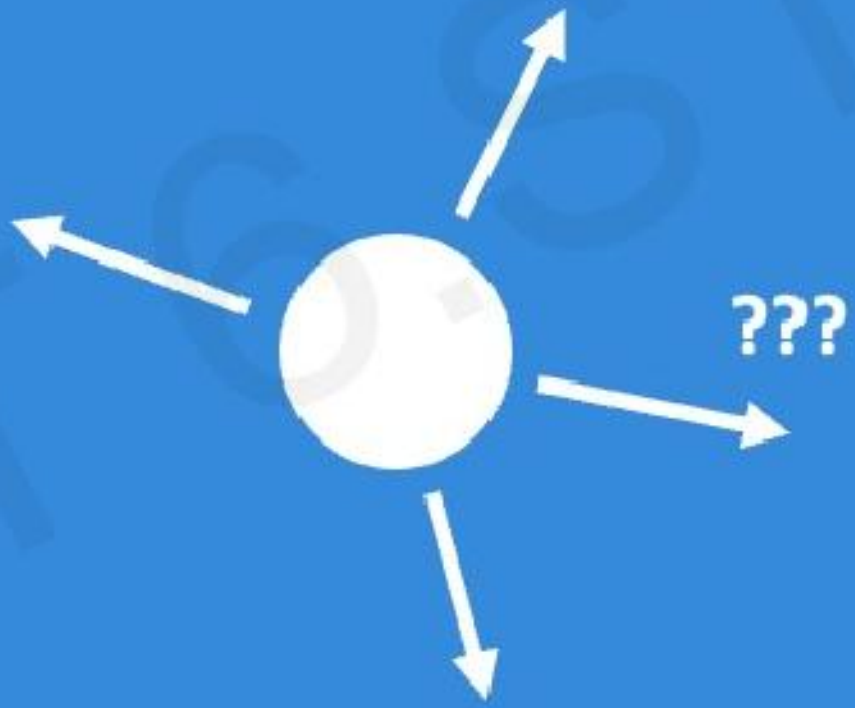
Outline

- Sequences in the Wild
- **What is Sequential Data?**
- **Sequence Modeling Applications**
  - Sequence Modeling – Video & Motion (Computer Vision)
  - Sequence Modeling – Natural Language Processing
- **Modeling Sequential Data before RNNs**
  - Frame-by-Frame Processing (FNNs & 2D CNNs)
  - Two-Stream Networks
  - 3D Convolutional Neural Networks
- Types of Neural Networks (by structure)
- Neurons with Recurrence
- **Recurrent Neural Networks**
  - Recurrent neurons
  - Memory cells (Hidden State)
  - Types of RNN Architecture
  - Training RNNs – Backpropagation Through Time (BPTT)
  - Challenges of RNN
- A sequence modeling problem: Predict the next word
- **Long Short-term Memory, LSTM**
  - LSTM – Core Idea
  - LSTM Cell Architecture
- Gated Recurrent Unit, GRU
- Combining CNNs and RNNs

Given an image of a ball,
can you predict where it will go next?

[Courtesy of Ava Amini]

Given an image of a ball,
can you predict where it will go next?



Given an image of a ball,
can you predict where it will go next?

Given an image of a ball,
can you predict where it will go next?

Sequences in the Wild

Audio

Sequences in the Wild

Audio

Sequences in the Wild

2025 Artificial Intelligence Course

Text

Sequences in the Wild

character: 2 0 2 5

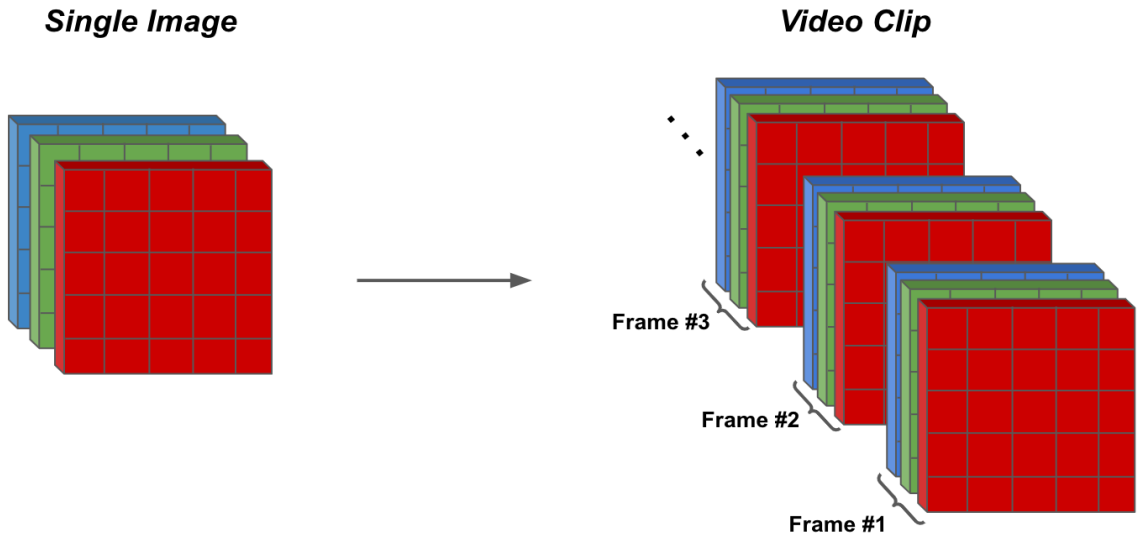
word Artificial Intelligence Course

Text

Sequences in the Wild

What is Sequential Data?

- Sequential data is data where the order of elements matters, past and future values are often dependent on each other.



- **Key Modalities:**

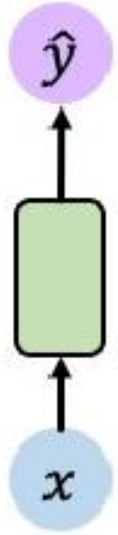
Text
A sentence is a sequence of words.

Time Series
Stock prices change over time

Video
A sequence of image frames

Audio
Speech is a sequence of sounds

Sequence Modeling Applications

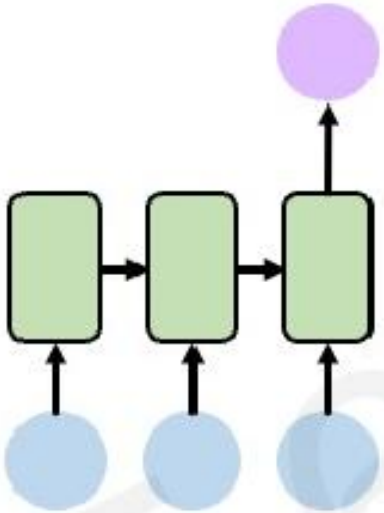


One to One

Binary Classification

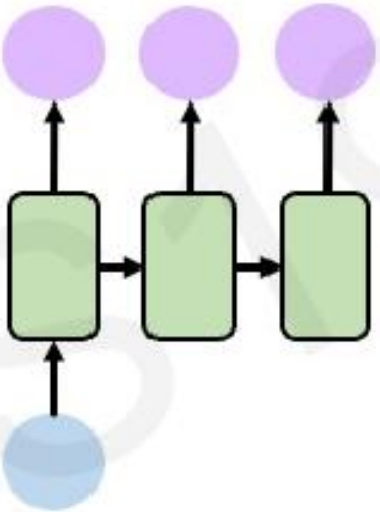
"Will I pass this class?"

Student → Pass?



Many to One

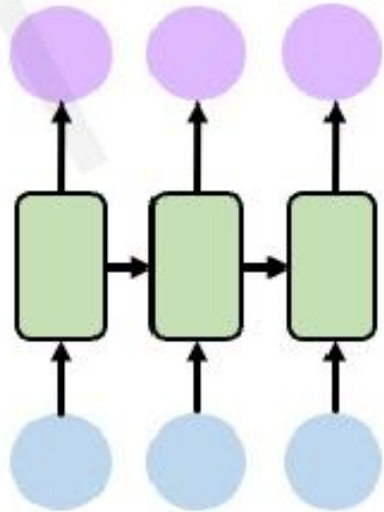
Sentiment Classification



One to Many

Image Captioning

"A baseball player throws a ball."



Many to Many

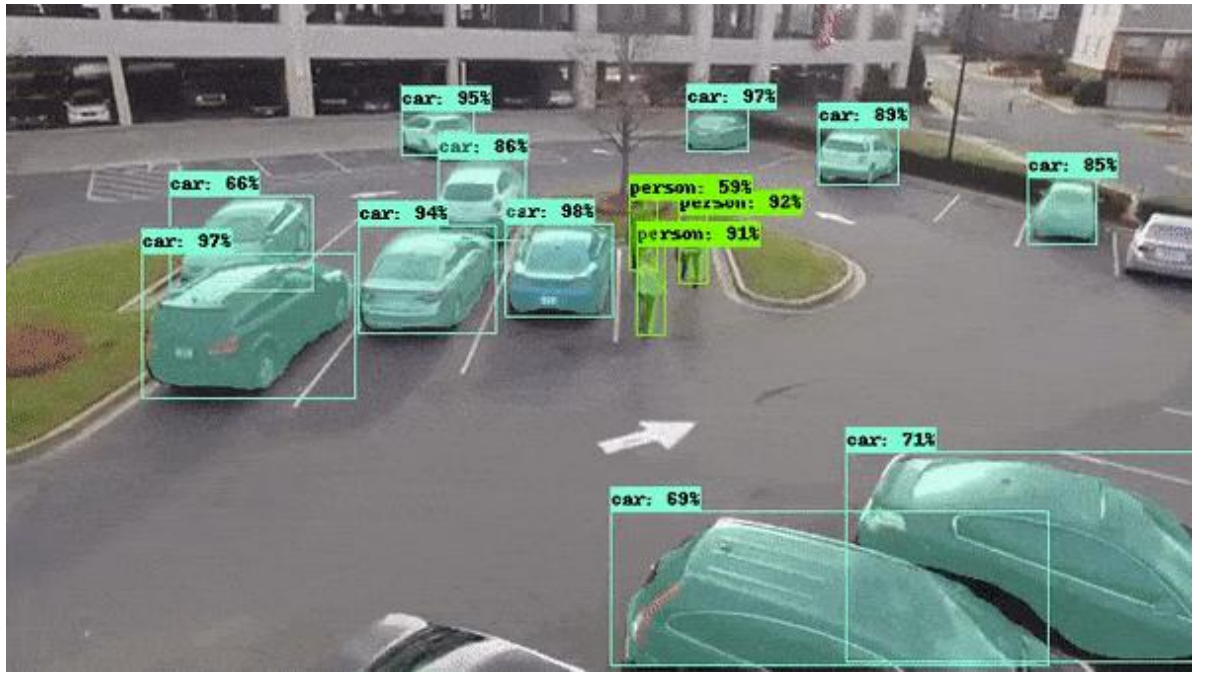
Machine Translation

Sequence Modeling – Video & Motion (Computer Vision)

Modeling temporal patterns in visual data to recognize, predict, or generate motion-related information.

Key Applications:

- **Video Analysis**: Analyzing entire video clips to detect activities, events, or overall themes.
- **Human Activity Recognition**: Identifying and classifying actions in a video, such as walking, running, or waving.
- **Surveillance Systems:** Identifying unusual behaviors for security monitoring
- **Trajectory and Motion Prediction**: Predicting future movement paths of objects, vehicles, or people.

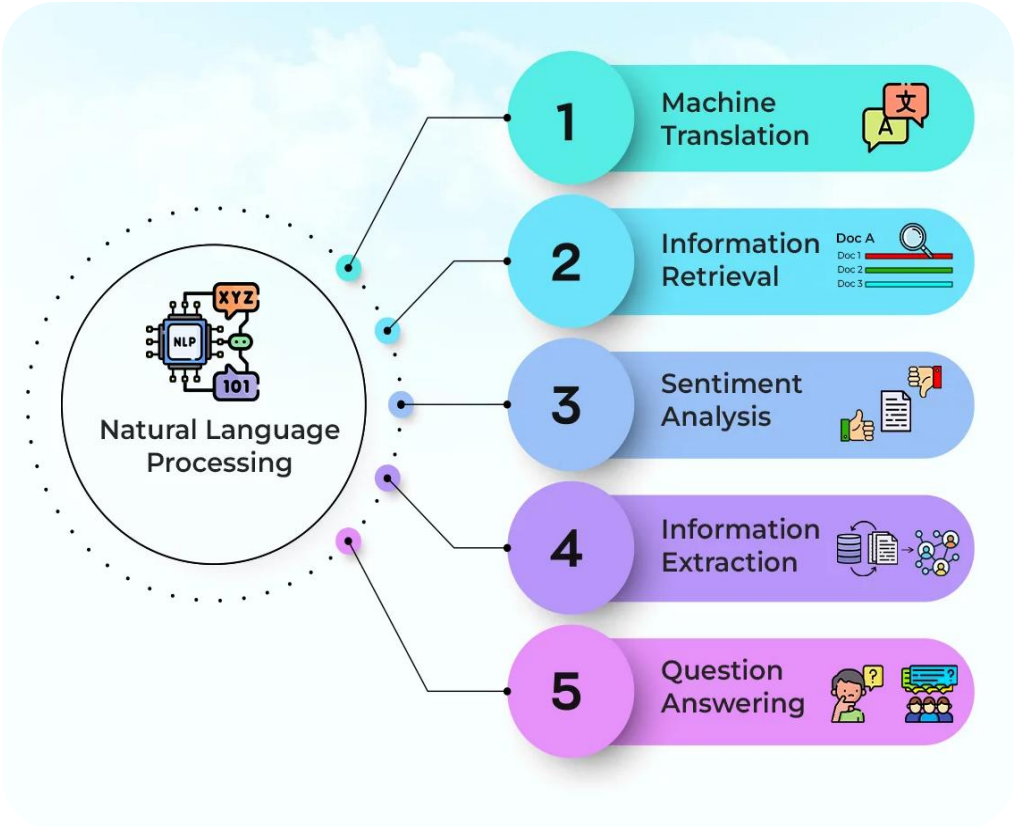


Sequence Modeling – Natural Language Processing (NLP)

Modeling sequences of tokens to understand, transform, or generate human language.

Key Applications:

- **Machine Translation:** Converting text from one language to another. (e.g., Google Translate).
- **Sentiment Analysis**: Determining whether a piece of text expresses positive, negative, or neutral emotions.
- **Text Summarization:** Condensing long documents into key points.
- **Question Answering:** Finding accurate answers to questions based on given text or documents.
- **Text Generation:** Creating human-like text (e.g., AI chatbots, content creation).



Modeling Sequential Data Before RNNs

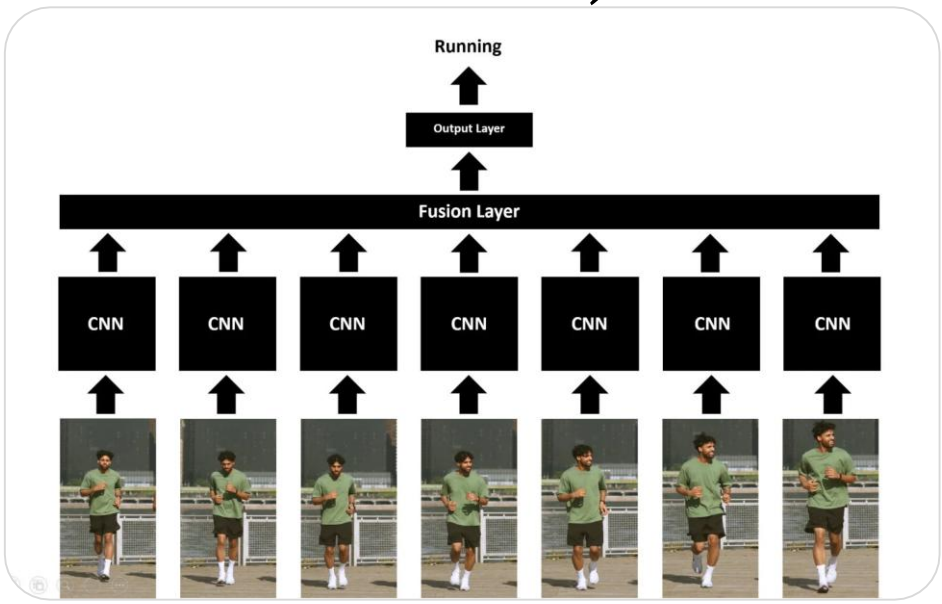
Frame-by-Frame Processing (FNNs & 2D CNNs)

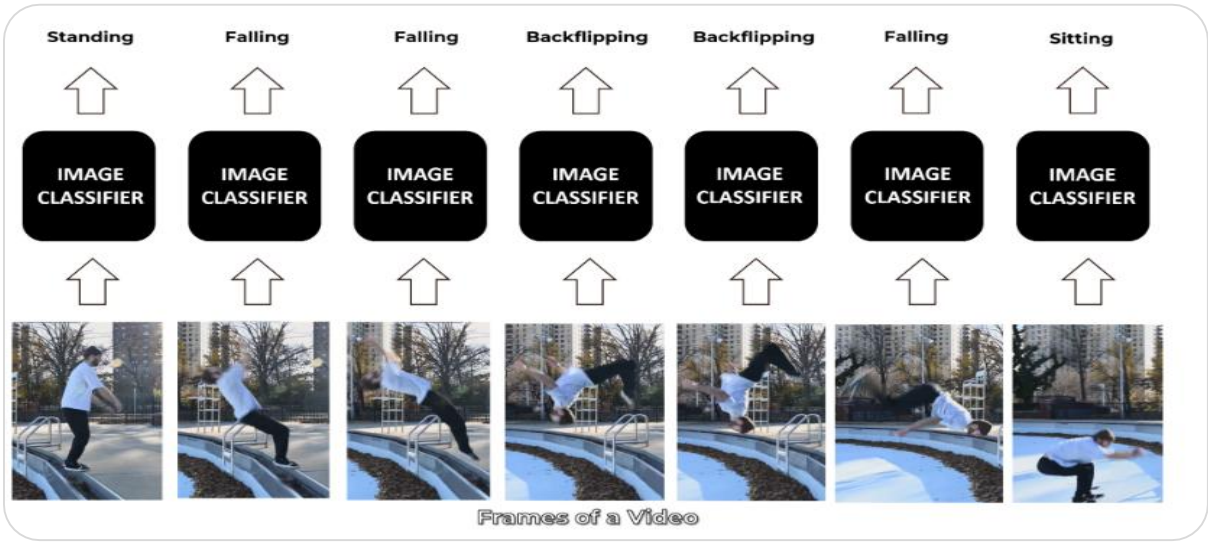
How It Works:

- Processes each frame independently, without considering temporal relationship between frames.
- Common in early video processing and basic time series tasks.

- **Limitations:**
  - **No temporal awareness**: Cannot detect changes or patterns over time.
  - **Loss of context**: Ignores information from previous frames.
  - **Poor for sequential tasks**: Cannot make predictions based on past observations.

- **Example:** Classifying objects in individual video frames without understanding motion.

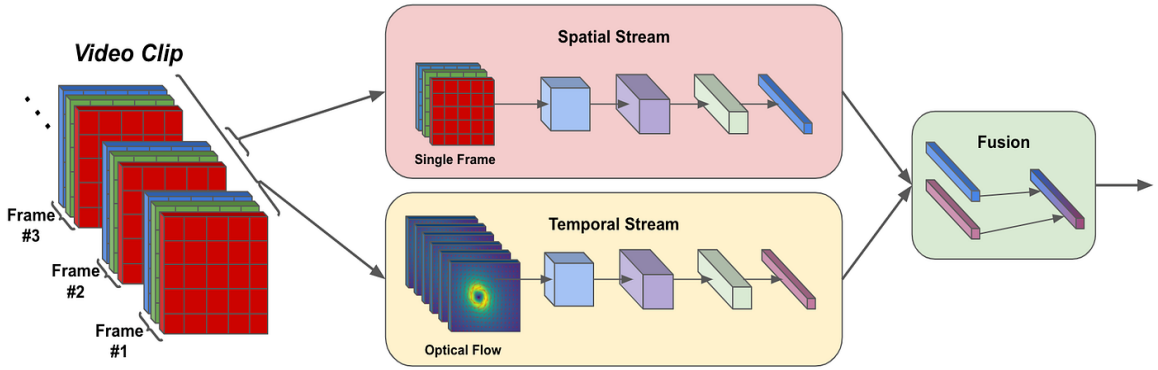




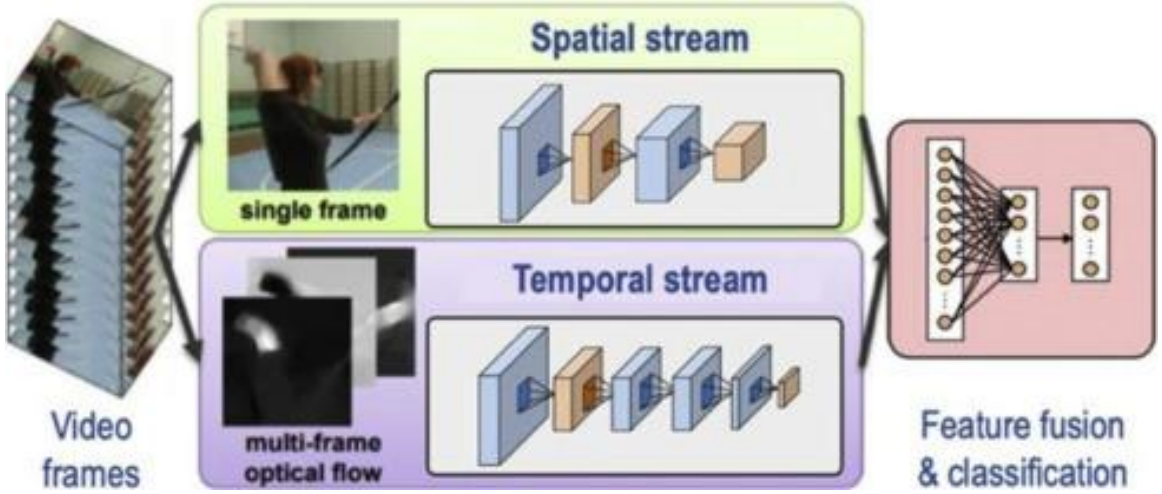
Two-Stream Networks

How It Works:

- Uses two parallel networks:
  - **Spatial stream:** Extract features from individual frames (appearance).
  - **Temporal stream:** Uses optical flow to capture motion between frames.
- Features from both networks are fused for final classification.



- **Strengths:**
  - Captures both appearance and short-term motion information.
- **Limitations:**
  - **Computationally expensive**: Requires both video frames and optical flow data.
  - **Fails in complex motion**: Struggles when motion alone is not enough (e.g., subtle gestures).
- **Example:** Human action recognition (e.g., detecting running vs. walking).

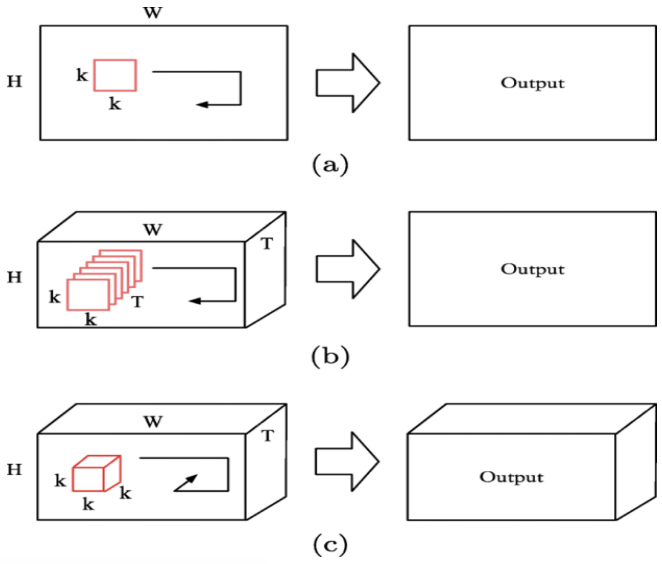


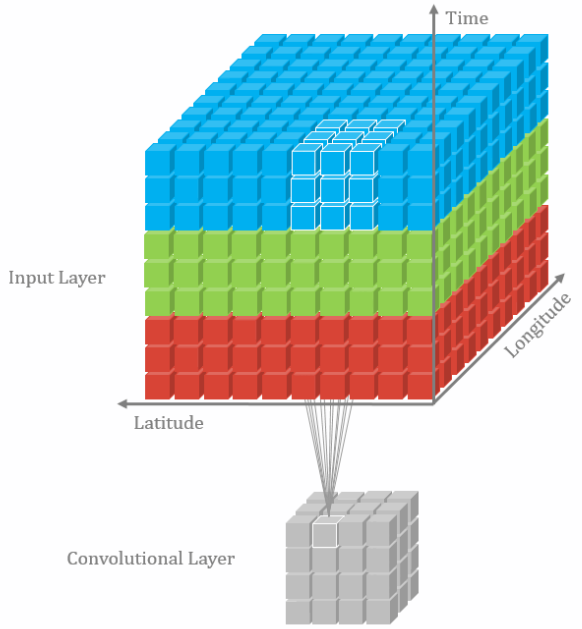
3D Convolutional Neural Networks (3D CNNs)

How It Works:

- Extends 2D convolutions by adding a temporal dimension (depth).
- Uses 3D kernels (depth, height, width) that slide across both space and time.
- Extracts spatio-temporal features directly from video clips.

- **Strengths:**
  - Captures both spatial and temporal patterns in videos.
  - Effective for video classification and action recognition tasks.
- **Limitations:**
  - High computational cost and large memory requirements.
  - Prone to overfitting; requires large datasets.
  - Limited ability to capture very long-term dependencies.





Types of Neural Networks (by structure)

1. Feedforward Neural Networks:

- Information flows in one direction.
  (input → output)
- No feedback connections or internal temporal state.
  $x \rightarrow H \rightarrow y$
- **Examples:**
  - **MLP:** Fully connected layers (baseline).
  - **CNN:** Widely used for images & spatial data.
  - **Autoencoders:** Encoder-decoder feedforward network for compression/ reconstruction.
  - **GAN (components):** Generator and discriminator are typically feedforward (often CNNs for images).

2. Recurrent Neural Networks:

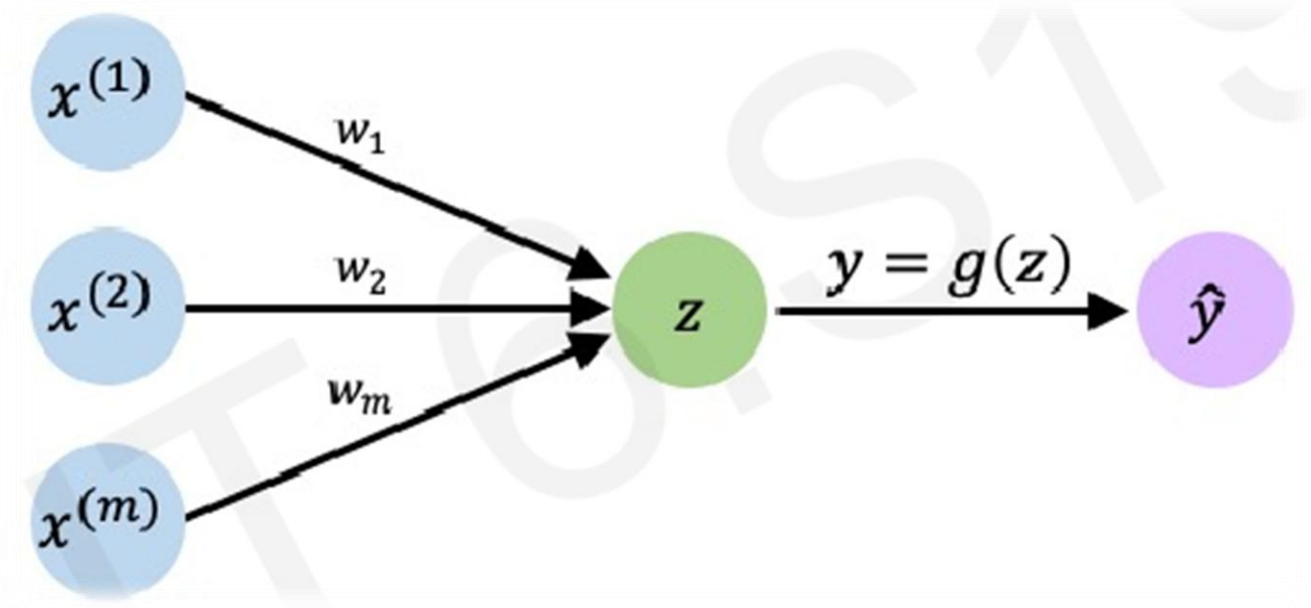
- Maintain a hidden state that carries information over time. Suitable for sequences.
- Previous hidden state influences the current computation.
  $h_t = f(x_t, h_{t-1})$
- **Examples:**
  - **Basic RNN**
  - **LSTM:** Handles long-term dependencies via gates.
  - **GRU:** Simpler alternative to LSTM

3. Attention-Based Networks:

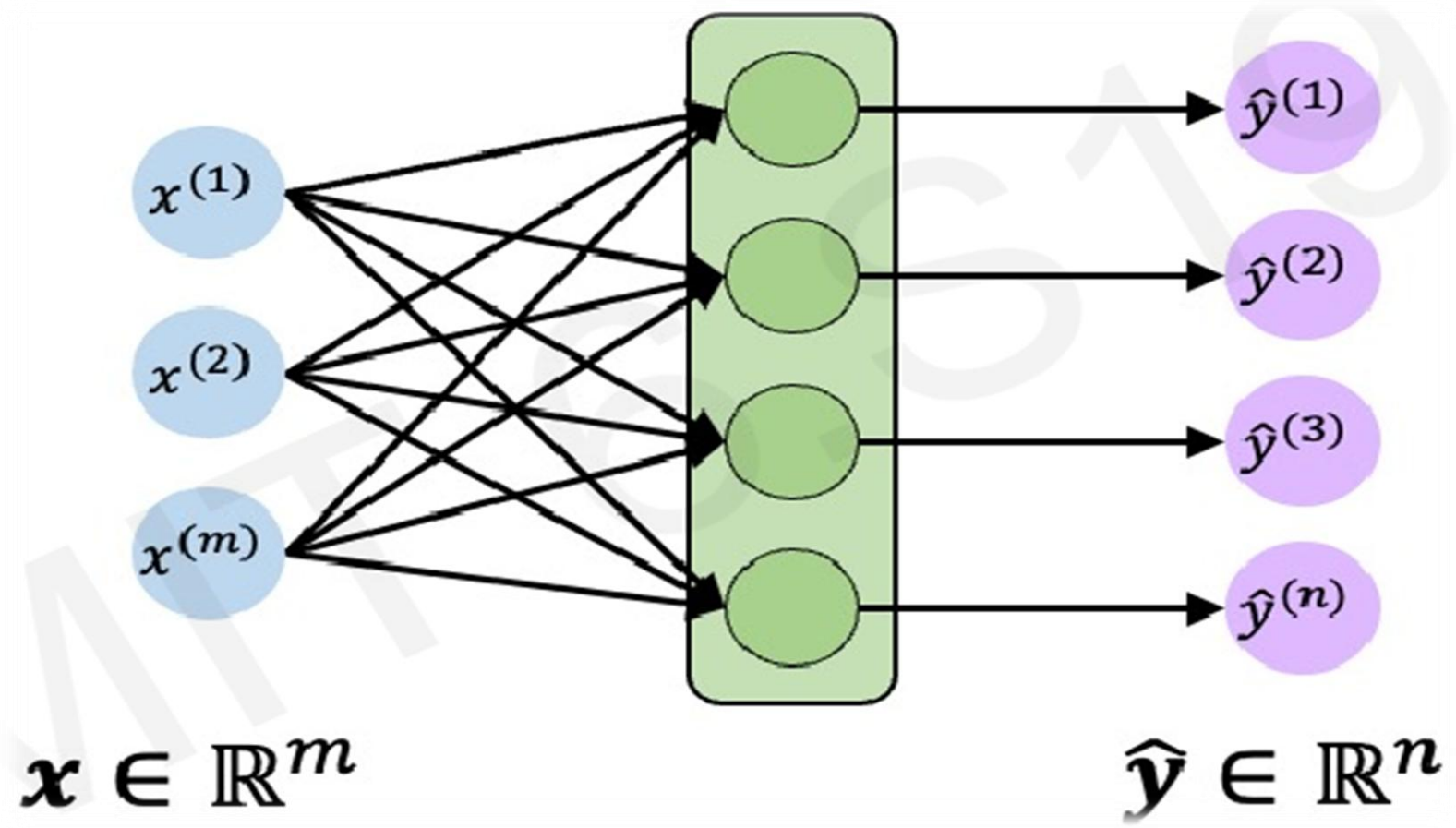
- Process sequences in parallel using self-attention plus positional encodings; no recurrence.
- **Example:**
  - **Transformer:** Encoder, decoder, or encoder-decoder architectures; built from attention + normalization + position-wise feedforward sublayers.

Neurons with Recurrence

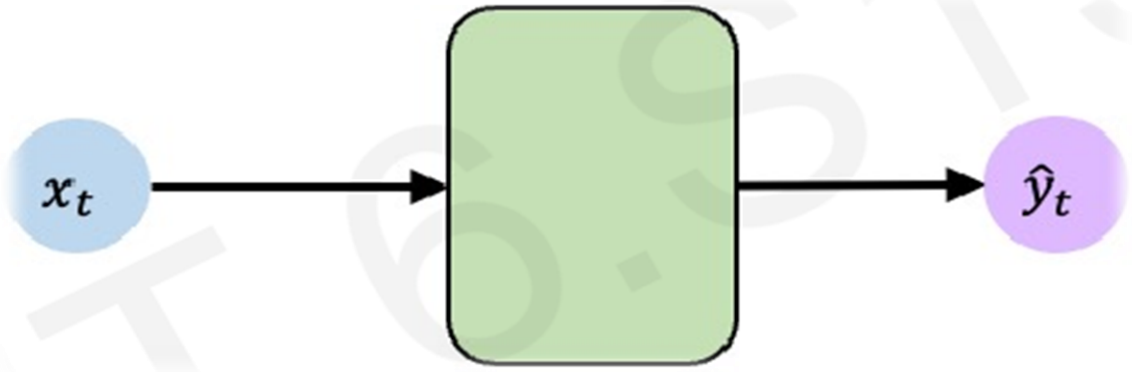
The Perceptron Revisited



Feed-Forward Networks Revisited

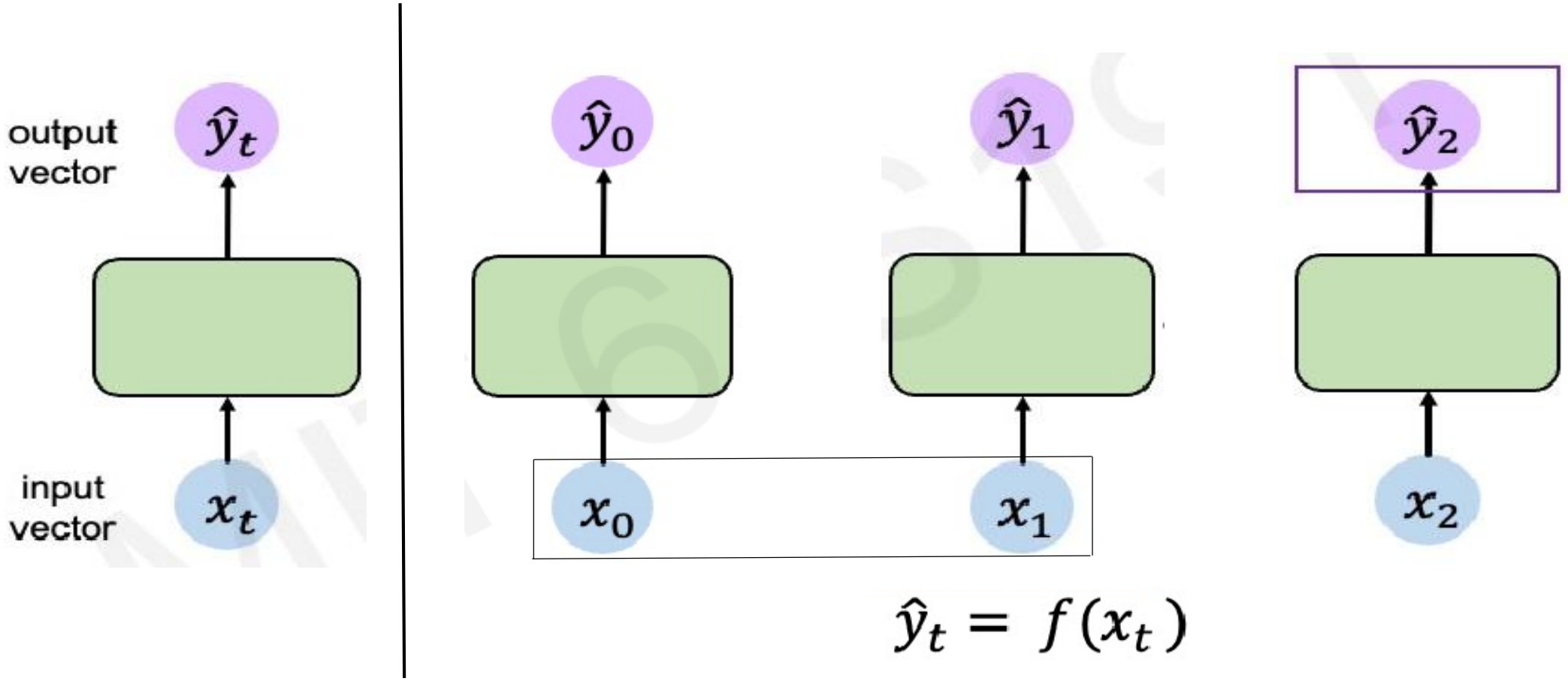


Feed-Forward Networks Revisited

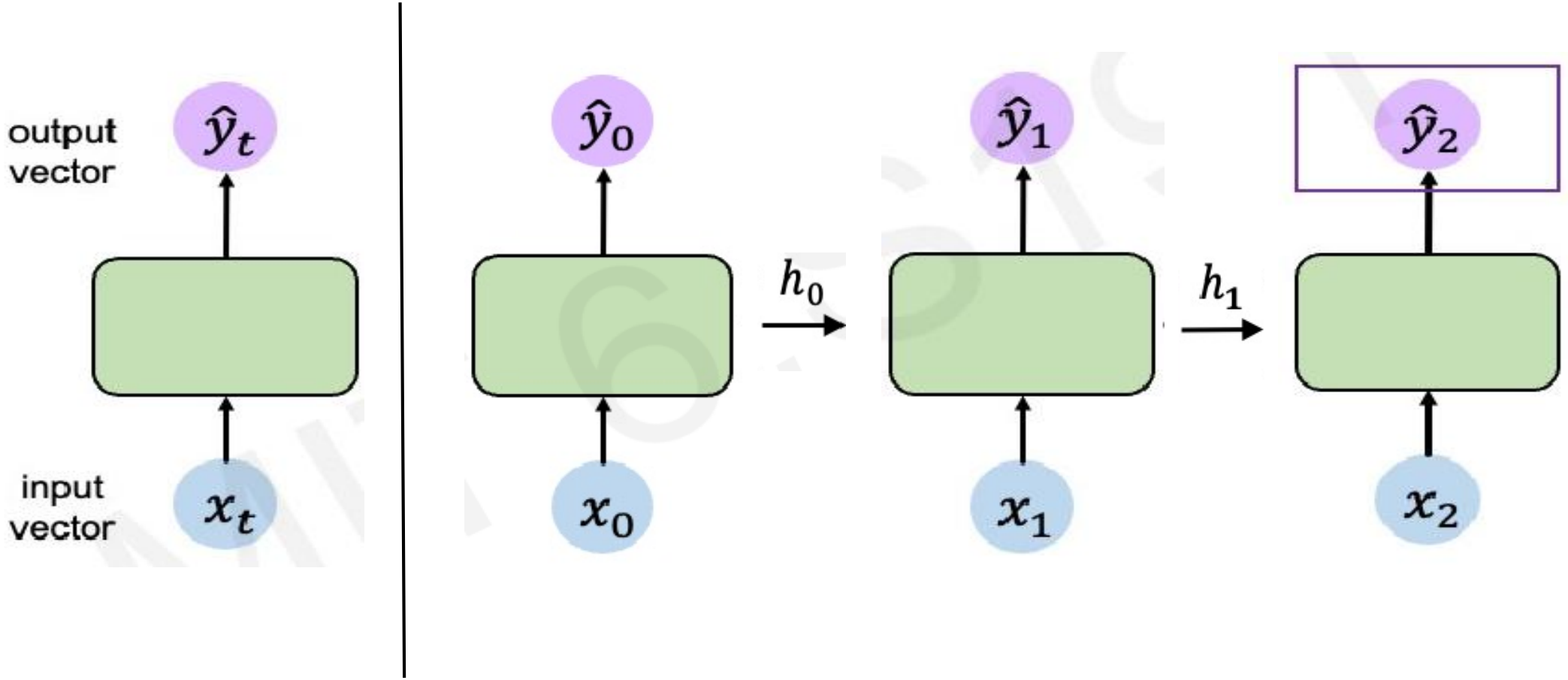


$$\boldsymbol{x_t} \in \mathbb{R}^m \qquad \hat{\boldsymbol{y}}_t \in \mathbb{R}^n$$

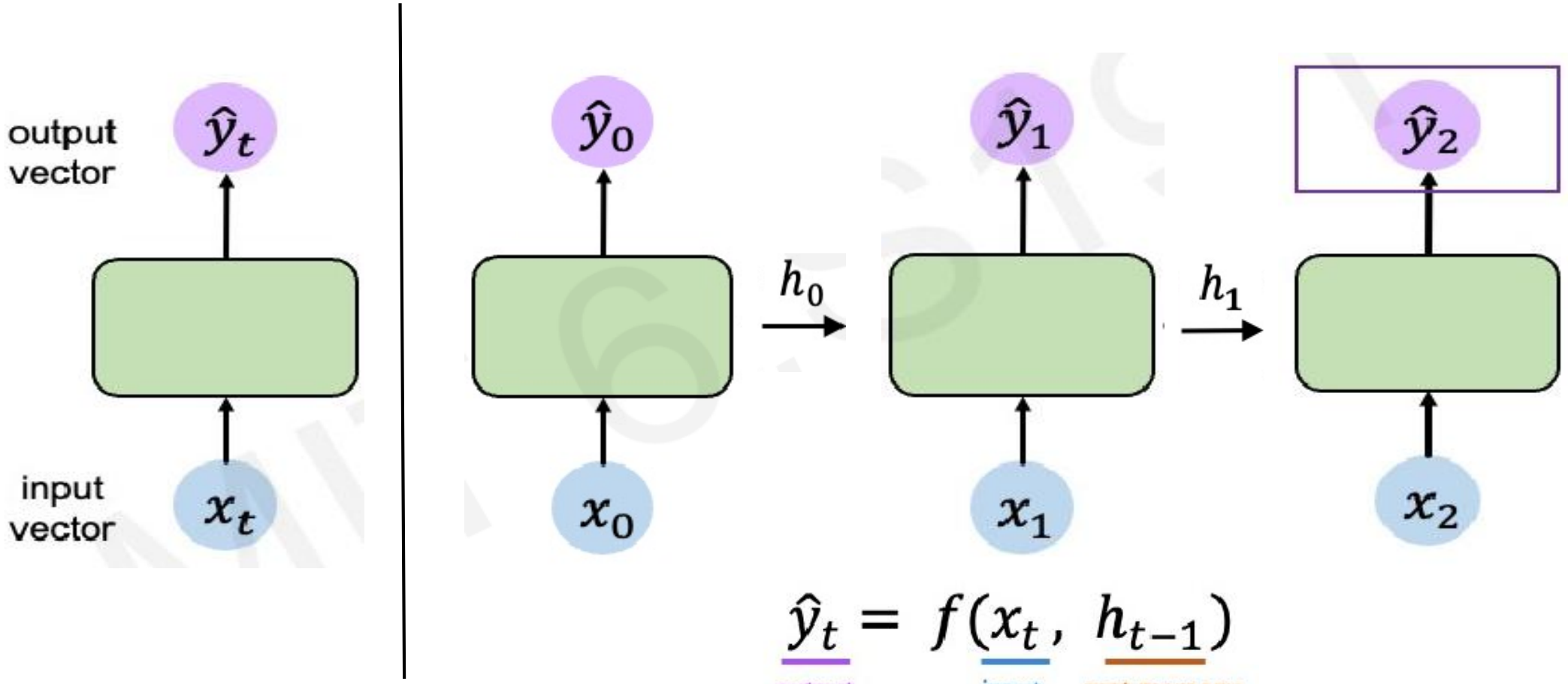
Handling Individual Time Steps



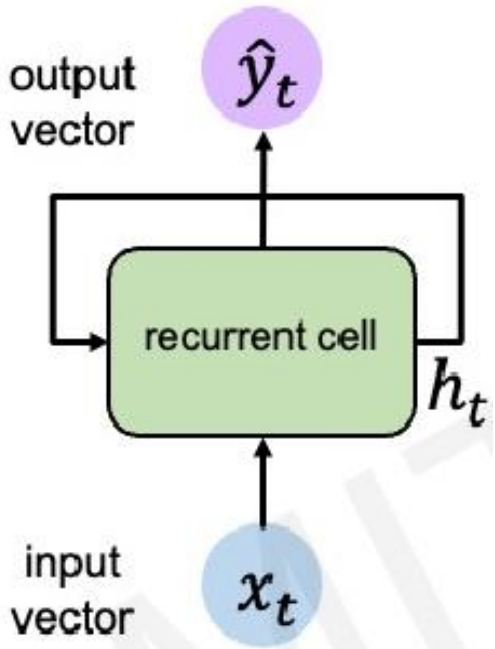
Neurons with Recurrence

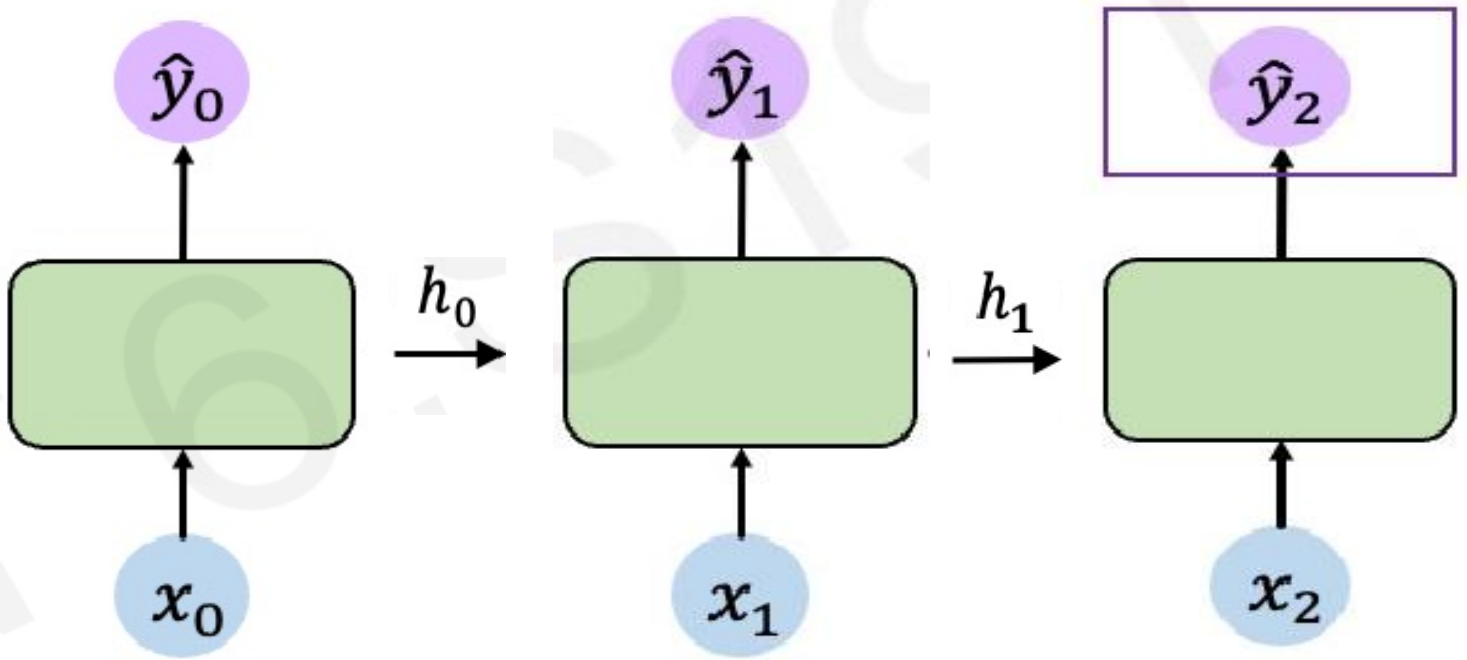


Neurons with Recurrence



Neurons with Recurrence





$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Recurrent Neural Networks

Why Do We Need RNNs?

Problems with previous methods:

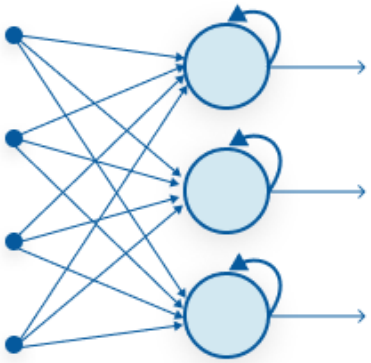
- **Frame-by-frame processing**: No memory of past frames.
- **Two-stream networks**: Require external motion data (optical flow algorithms).
- **3D CNNs**: Still lack long-term dependencies.

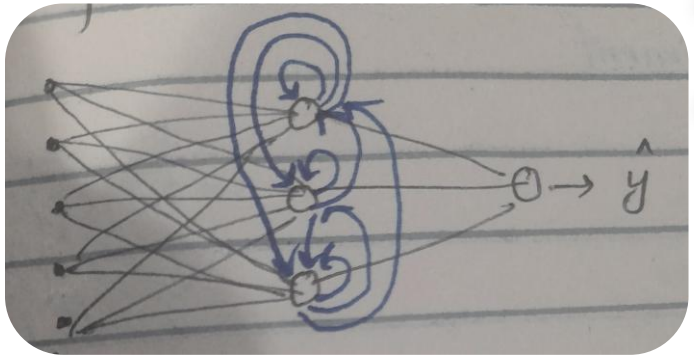
What Do We Need?

- A model that remembers past information and captures long-term dependencies.
- A model designed specifically for sequential data → **Recurrent Neural Networks (RNNs)**.

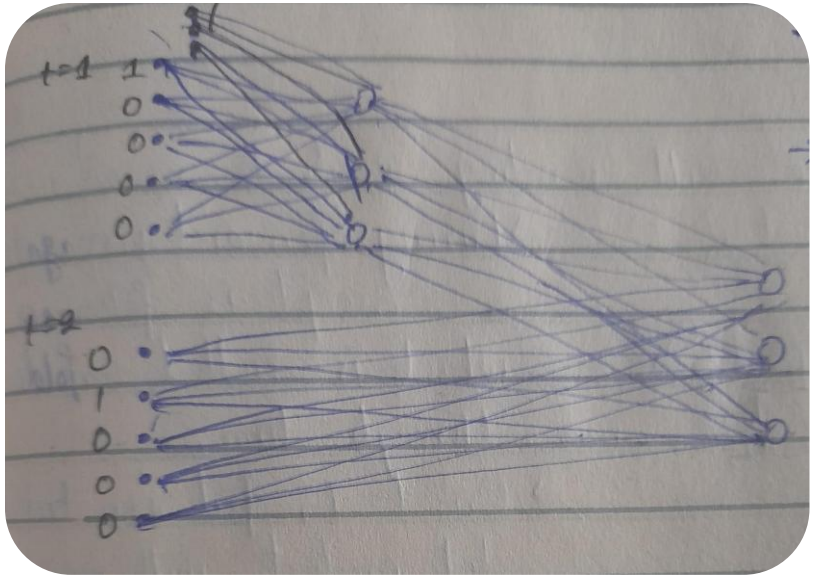
Introduction to Recurrent Neural Networks

- **What is an RNN?**
  - A class of neural networks designed for sequential data.
  - Unlike traditional neural networks, RNNs can process sequences of arbitrary length.
  - Uses a loop-like structure, where information is passed from one time step to next.





- This section covers:
  - ✓Recurrent neurons
  - ✓Memory cells
  - ✓Types of RNNs
  - ✓Training RNNs
  - ✓Challenges with RNNs

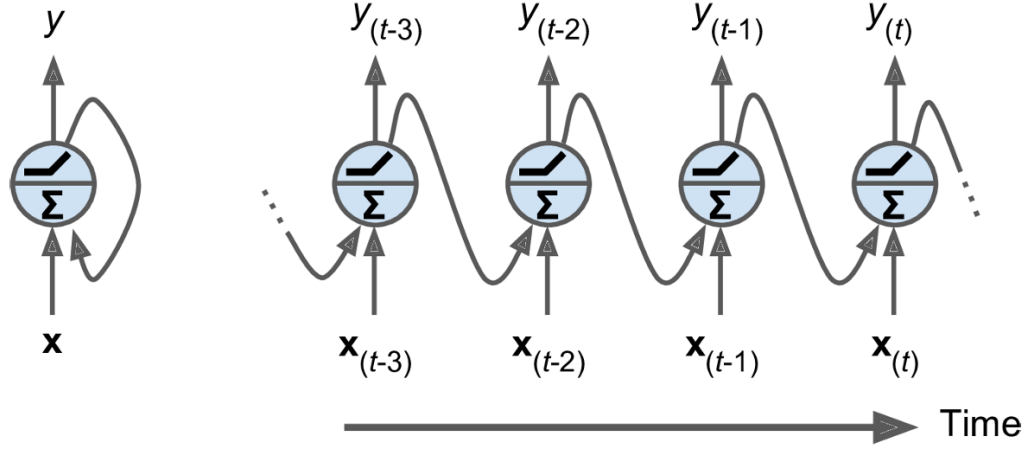


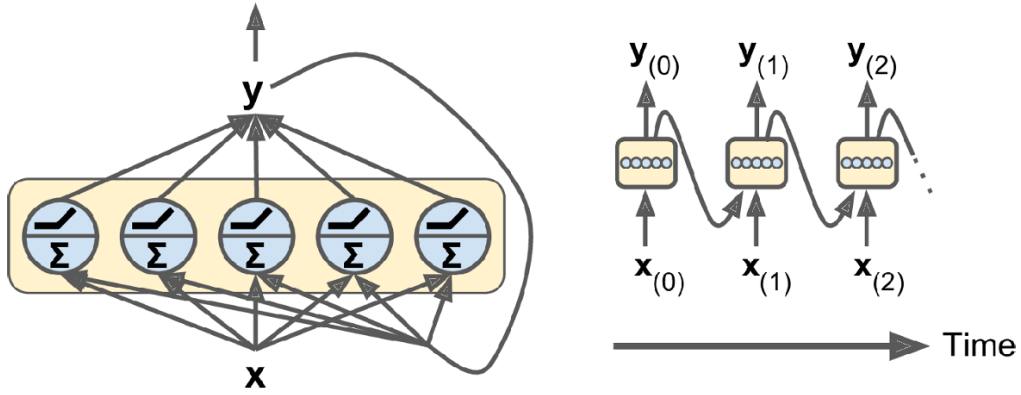
Recurrent Neurons

How Do Recurrent Neurons Work?

- Similar to standard neurons but with a feedback connection.
- At time step t, the neuron receives:
  - **Input** x_t from the current time step.
  - **Hidden state** h_{t-1} from the previous time step.
- These are combined using weights:
  - W_x → Weight for current input x_t
  - W_h → Weight for previous hidden state h_{t-1}.
- **Recurrent Layer Formation:**
  - A group of recurrent neurons forms a recurrent layer, enabling sequence modeling.
- **Equation:** $h_t = f(W_x x_t + W_h h_{t-1})$

where f is an activation function (typically *tanh* or *ReLU*).





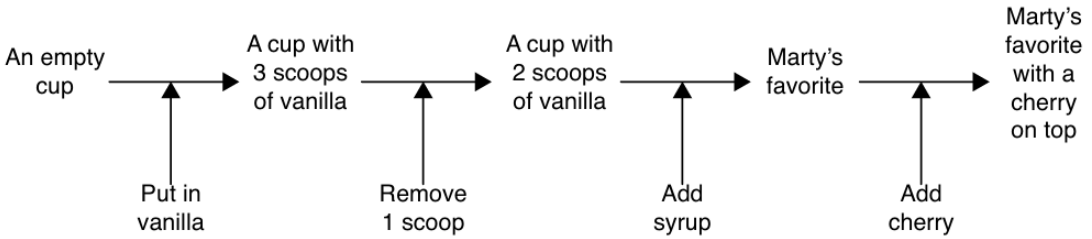
Memory Cells (Hidden State)

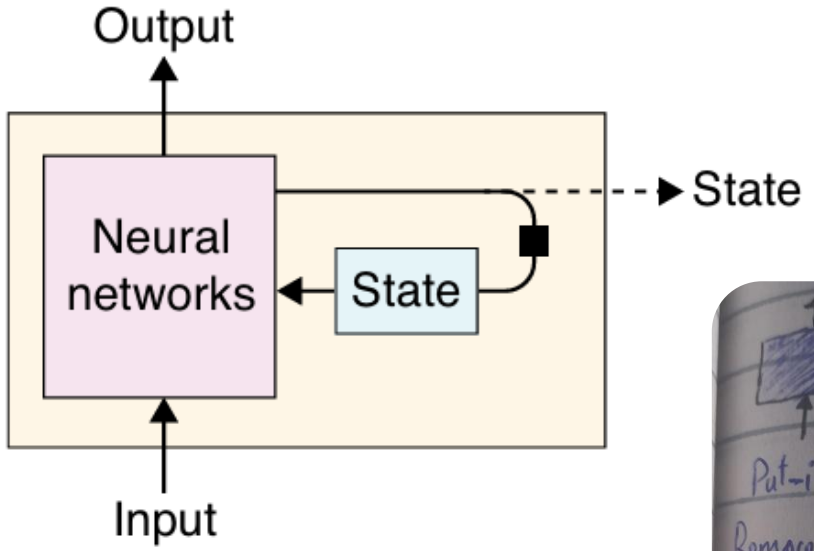
How does RNN store Information?

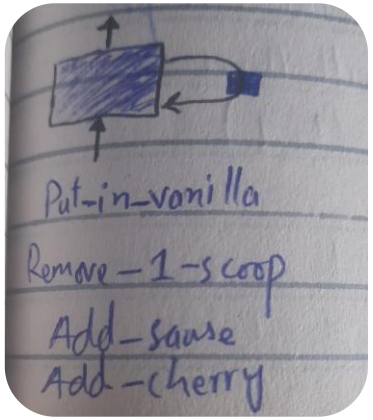
- RNNs have a hidden state (memory) that stores information from previous time steps.
- At any time step t, the hidden state is denoted as:

$$h_t = f(h_{t-1}, x_t)$$

- This allows RNNs to capture short-term dependencies
- **Limitations:**
  - Memory is limited: RNNs struggle to remember long-term dependencies.

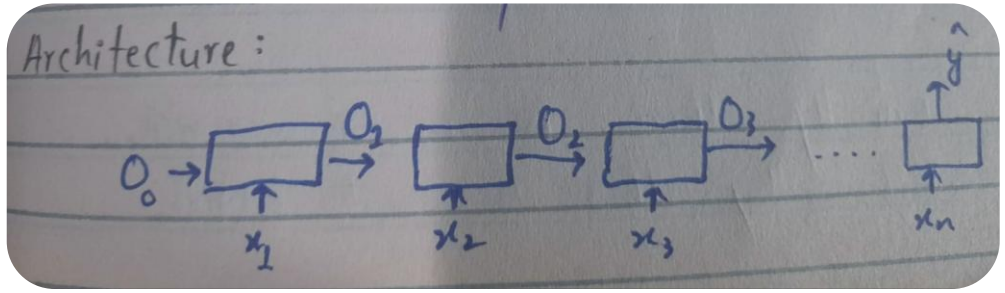




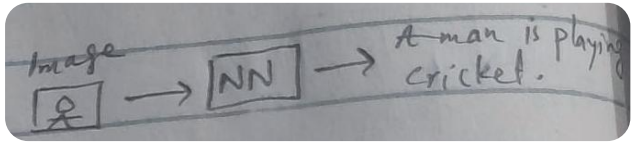


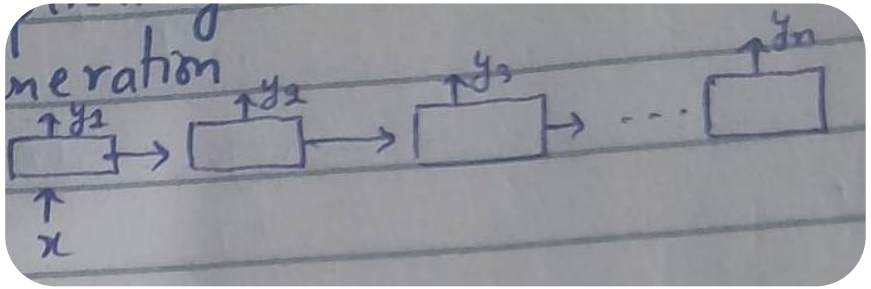
Types of RNN Architecture

1. **Many-to-One**: Inputs are sequences, output is a single value.
   - **Example:** Sentiment Analysis



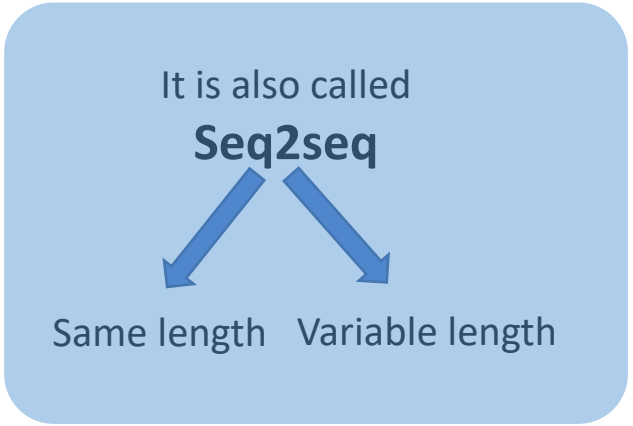
2. **One-to-Many**: Input is a single value, output is a sequence (e.g., music generation).
   - **Example:** Image Captioning, Music Generation

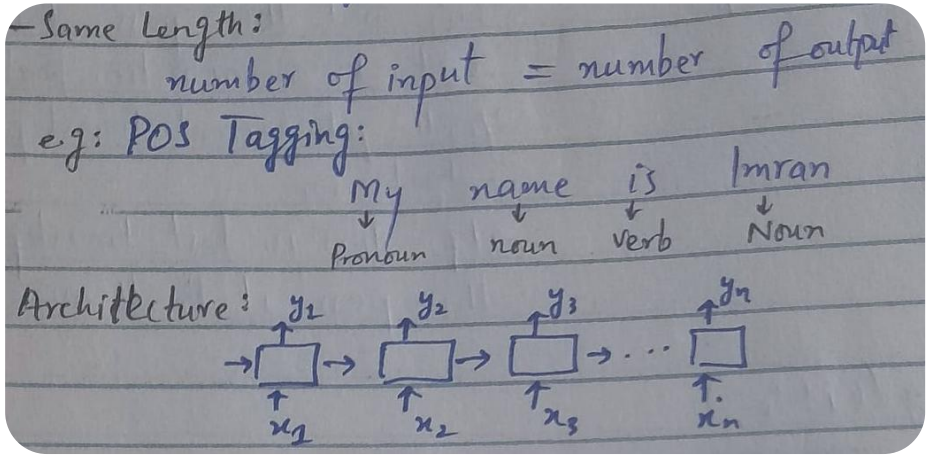


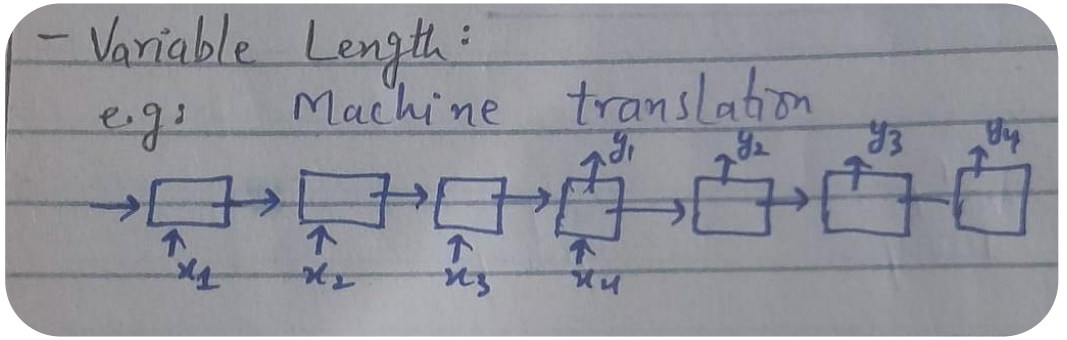


Types of RNN Architecture

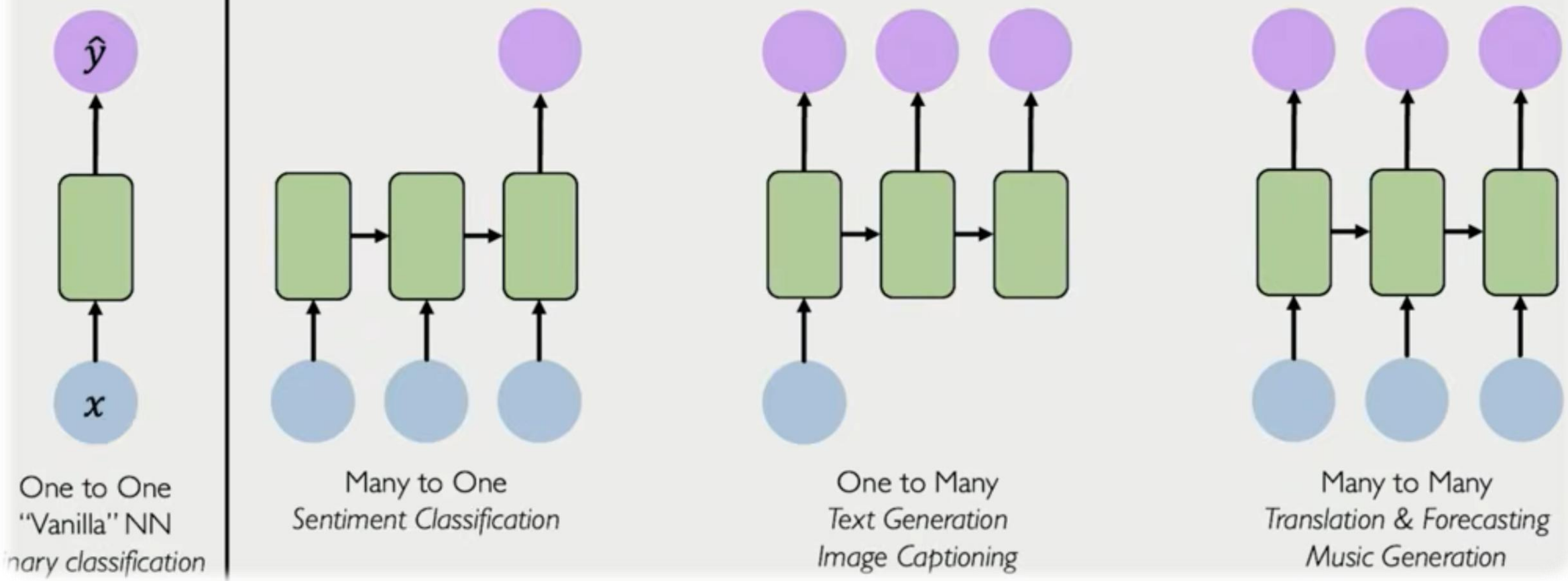
3. **Many-to-Many (Sequence-to-Sequence):** Inputs and outputs are sequences.







RNNs for Sequence Modeling



Training RNNs – Backpropagation Through Time (BPTT)

How RNN Training Works:

1. **Unroll the network:** Expand the recurrent loop into a sequence of layers over time.

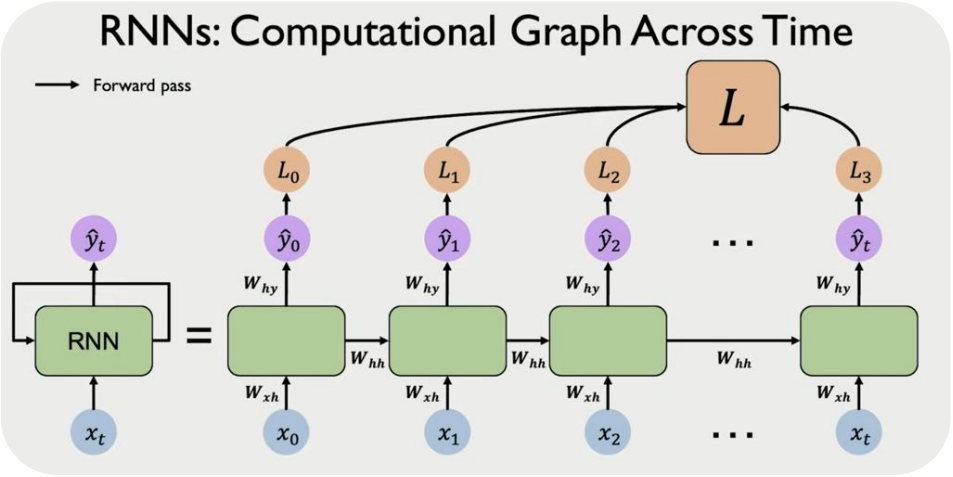
2. **Forward Propagation:** Process the input sequence step by step.

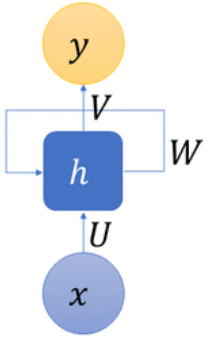
3. **Loss Calculation:** Measure the error at each step (for sequence outputs) or only at the last step (for a single final output), depending on the task.

4. **Backpropagate Errors:** Propagate errors backward through all time steps.

5. **Compute gradients:** Calculate gradients for all parameters across time steps.

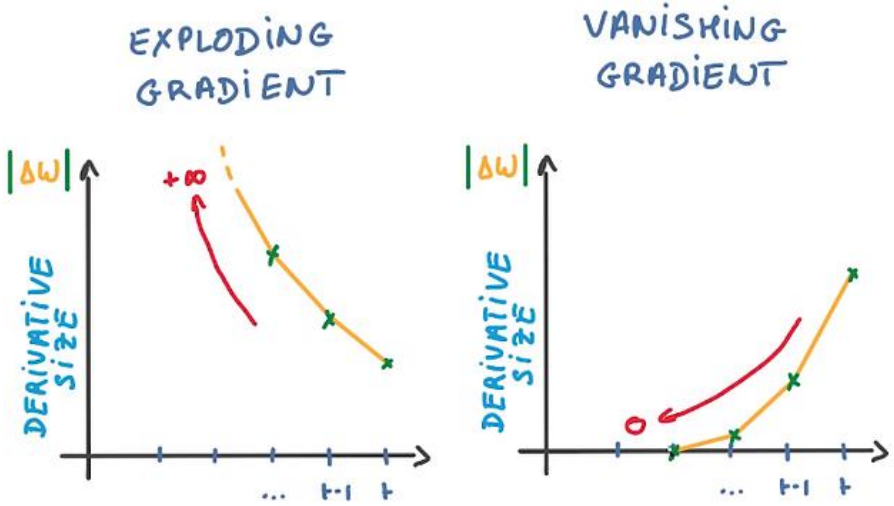
6. **Update weights:** Adjusts weights and biases using gradient descent (or variants).





Challenges of RNNs

- **Unstable Gradients:**
  1. **Vanishing Gradient Problem:** Gradients become too small, making it hard to learn long-term dependencies.
  2. **Exploding Gradient Problem**: Gradients become too large, causing instability in training.
- **Short-Term Memory Issue:**
  - As the sequence length increases, RNNs tend to forget earlier inputs.
  - The hidden state has a fixed size, limiting how much past information can be retained.



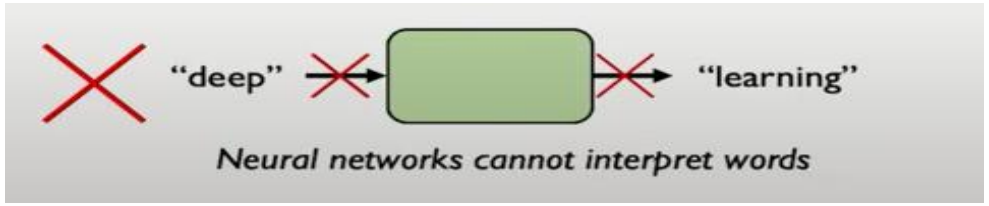
A sequence Modeling Problem: Predict the Next World

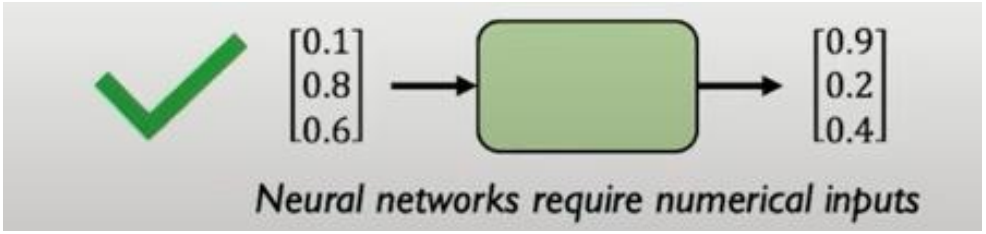
A sequence Modeling Problem: Predict the Next

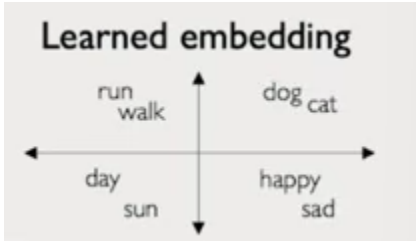
"This morning I took my cat for a walk."

Given these words Predict the next word

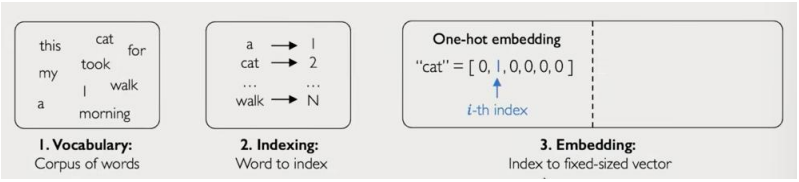
Representing Language to a Neural Network







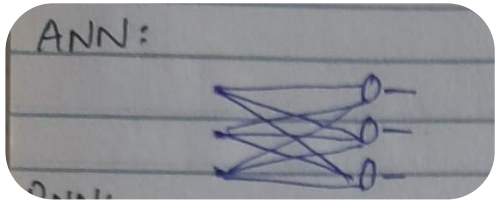
Embedding: transform indexes into a vector of fixed size.



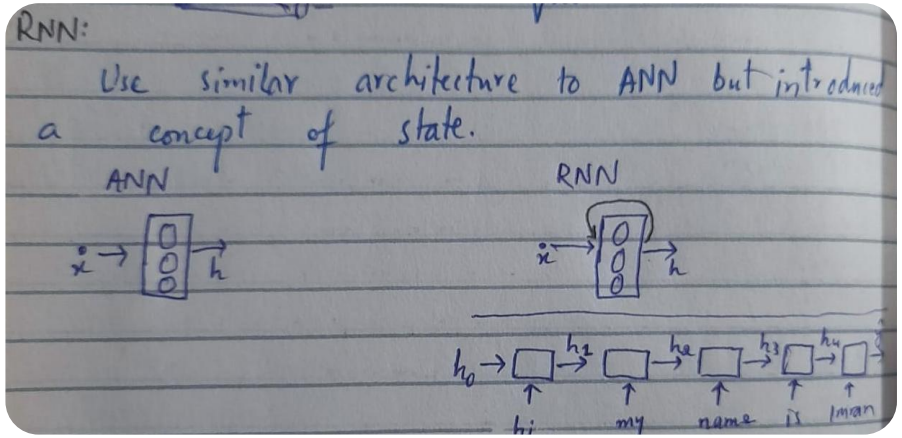
Long Short-term Memory

Introduction to LSTM

- **Why Do We Need LSTMs?**
  - Standard RNNs suffer from short-term memory and struggle with long-term dependencies due to vanishing gradients.
  - **LSTMs** (Long Short-Term Memory networks) solve this by introducing a structured memory mechanism.



Problem: Not working with sequential data

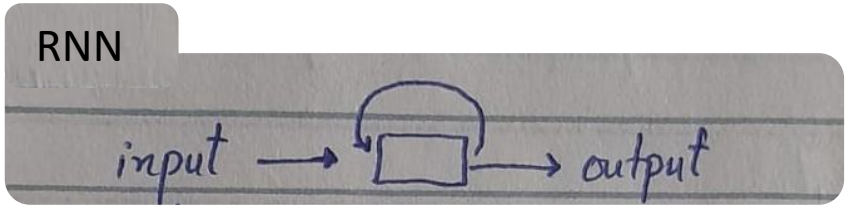


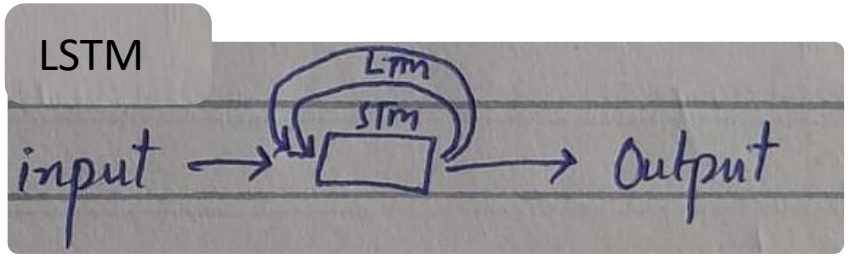
Problem: When the chain in an RNN gets too long, the prediction at the end tends to forget the initial inputs, so those inputs have little effect on outputs far away in the sequence.

- **Key Features of LSTMs:**
  - Designed to capture **long-term dependencies**.
  - Introduces two types of memory:
    - **Short-term state** (h_t)
    - **Long-term state** (c_t)
  - Uses **gates** to control information flow.

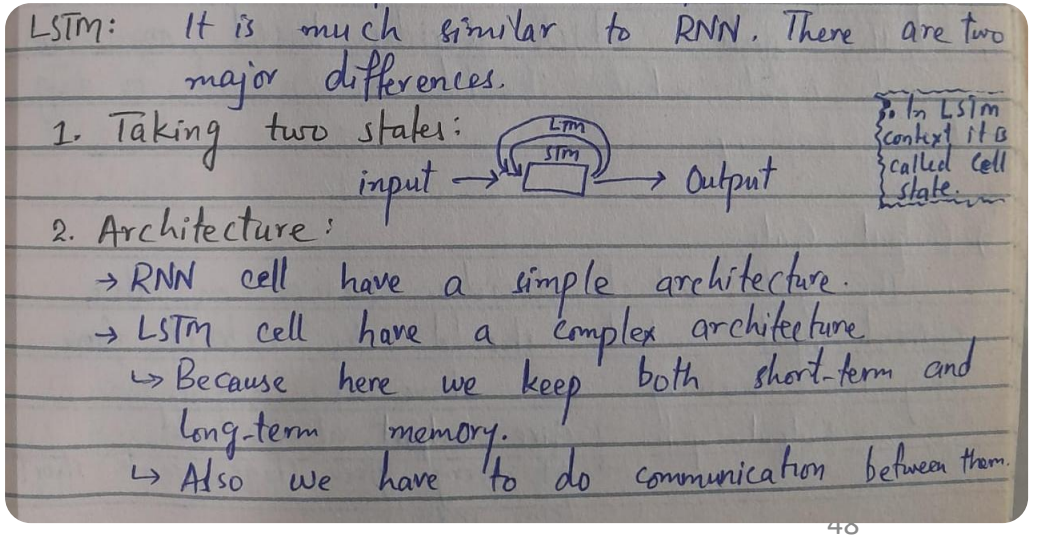
LSTM – Core Idea

- In a basic RNN, there's only one hidden state path to retain past information.
- This single path must handle both short-term and long-term context.
- In practice, the short-term context often dominates, making it hard to preserve long-term dependencies.



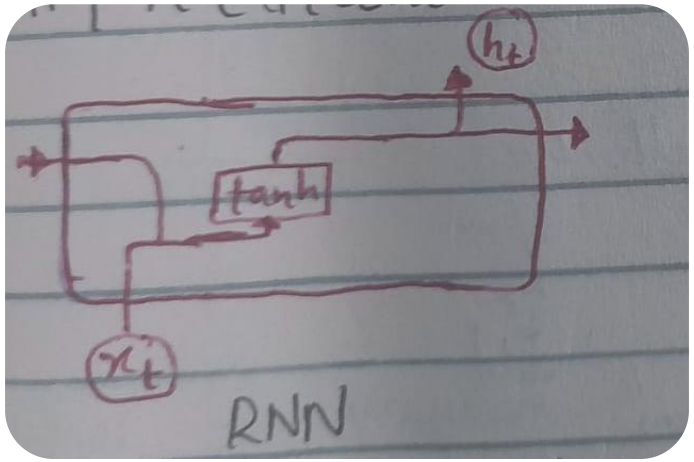


- Researchers introduced an additional path in LSTM:
  - Short-term memory path (hidden state)
  - Long-term memory path (cell state)
- Now, if something is considered important early on, it can persist until explicitly removed by the network's gates.
- In LSTM, the cell state acts as a dedicated path for long-term memory, separate from the short-term path.

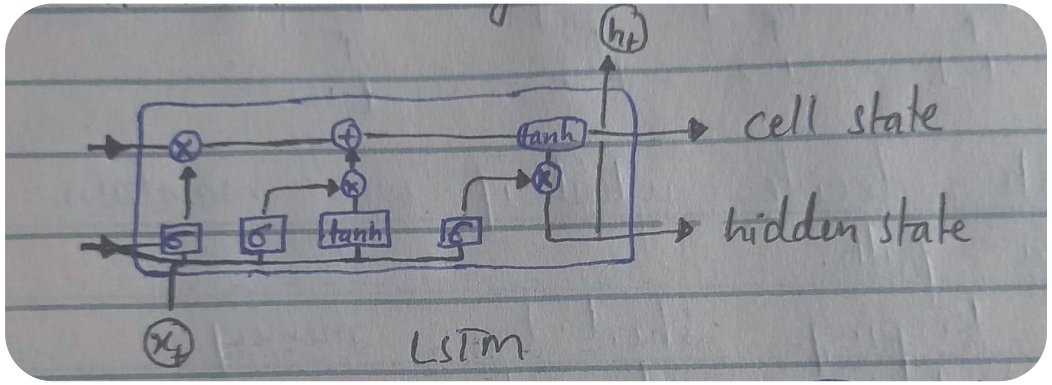


LSTM Cell Architecture

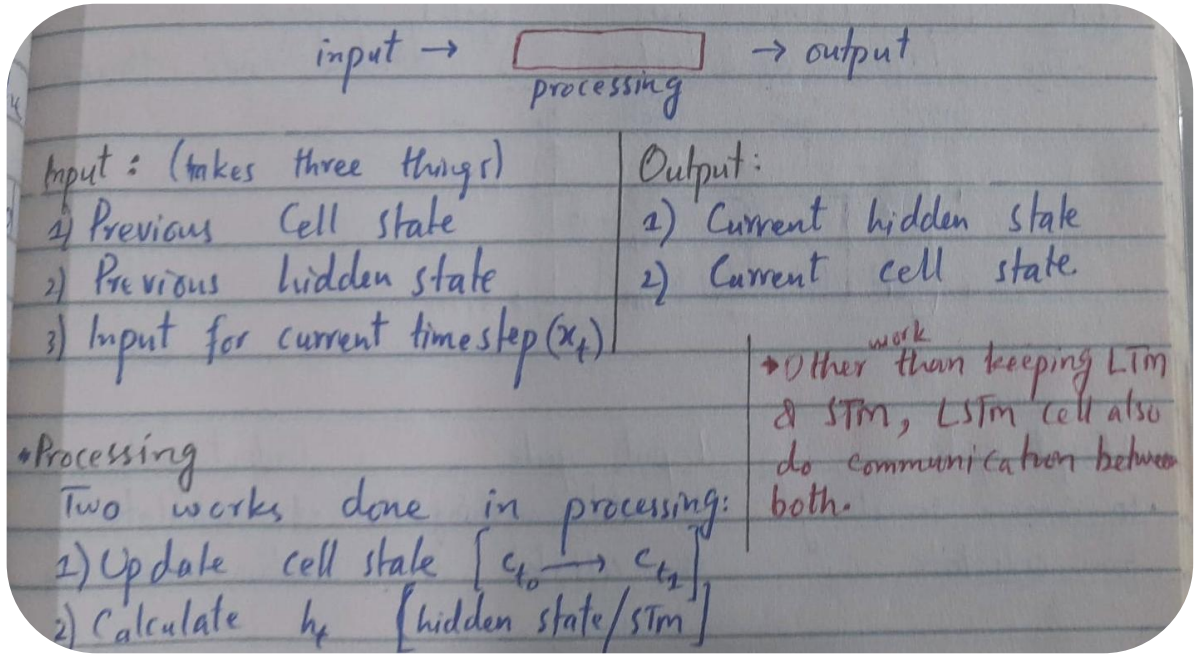
- The basic RNN cell processes current input (x_t) and previous hidden state (h_t) through a *tanh* activation function.
- The complex design of an LSTM cell is because of, to enable communication between long-term and short-term memory.
- LSTM maintains two types of context:
  - **LTM (Long-Term Memory):** called the *cell state*
  - **STM (Short-Term Memory):** called the *hidden state*

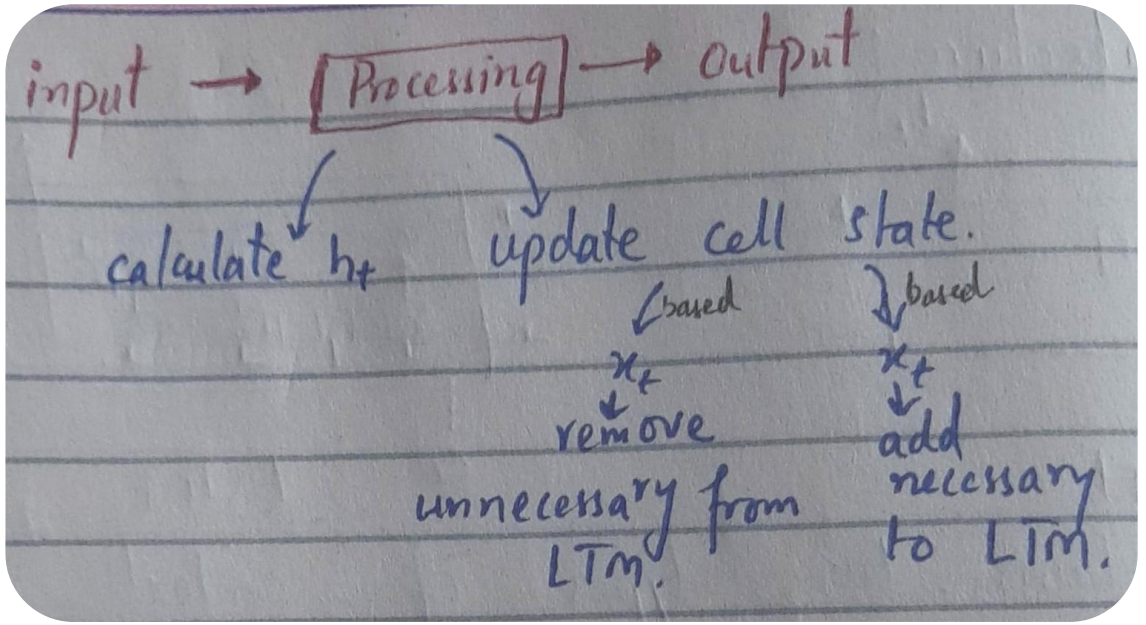


- **Number of Neurons:**
  - Each gate has the same number of neurons = hidden_size (your choice)
  - All 4 gates in one cell must have identical size
  - **Example:** If hidden_size = 128, each gate has 128 neurons
- **Architecture:**
  - LSTM cell contains 4 separate layers (3 gates + candidate)
  - **Within a cell:** All layers have same neuron count
  - Across stacked cells: Sizes can vary (but usually kept same)



LSTM Cell Architecture





LSTM Cell Architecture

LSTM Cells:

- Unlike standard RNN neurons, LSTM cells **maintain a long-term state** (c_t).
- Each LSTM cell consists of **four fully connected layers**, each serving a specific function.

• Components of an LSTM Cell:

- **Forget Gate**: Decides based on the current input and short-term context, which past information should be discarded.

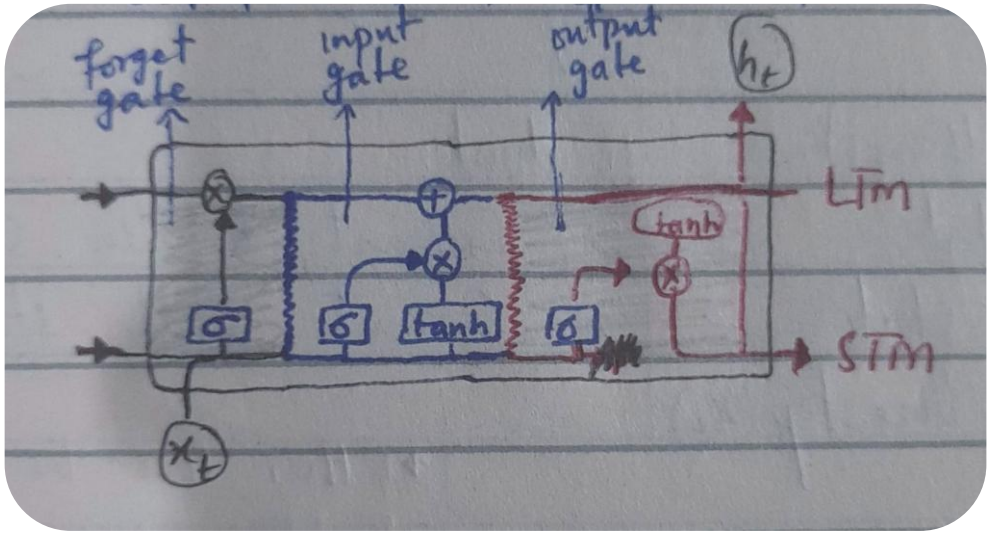
  $\boldsymbol{f}_t = \sigma(W_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f)$

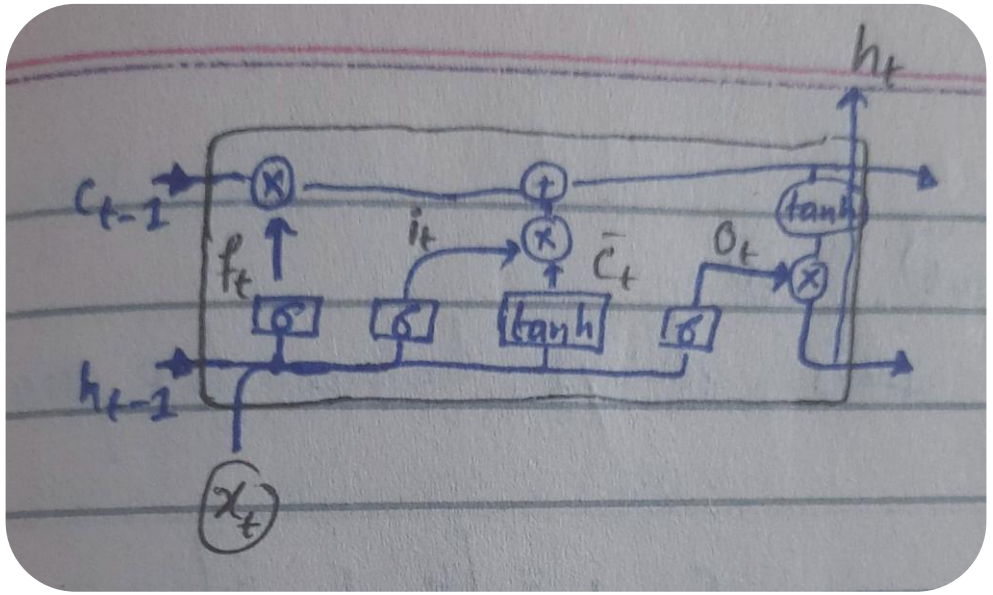
- **Input Gate**: Controls how much of the new input should be added to memory.

  $\boldsymbol{i}_t = \sigma(W_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i)$

- **Output Gate**: Controls which part of the long-term state should be read and output at this time step.
  - Provide output at the end
  - At every given stage, create short term memory

  $\boldsymbol{o}_t = \sigma(W_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o)$

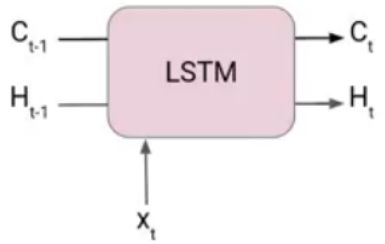


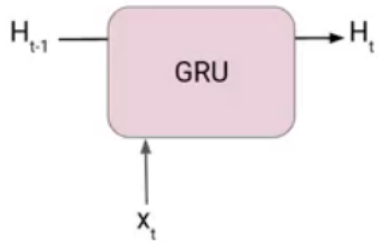


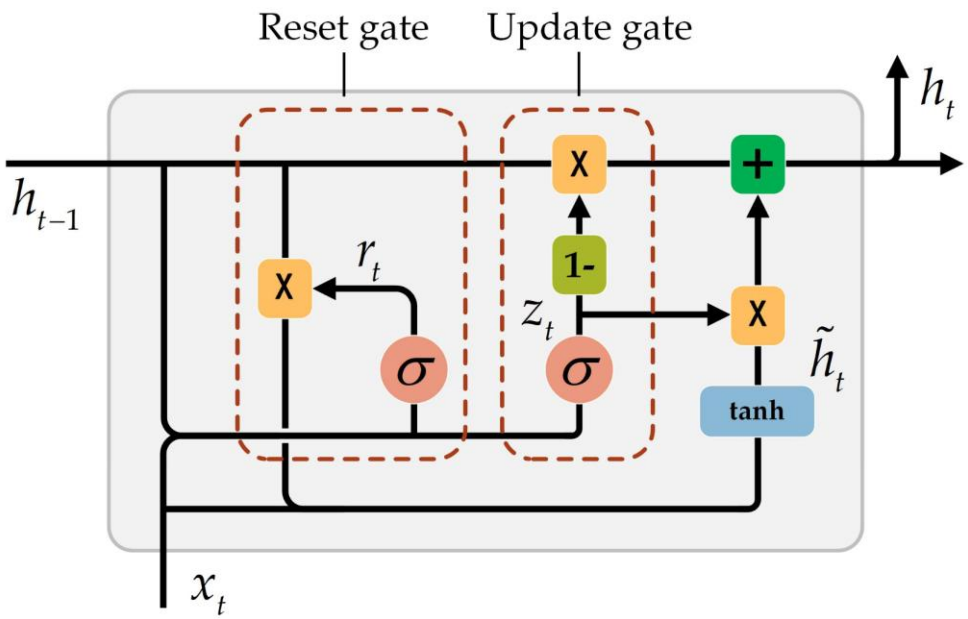
Gated Recurrent Unit

Gated Recurrent Unit

- **Developed By:** Kyunghyun Cho et al., 2014
- **Why It Was Developed:**
  - Addresses vanishing gradient issues in standard RNNs
  - Provides a simplified alternative to LSTMs with fewer parameters
  - Suitable for sequence-based tasks such as NLP and time series forecasting.
- **Use Cases**
  - Speech Recognition
  - Machine Translation
  - Time Series Forecasting
  - Text Generation



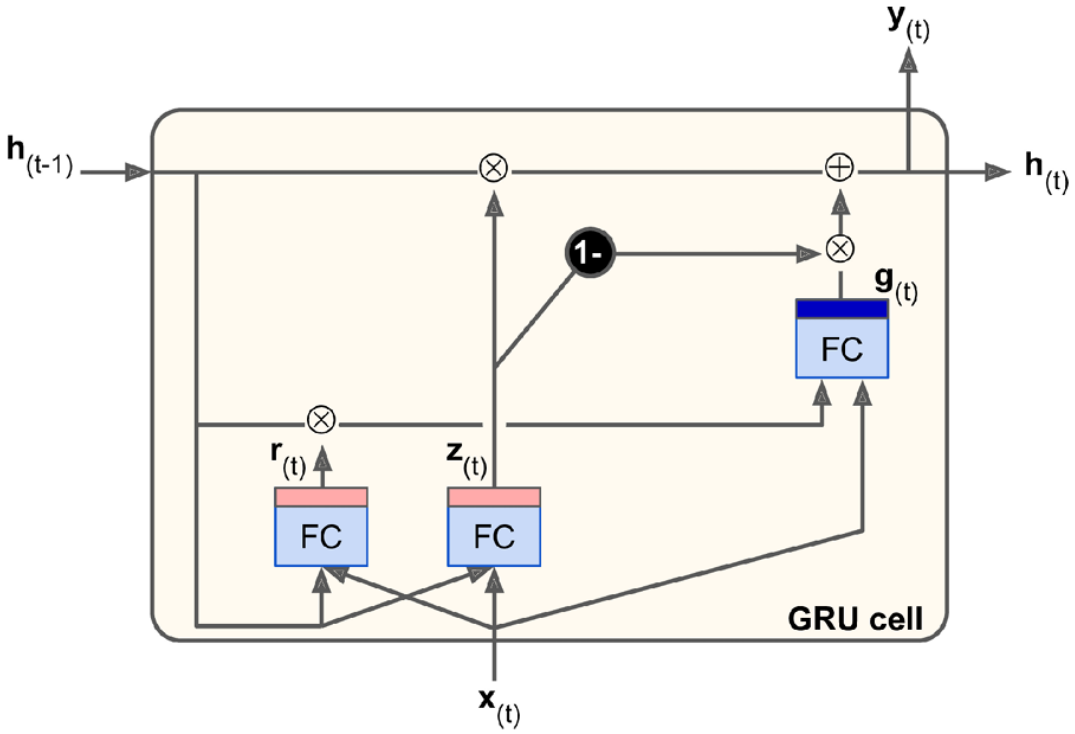




Gated Recurrent Unit

How GRU Works

- A **simplified version of LSTM**, merging cell state and hidden state into a single vector
- Uses **two gate controllers:**
  1. **Update Gate** – Combines the functions of LSTM's forget and input gates, determining how much of the past state should be retained
  2. **Reset Gate** – Controls how much past information should be forgotten before passing the current state to the main layer
- **Key Difference from LSTM**



Feature	LSTM	GRU
Gates	Forget, Input, Output	Update, Reset
Cell State	Maintains separate cell state	Merges with hidden state
Complexity	More parameters, slower training	Fewer parameters, faster training
Performance	Better for long sequences	Efficient for shorter sequences

Combining CNNs and RNNs: Leveraging Spatial and Temporal Features

Why Combine CNN and RNN?

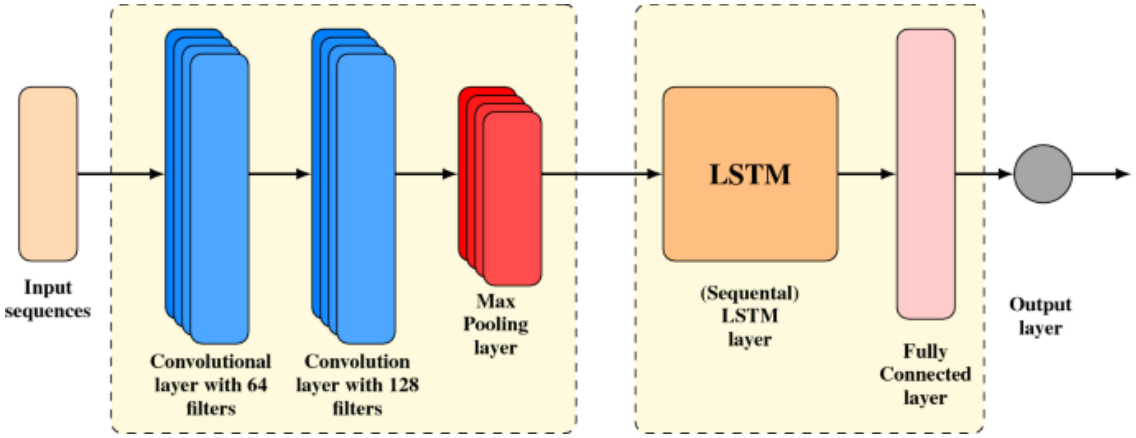
- **CNNs** are excellent at extracting **spatial features** from images and frames.
- **RNNs** (or LSTMs/GRUs) excel at capturing **temporal dependencies** in sequences.
- By **combining them**, we can create models that learn **rich representations** from data.

How It Works:

- **CNN extracts spatial features** from input data (e.g., image frames from a video).
- **RNN processes sequential features** from CNN outputs to model **temporal dependencies**.

Applications:

- **Image Captioning** → CNN extracts image features, RNN generates text descriptions.
- **Video Analysis** → CNN processes each frame, RNN models the sequence for action recognition.
- **Speech Recognition** → CNN extracts acoustic features, RNN deciphers sequential patterns.



Do you have any Questions?

Thank You