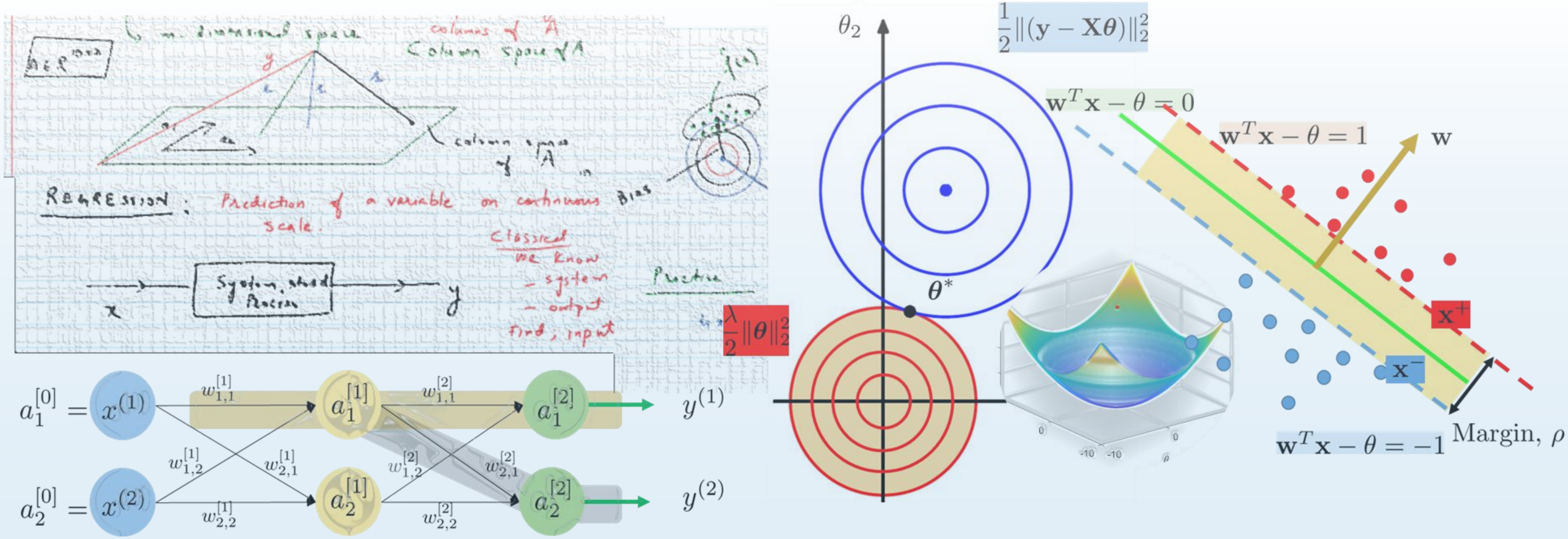


Machine Learning

Recurrent Neural Networks (RNNs) and LSTM Networks

School of Science and Engineering

https://www.zubairkhalid.org/ee514_2025.html

LUMS
A Not-for-Profit University

Outline

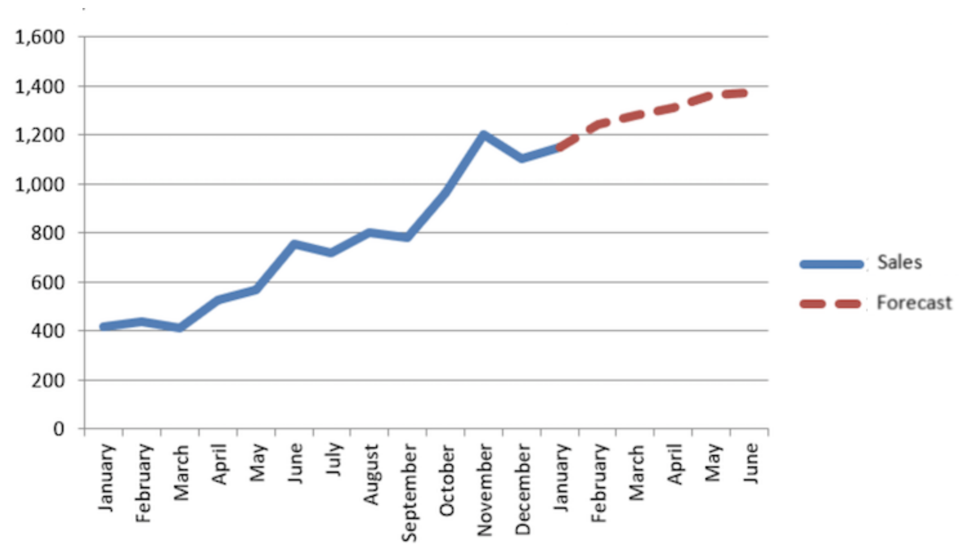
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory (LSTM) Networks
    - Gated Recurrent Units

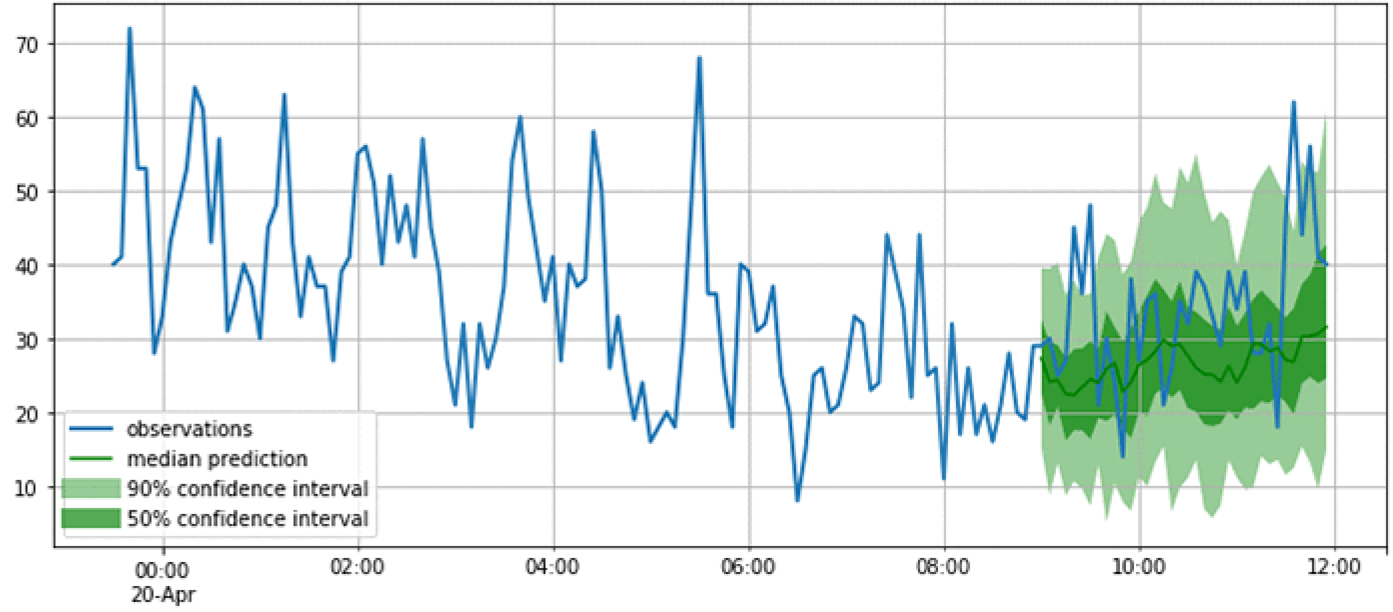
Recurrent Neural Networks

Sequence Modeling:

Predicting or Generating sequences of data
by capturing
patterns and dependencies **over time.**

Given current and past values, determine the next and future values.





Recurrent Neural Networks

Sequence Modeling – Applications:

Speech Recognition

→ Sequence Model → *I love oranges*

Image Captioning

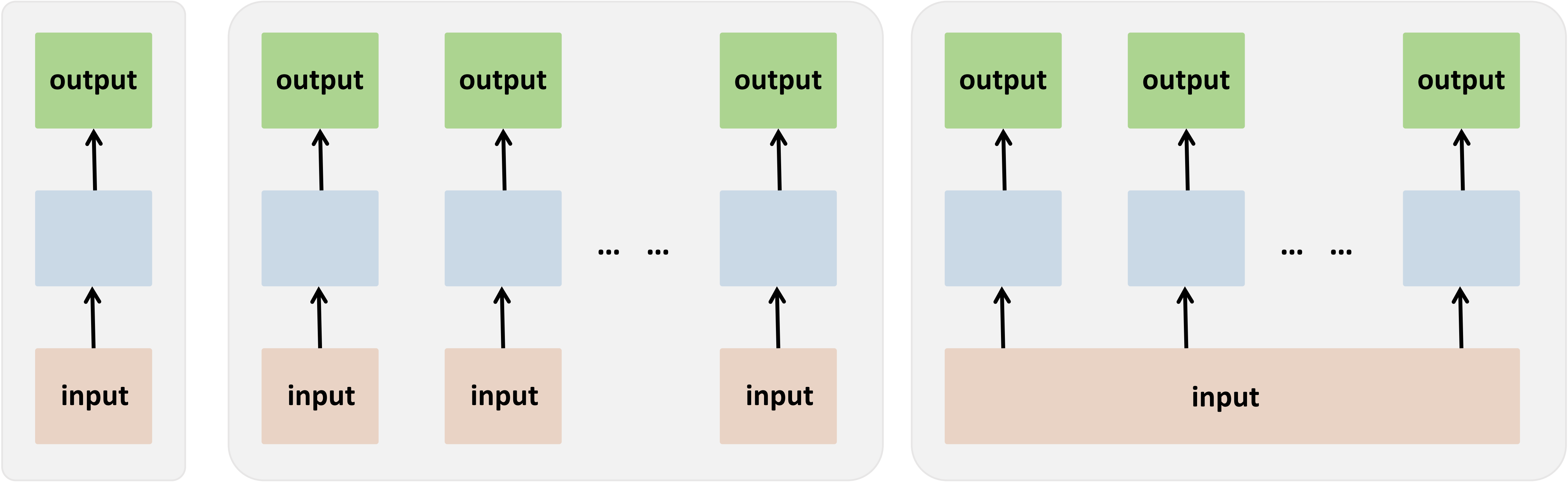
→ Sequence Model → *Two dogs are playing with a ball*

Subtitle Generator

→ Sequence Model → *How you doin?*

Recurrent Neural Networks

Feedforward Neural Network:

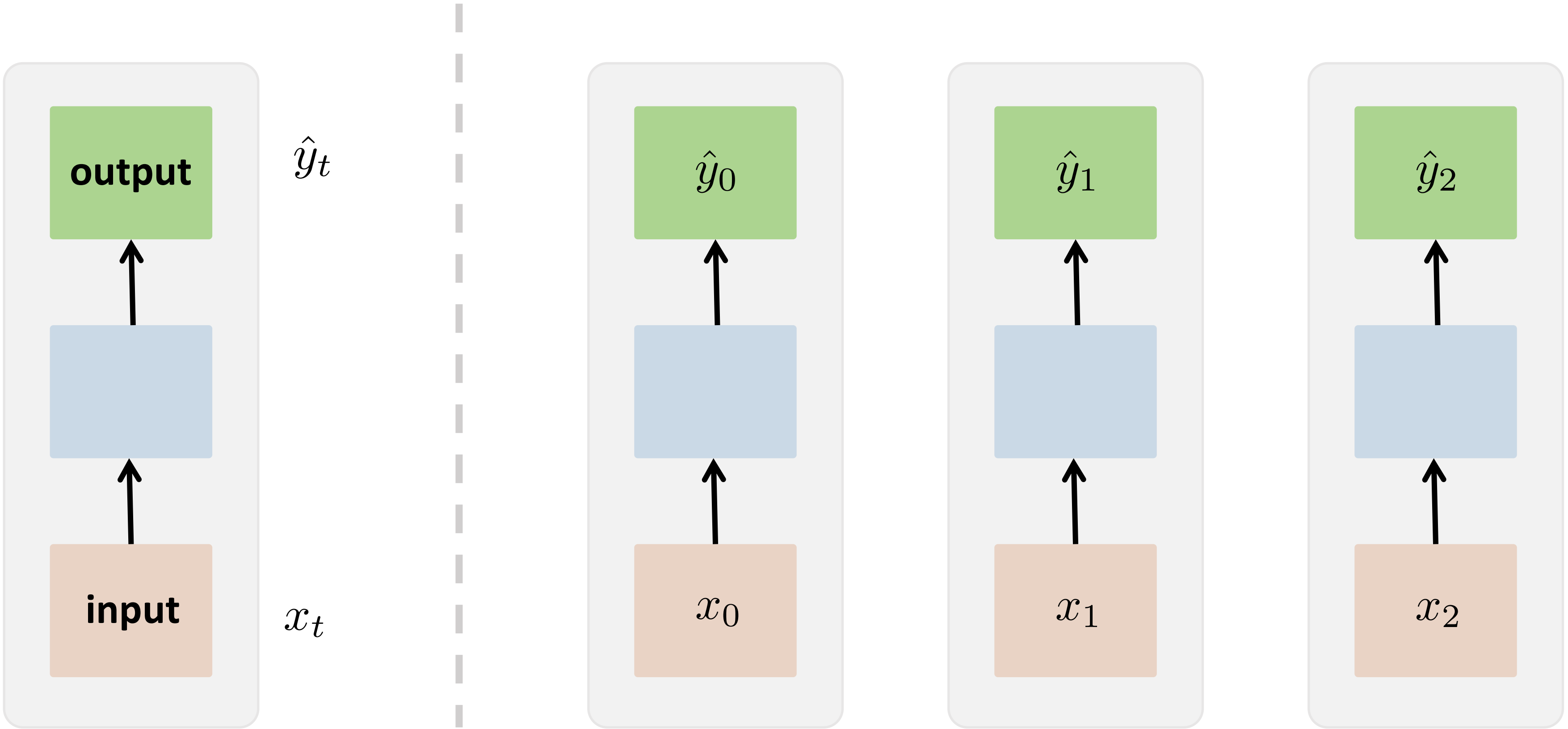


Key Idea: *Output of one layer serves as input to the next layer.*

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) – Concept:

Feedforward Network:



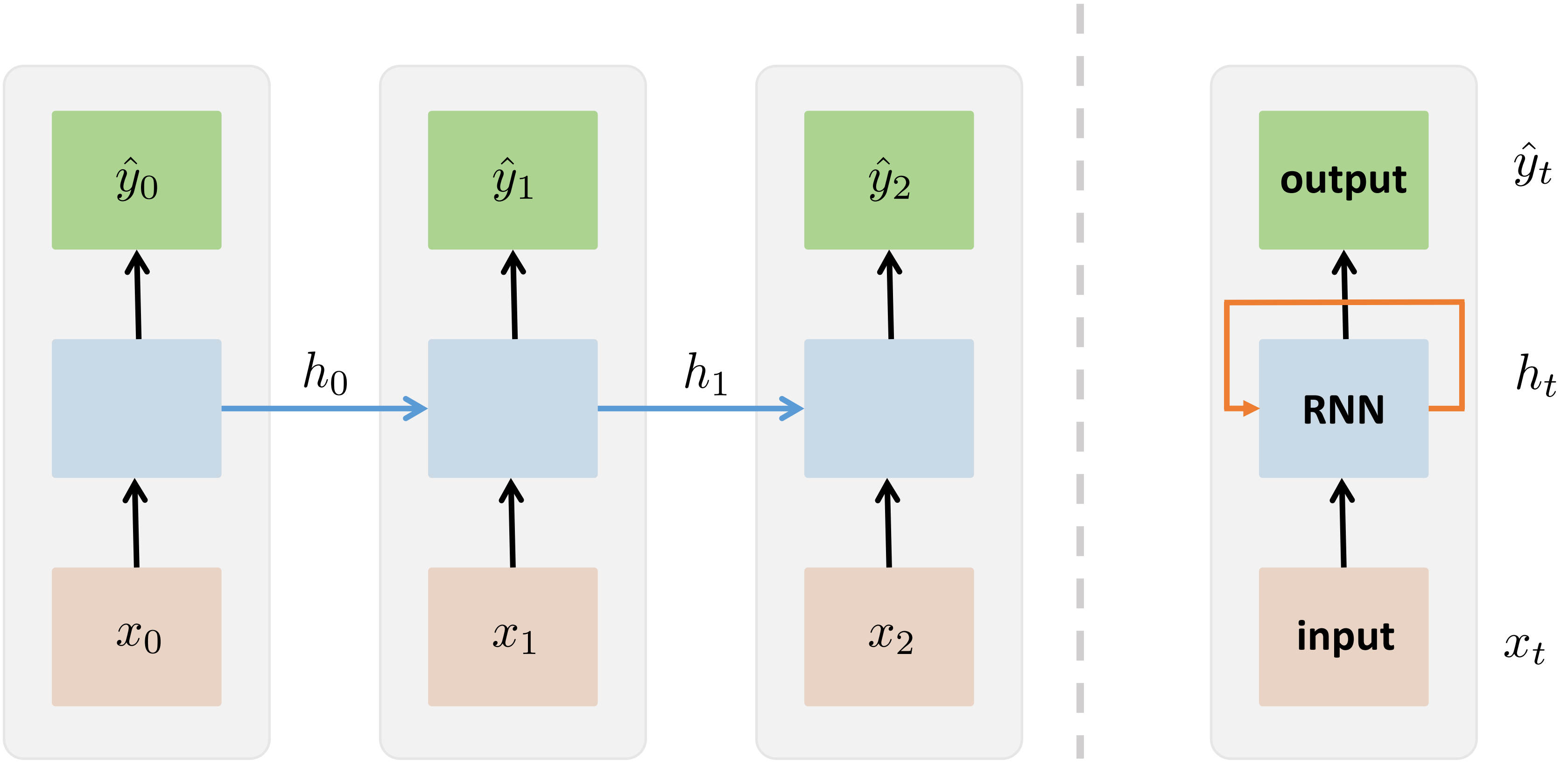
$$\hat{y}_t = f(x_t)$$

Key Idea: Output at time t depends on input at time t.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) – Concept:

Recurrence: Past inputs are captured by the state h_t, that is the output of the recurrent cell.



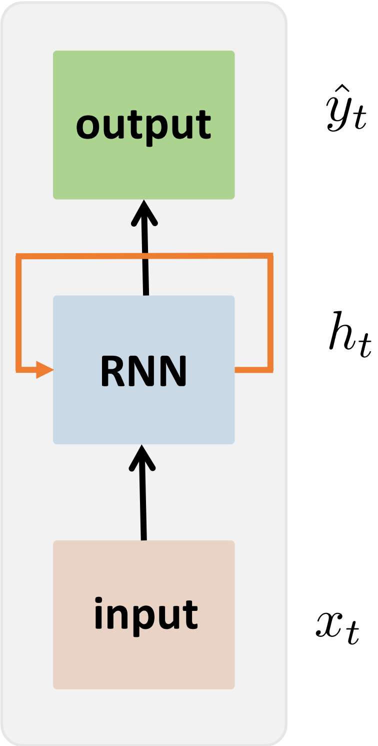
$$\hat{y}_t = f(x_t, h_{t-1})$$

Key Idea: Output at time t depends on input at time t and past inputs.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) – Concept:

Recurrence: Past inputs are captured by the state h_t, that is the output of the recurrent cell.



$$\boxed{h_t} = f_W(\boxed{x_t}, \boxed{h_{t-1}})$$

new state

input at time t

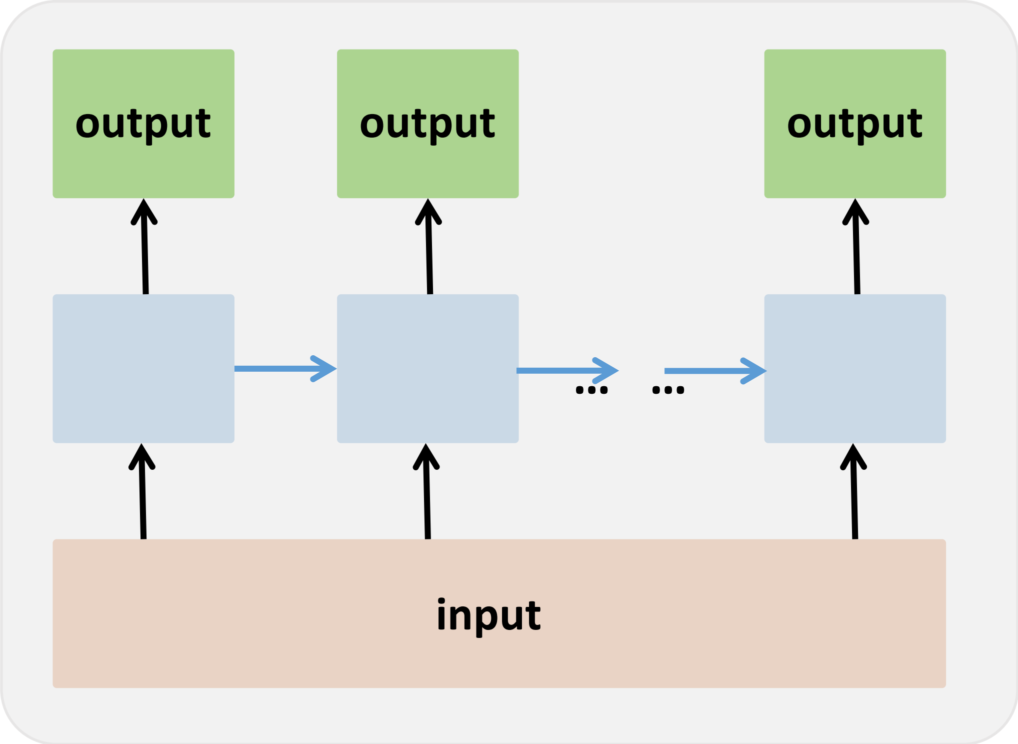
old state

f_W: Functional mapping characterized by some parameters W

f_W: independent of time, that is, we have same W and f for each time step

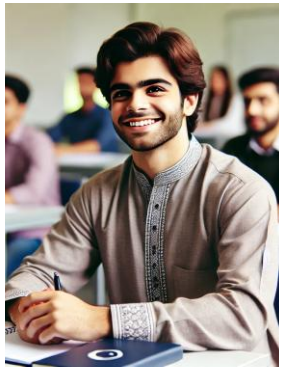
Recurrent Neural Networks

RNNs – One to Many:



Applications:

Captioning an Image
- *Input: Image*
- *Output: Sequence of Words*

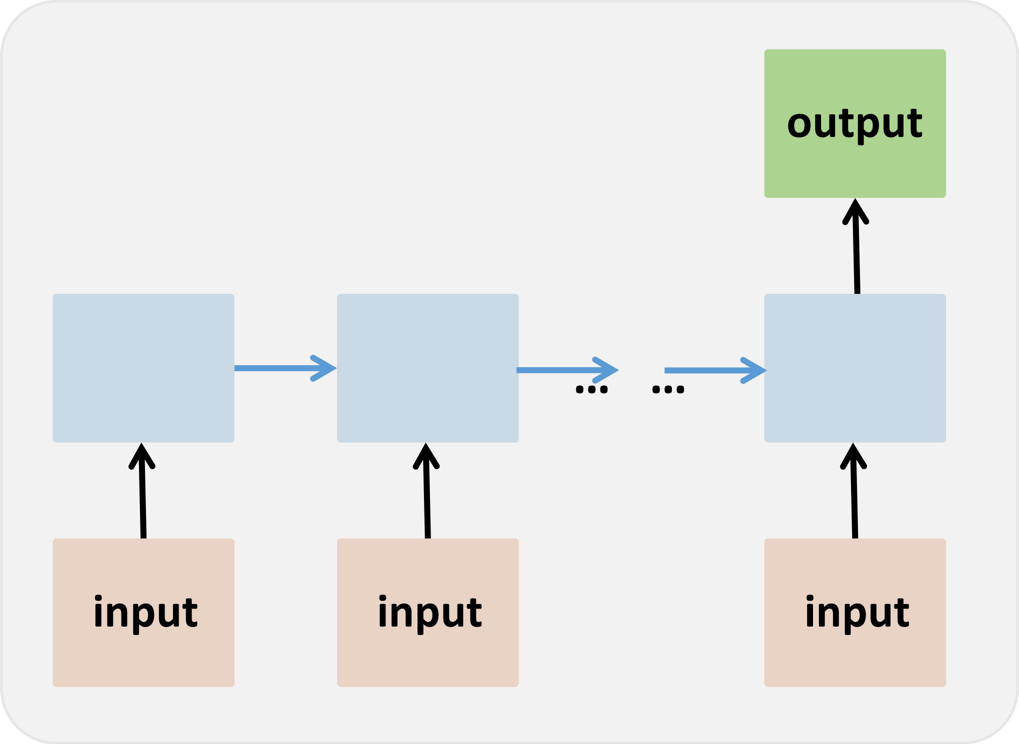


→ *Student in Machine Learning Class*

Key Idea: Output of one layer serves as input to the same layer.

Recurrent Neural Networks

RNNs – Many to One:



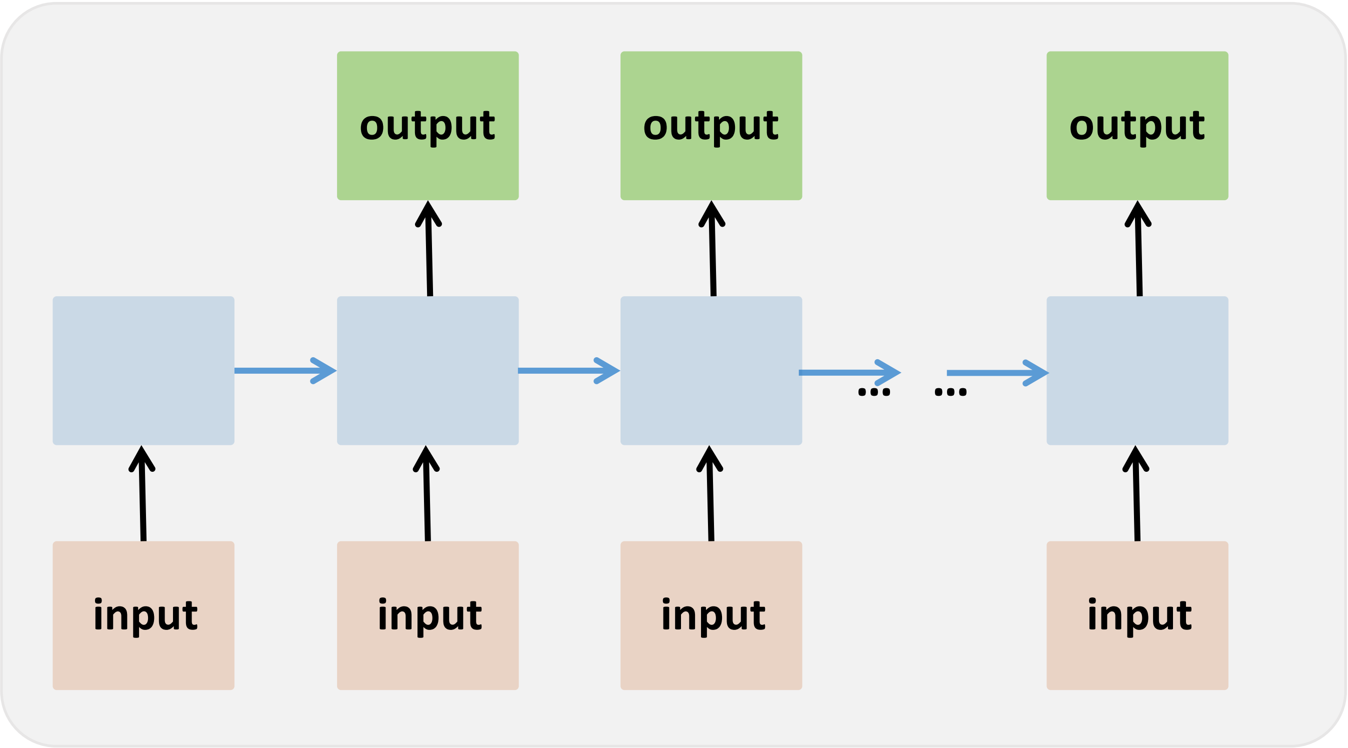
Applications:

Sentiment Analysis
- *Input: Tweet (Sequence of words)*
- *Output: Sentiment*

Key Idea: Output of one layer serves as input to the same layer.

Recurrent Neural Networks

RNNs – Many to Many:



Applications:

Auto-tweet or Translation
- Input: Tweet
- Output: Sequence of Words

Object tracking in Video
- Input: Frames of Video
- Output: Object position in a scene

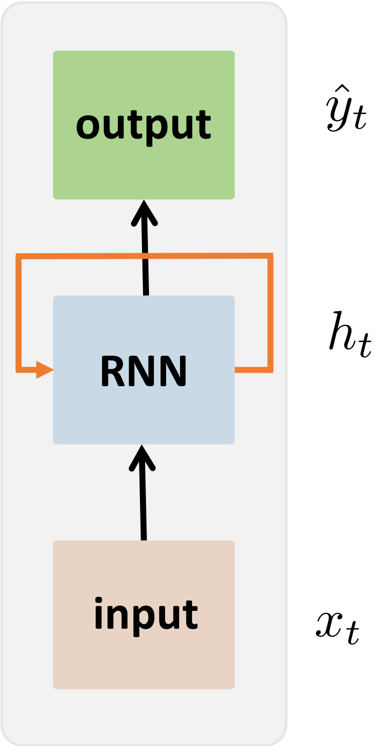
Key Idea: Output of one layer serves as input to the same layer.

Key Idea: Neurons with Recurrence

Recurrent Neural Networks

Recurrent Neural Networks – Vanilla Variant:

Recurrence: Past inputs are captured by the state h_t, that is the output of the recurrent cell.



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

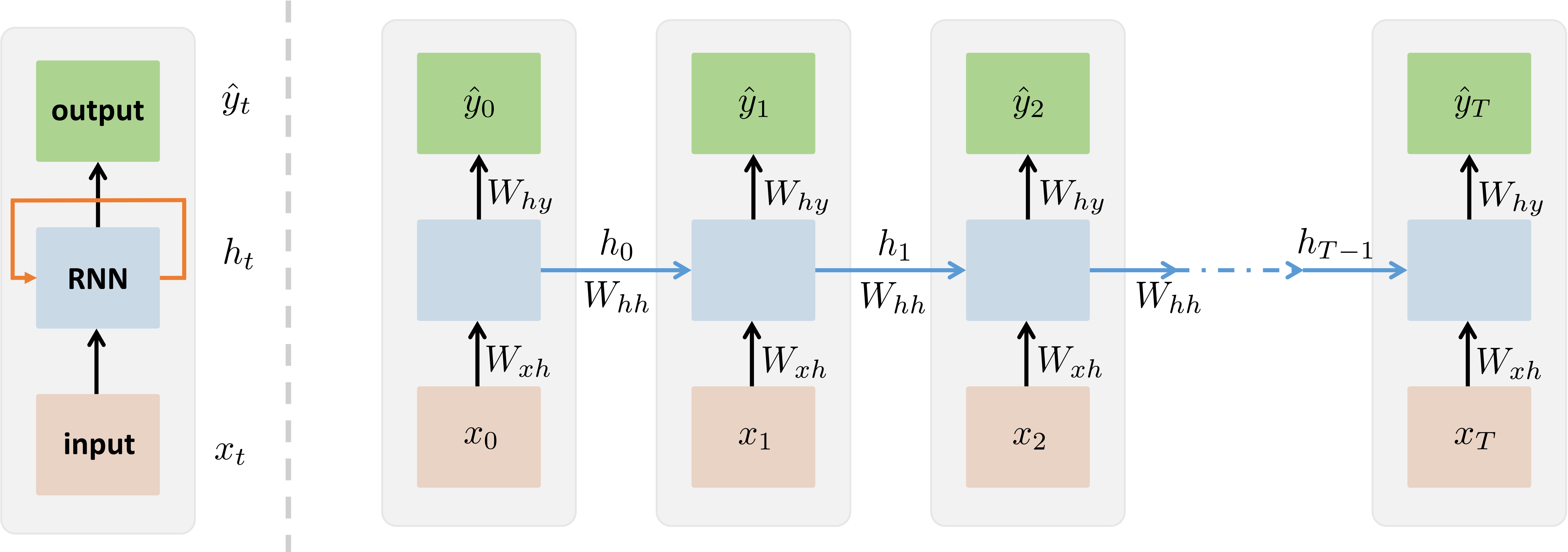
$$= \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

Recurrent Neural Networks

Recurrent Neural Networks – Vanilla Variant – Computational Graph:

Recurrence: Past inputs are captured by the state h_t, that is the output of the recurrent cell.



Same weights at each time step

Recurrent Neural Networks

RNNs for Sequence Modeling:

Example – Predict the Next Word – Many to Many:

Objective: Predict the next word in a sequence using embeddings and RNNs

Example Sentence:

$\boxed{\textit{I live in Canberra and I fluently speak English}}$

Key challenges:

Variable-length sequences

Long-term dependencies (“I” $\rightarrow$ “speak”)

Semantic relationships (“fluently” $\rightarrow$ “speak”)

Recurrent Neural Networks

RNNs for Sequence Modeling:

Example – Predict the Next Word

Step 1: Tokenization & Vocabulary:

Tokenized Sentence:

```
["I", "live", "in", "Canberra",
 "and", "I", "fluently",
 "speak", "English"]
```

Vocabulary:

Token	Index
I	0
live	1
in	2
Canberra	3
and	4
fluently	5
speak	6
English	7

Step 2: Input-Output Pairs:

Training sequences for RNN:

Input Sequence	Target Word
[0]	"live" (1)
[0,1]	"in" (2)
[0,1,2]	"Canberra" (3)
[0,1,2,3]	"and" (4)
[0,...,4]	"I" (0)
[0,...,0]	"fluently" (5)
[0,...,5]	"speak" (6)
[0,...,6]	"English" (7)

Note: Shortened notation [0,...,4] represents growing sequence

Recurrent Neural Networks

RNNs for Sequence Modeling:

Example – Predict the Next Word

Representation Comparison:

One-Hot Encoding
(vocab size = 8)

```
"I" --> [1,0,0,0,0,0,0,0]
"and" --> [0,0,0,0,1,0,0,0]
```

Problems:

High dimensionality

No semantic meaning

Embeddings
(dim = 2 for illustration)

```
"I" --> [0.1, -0.3]
"and" --> [0.5, 0.2]
"fluently" --> [-0.4, 1.1]
"speak" --> [1.2, 0.7]
```

Advantages:

Compact representation

Captures relationships

Recurrent Neural Networks

RNNs for Sequence Modeling:

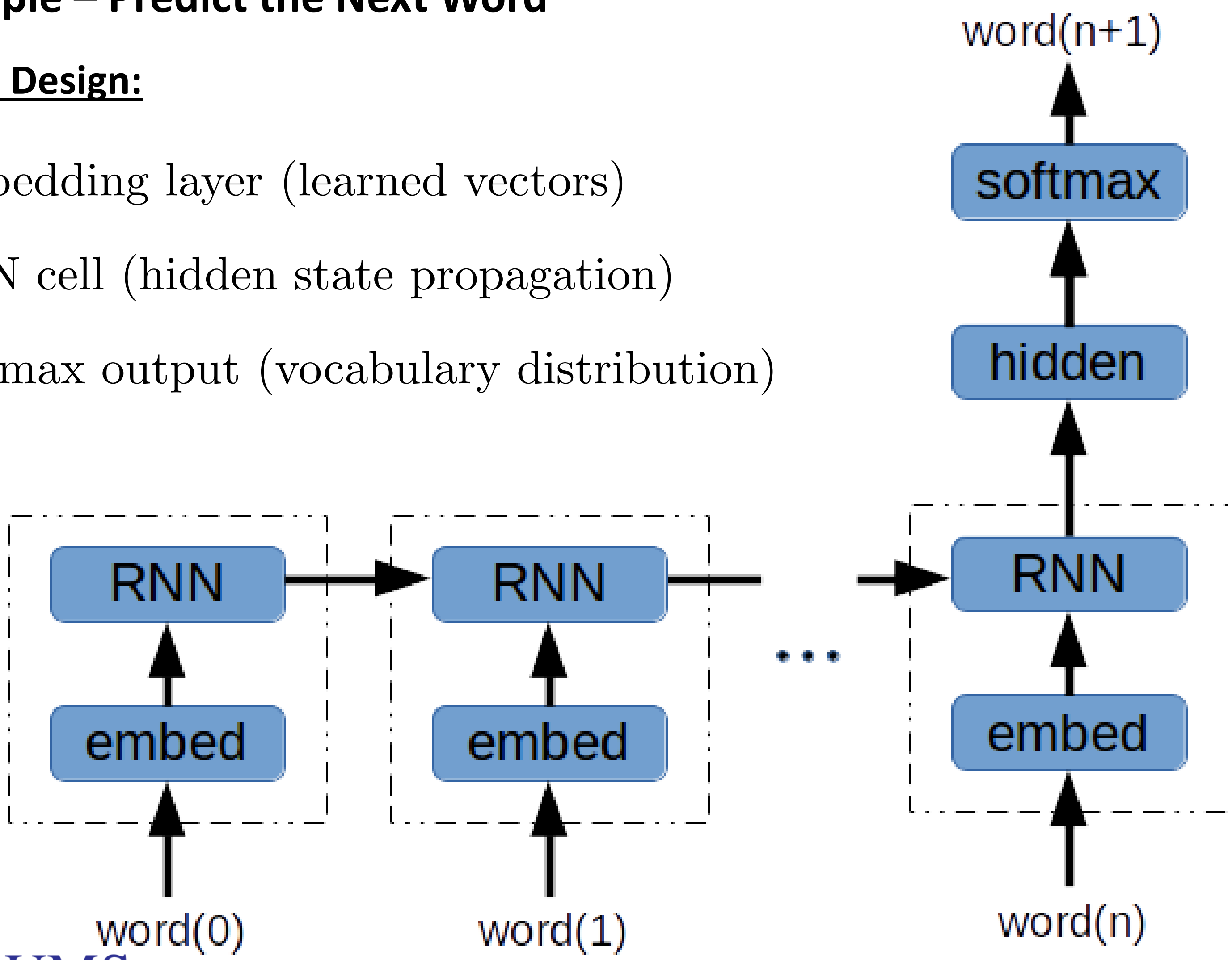
Example – Predict the Next Word

Model Design:

Embedding layer (learned vectors)

RNN cell (hidden state propagation)

Softmax output (vocabulary distribution)



Parameter Sharing: Same weights used at all time steps.

Teacher Forcing: During training, use ground truth inputs instead of previous predictions.

Recurrent Neural Networks

RNNs for Sequence Modeling:

To effectively model sequences, RNNs satisfy the following design criteria:

- **Handle Variable-Length Sequences:**

The model should accommodate input sequences of varying lengths, ensuring flexibility across different data scenarios.

- **Track Long-Term Dependencies**:

The model must capture relationships between elements that are far apart in the sequence, preserving context over extended intervals.

- **Maintain Information About Order**:

The sequential nature of the data should be preserved, as the order of elements often carries critical meaning.

- **Share Parameters Across the Sequence**:

Parameter sharing is essential to ensure the model generalizes well and remains efficient, especially for long sequences.

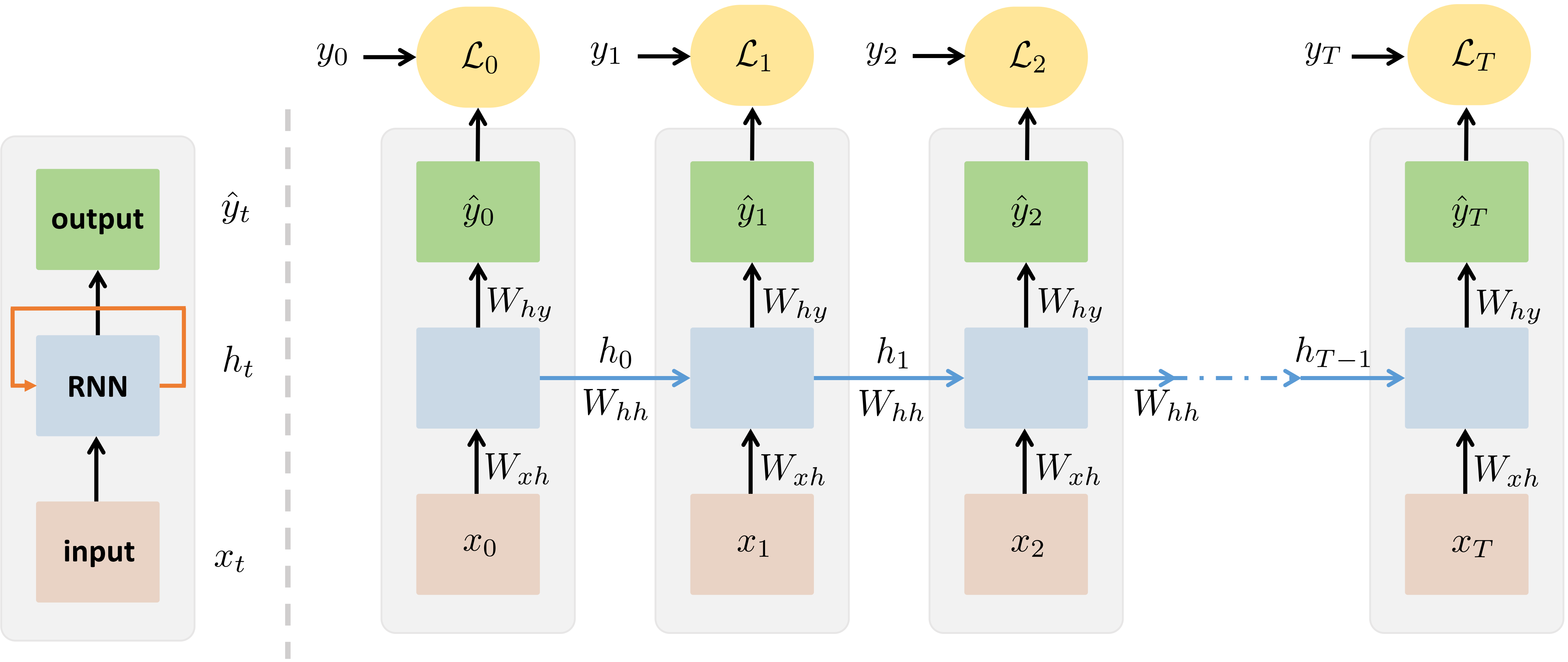
Recurrent Neural Networks

Recurrent Neural Networks – Vanilla Variant – Computational Graph:

Loss Computation:

Total Loss

$$\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}_T$$



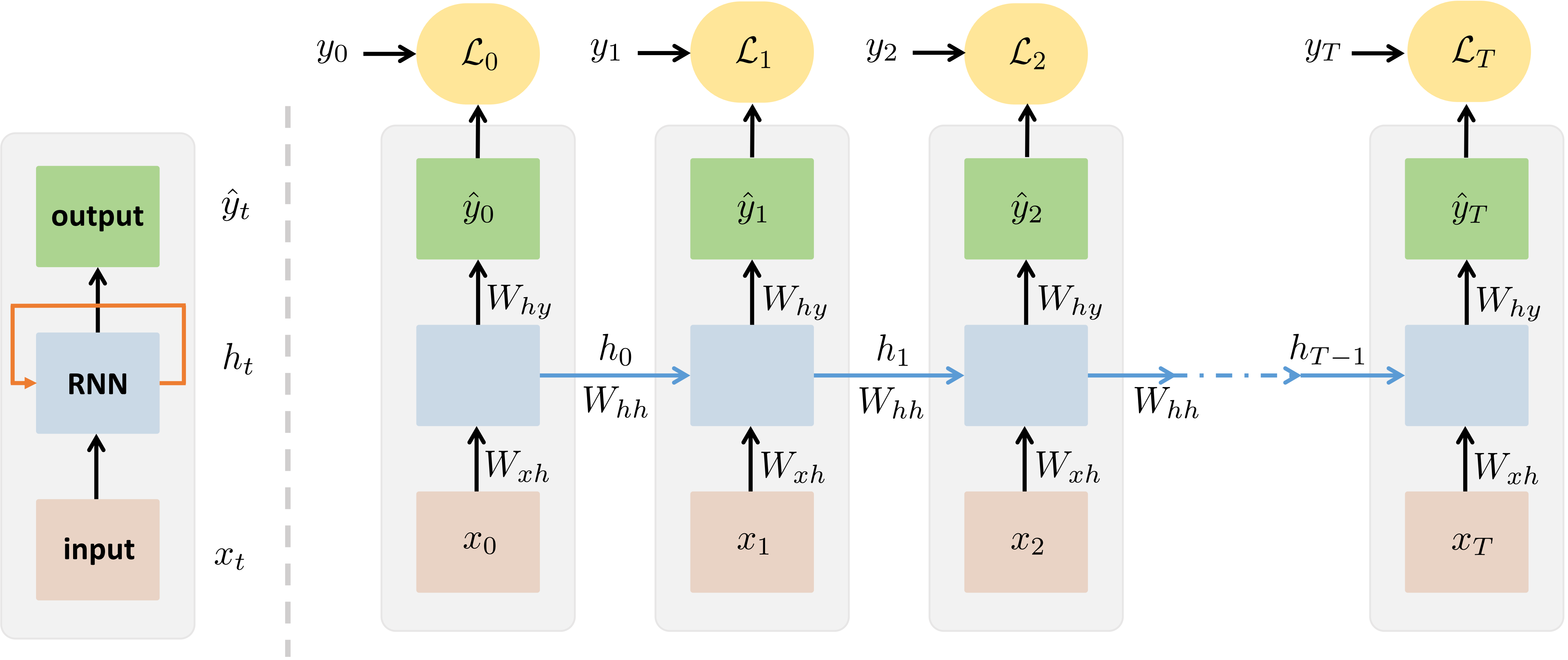
Recurrent Neural Networks

Recurrent Neural Networks – Vanilla Variant – Computational Graph:

How do we determine the weights/parameters? *Back Propagation (through time)*

Total Loss

$$\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}_t$$



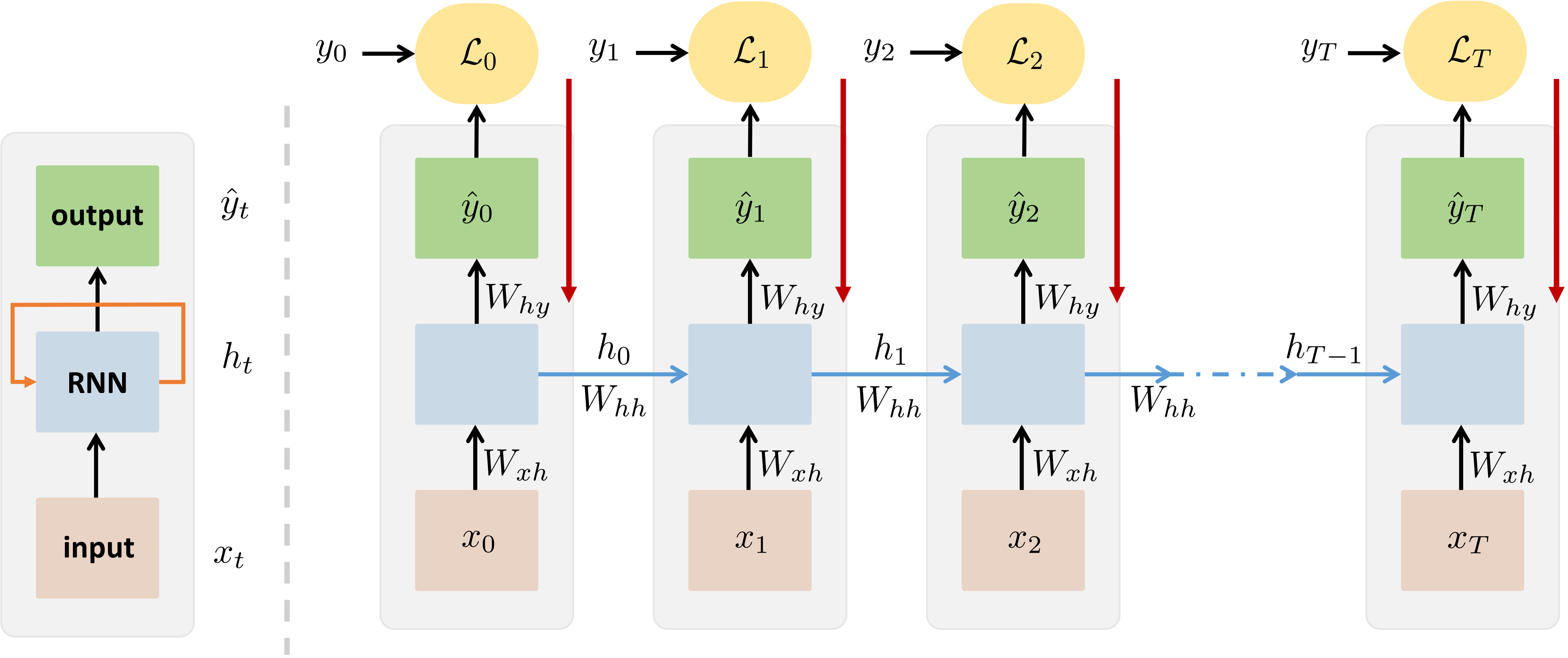
Recurrent Neural Networks

Recurrent Neural Networks – Vanilla Variant – Computational Graph:

How do we determine the weights/parameters? *Back Propagation (through time)*

Total Loss

$$\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}_t$$



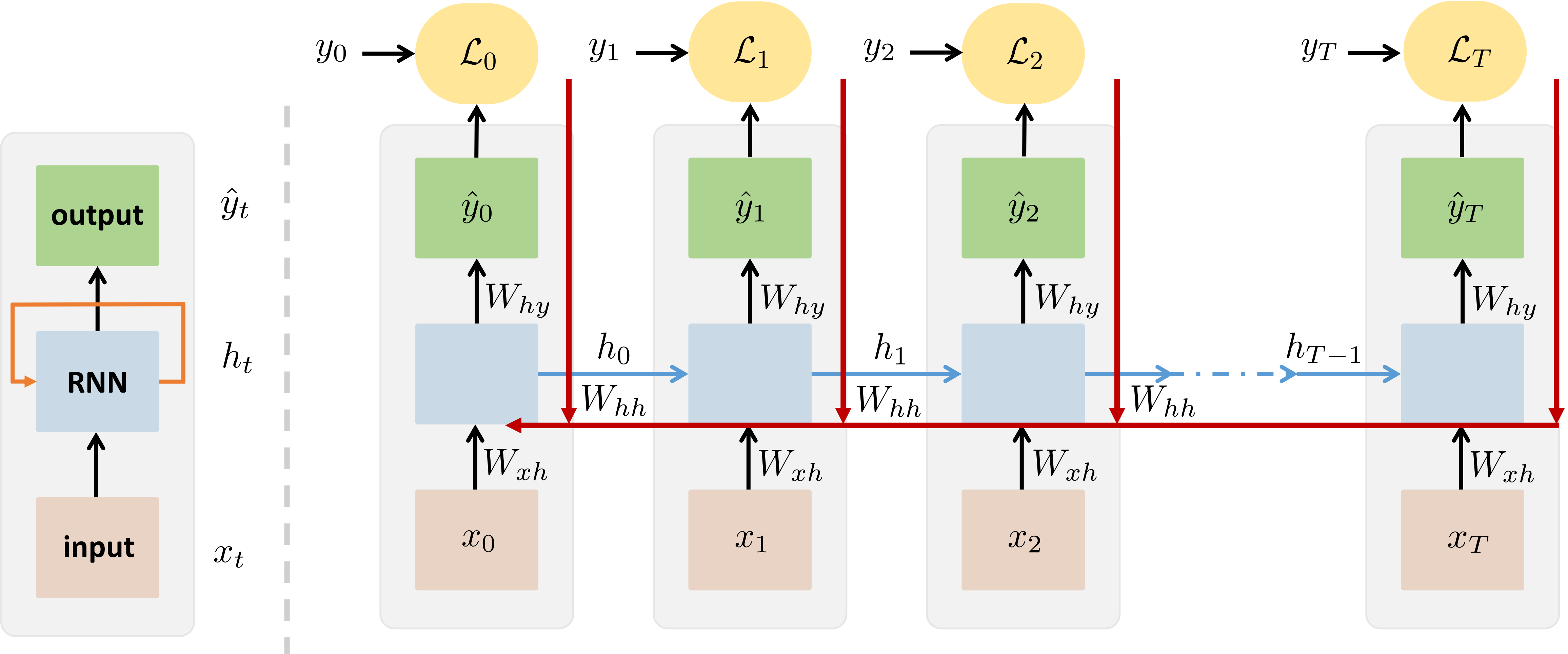
Recurrent Neural Networks

Recurrent Neural Networks – Vanilla Variant – Computational Graph:

How do we determine the weights/parameters? *Back Propagation (through time)*

Total Loss

$$\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}_t$$



LUMS
A Not-for-Profit University

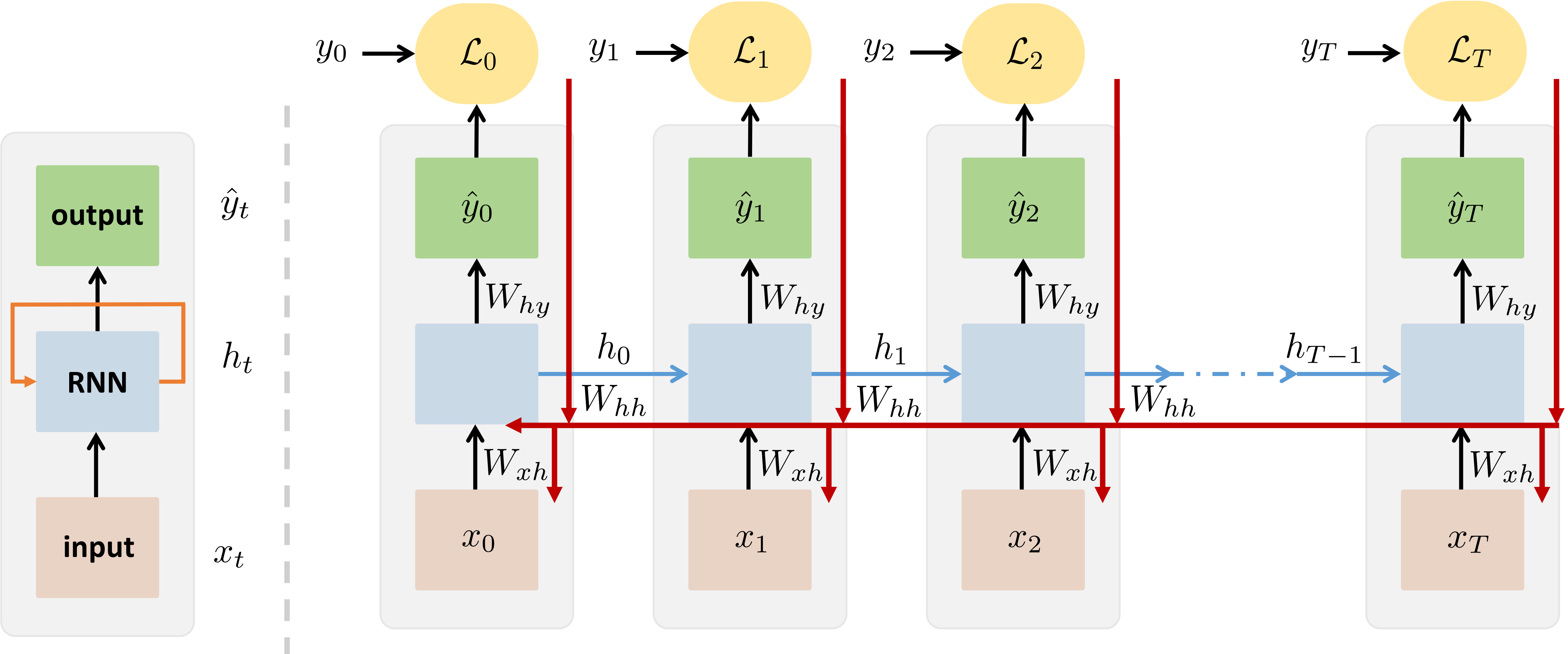
Recurrent Neural Networks

Recurrent Neural Networks – Vanilla Variant – Computational Graph:

How do we determine the weights/parameters? *Back Propagation (through time)*

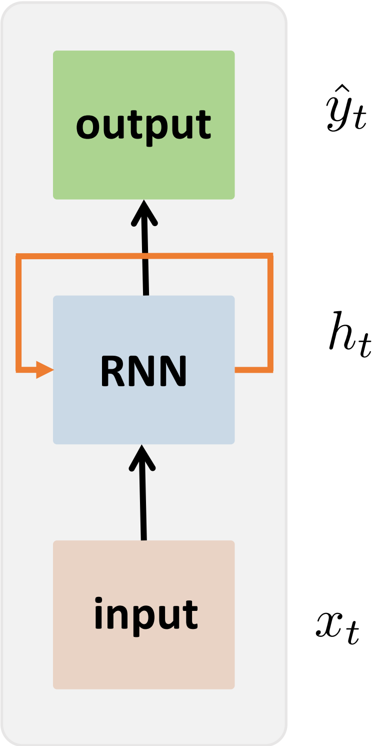
Total Loss

$$\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}_t$$



Recurrent Neural Networks

RNNs – Limitations and Extensions:



Basic RNN Limitations:

- Vanishing/Exploding gradients
- Struggles with long sequences

I lived my entire life in Pakistan and have recently moved to Canberra. I live in Canberra and I fluently speak ....

Recurrent Neural Networks

RNNs – Understanding Vanishing/Exploding Gradients

Consider a simple RNN with the following recurrence relation:

$$h_t = \phi(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

ϕ is a non-linear activation function (e.g., tanh).

Let the loss function at the final time step be $\mathcal{L}$. The gradient of the loss with respect to the hidden state h_t is:

$$\frac{\partial \mathcal{L}}{\partial h_t} = \frac{\partial \mathcal{L}}{\partial h_T} \cdot \frac{\partial h_T}{\partial h_t}$$

$$\frac{\partial \mathcal{L}}{\partial h_t} = \frac{\partial \mathcal{L}}{\partial h_T} \cdot \prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}}$$

(Expanding using the chain rule)

Since

$$\frac{\partial h_k}{\partial h_{k-1}} = \phi'(a_k) \cdot W_{hh}, \quad \text{where } a_k = W_{hh}h_{k-1} + W_{xh}x_k + b_h,$$

we get

$$\frac{\partial \mathcal{L}}{\partial h_t} = \frac{\partial \mathcal{L}}{\partial h_T} \cdot \prod_{k=t+1}^{T} (\phi'(a_k) \cdot W_{hh})$$

Recurrent Neural Networks

RNNs – Understanding Vanishing/Exploding Gradients

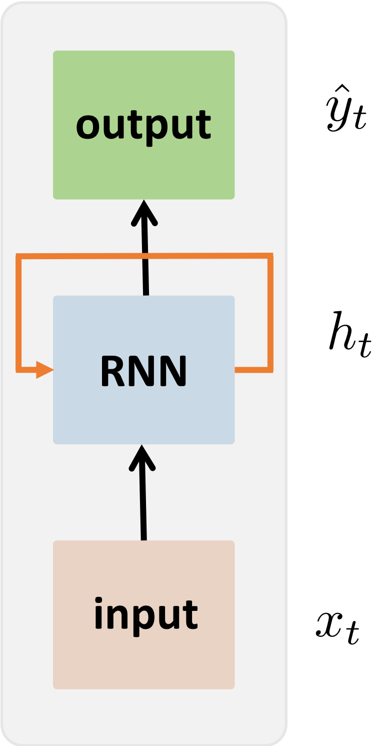
If the spectral norm of W_{hh} is less than 1 and ϕ' produces small values (as in the case of tanh or sigmoid), then:

$$\left\| \prod_{k=t+1}^{T} \left(\phi'(a_k) \cdot W_{hh}\right) \right\| \to 0 \quad \text{as } T - t \to \infty$$

This leads to the vanishing gradient problem, where early layers (smaller t) receive negligible gradient signals during backpropagation.

Recurrent Neural Networks

RNNs – Limitations and Extensions:



Basic RNN Limitations:

- Vanishing/Exploding gradients
- Struggles with long sequences

I lived my entire life in Pakistan and have recently moved to Canberra. I live in Canberra and I fluently speak ....

Improvements:

- LSTMs (next topic)
- Attention mechanisms

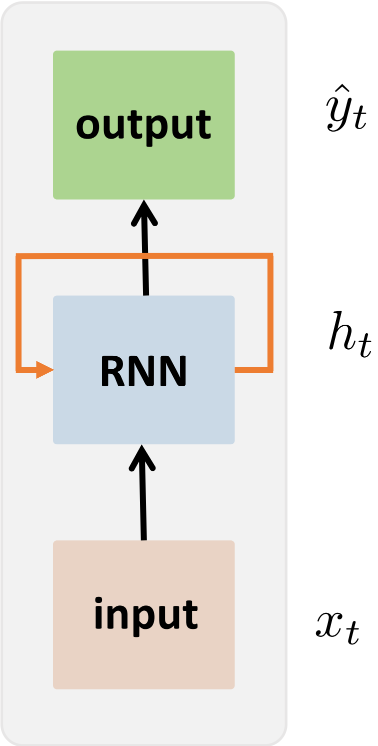
Real-World Scaling:

- Use pre-trained embeddings (GloVe, Word2Vec)

Outline

- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory (LSTM) Networks

Limitations of Vanilla RNNs



Problem 1: Vanishing/Exploding Gradients

- During Backpropagation Through Time (BPTT), gradients are multiplied by weight matrices repeatedly, causing them to shrink (vanish) or grow (explode). This makes learning long-term dependencies difficult.
- Example: In a sequence of length 100, gradients involve $\mathbf{W}^{100}$, leading to instability.

Problem 2: Short-Term Memory

- Hidden states in RNNs are overwritten at each time step, making it challenging to retain information over long sequences.

Long Short-Term Memory (LSTM) Networks

Intuition Behind Selective Memory Update

LSTM networks are designed to remember important information and forget what's not useful. They do this using three key mechanisms:

1. **Forget Gate:**

Decides what part of the past information should be *forgotten*. Think of it like cleaning up memory —"Is this old info still relevant?"

2. **Input Gate (Selective Write):**

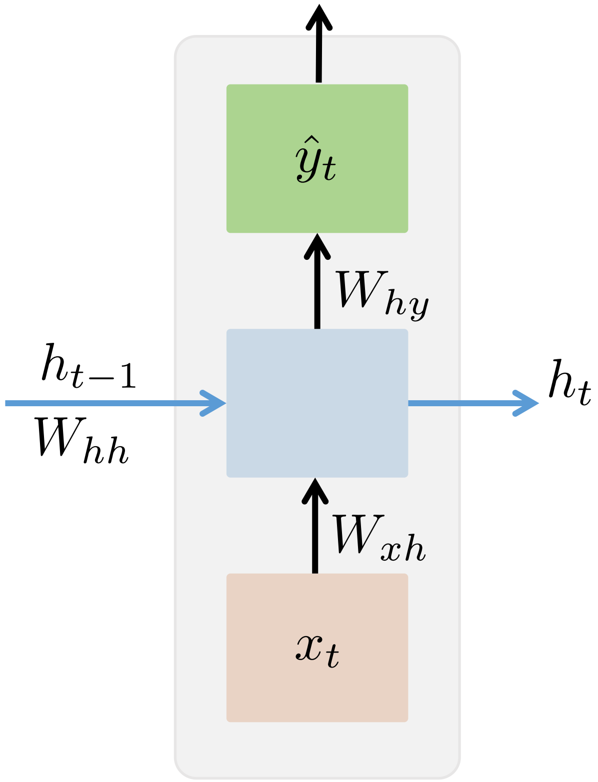
Determines what new information should be added to memory. This is like saying, "I just saw something new — should I store it?"

3. **Output Gate (Selective Read):**

Chooses what part of the memory to use for the next step. It's like asking, "What do I need to remember right now to make a decision?"

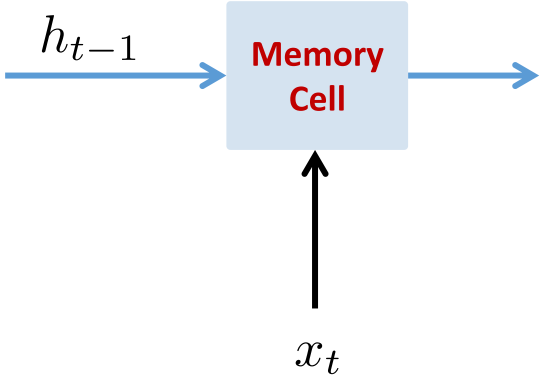
Together, these gates help LSTMs handle long-term dependencies more effectively.

Long Short-Term Memory (LSTM) Networks



RNN

RNN



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

Current hidden state depends on

- *previous hidden state*
- *current input*

Long Short-Term Memory (LSTM)

Equip memory cell with

- *'internal' state*, and
- *multiplicative gates* that determine
  - (i) How should a given input impact the internal state (the *input gate*)?
  - (ii) How should the internal state be flushed to 0 (the *forget gate*)?
  - (iii) How should the internal state of a given neuron be allowed to impact the cell's output (the *output gate*)?

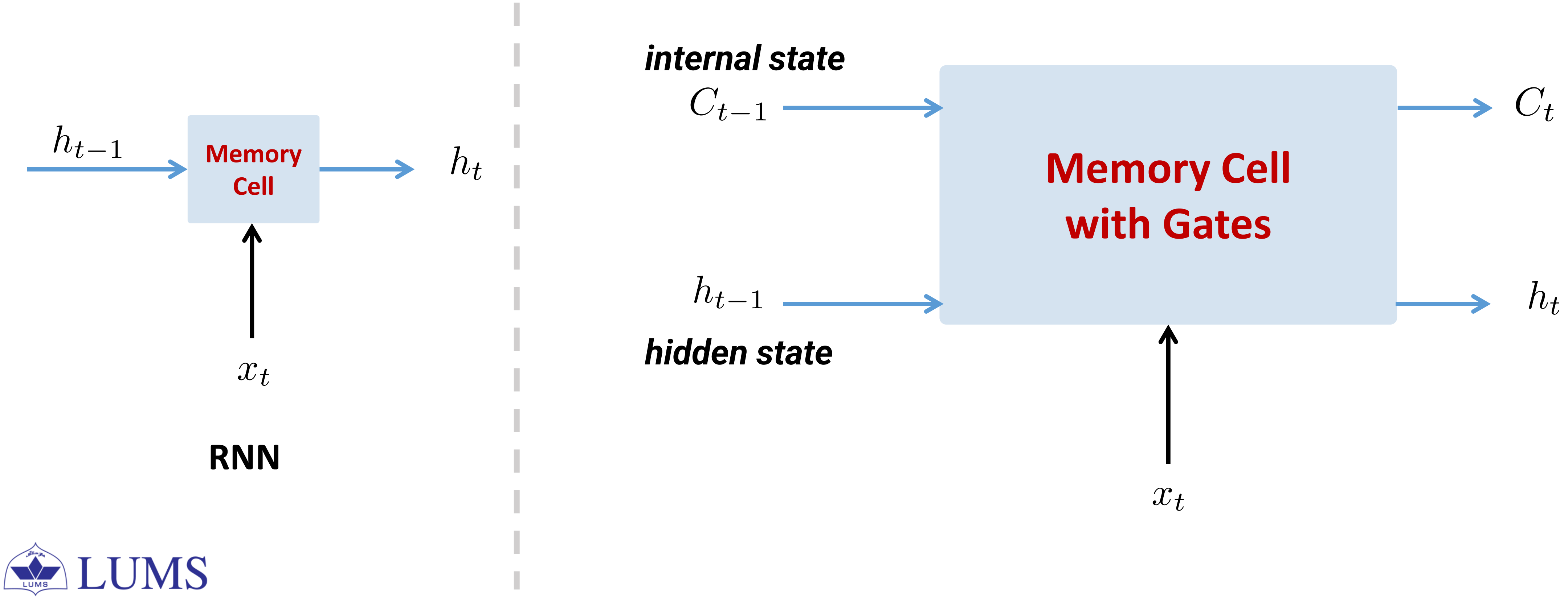
Long Short-Term Memory (LSTM) Networks

Key Idea:

Introduce a cell state (C_t) as a

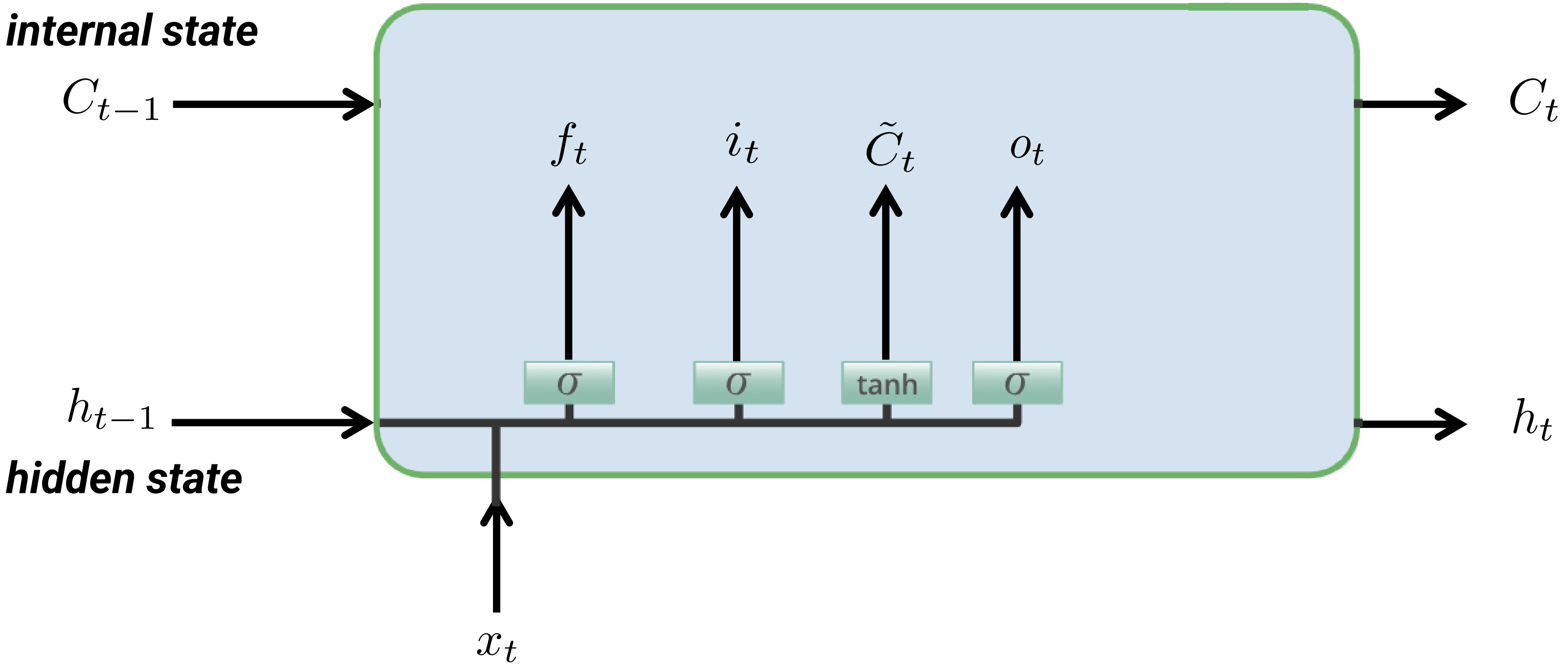
"memory highway"

regulated by gates controlling information propagation over time.



Long Short-Term Memory (LSTM) Networks

Formulation:



Forget gate $f_t = \sigma(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f)$

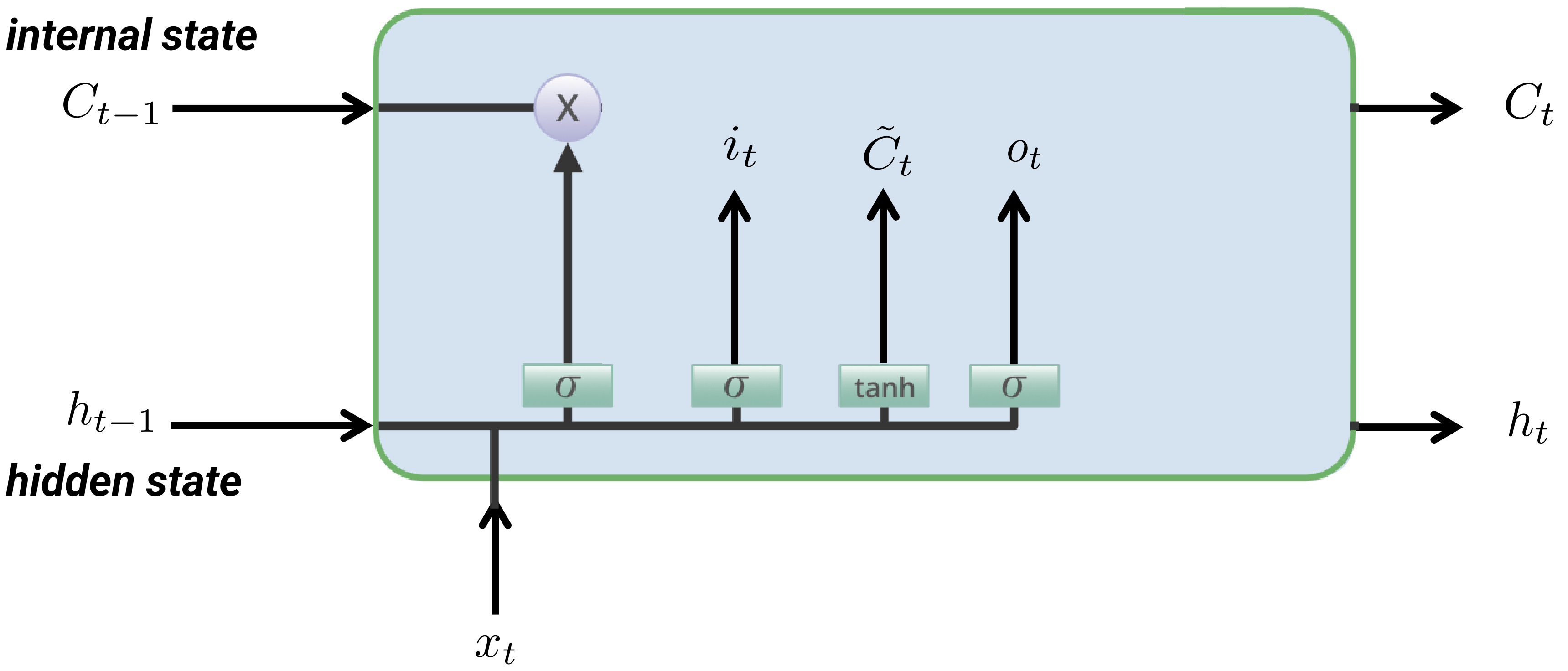
Input gate $i_t = \sigma(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i)$

Output gate $o_t = \sigma(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o)$

Input node $\tilde{C}_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$

Long Short-Term Memory (LSTM) Networks

Formulation – Forget Gate Action:



Forget gate $f_t = \sigma(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f)$

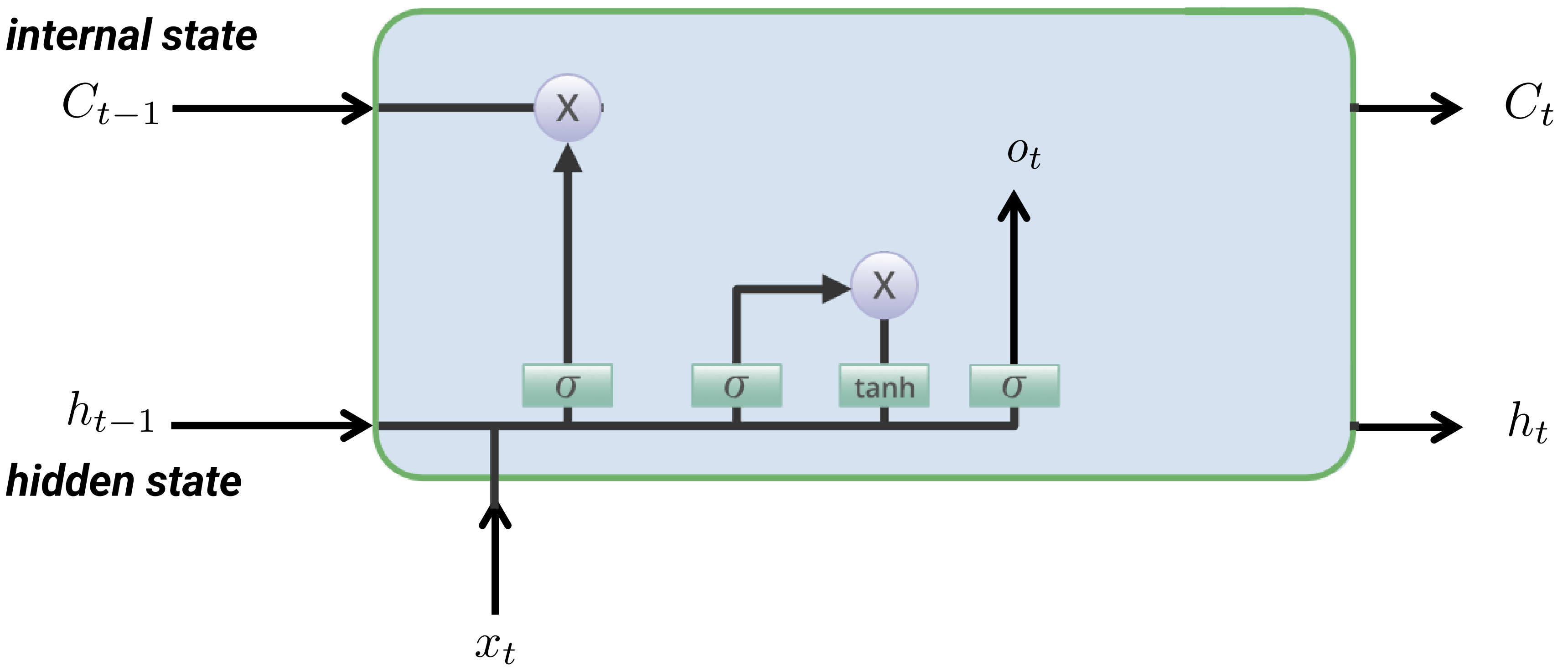
Input gate $i_t = \sigma(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i)$

$o_t = \sigma(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o)$ **Output gate**

$\tilde{C}_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$ **Input node**

Long Short-Term Memory (LSTM) Networks

Formulation – Input Gate Action:



Forget gate $f_t = \sigma(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f)$

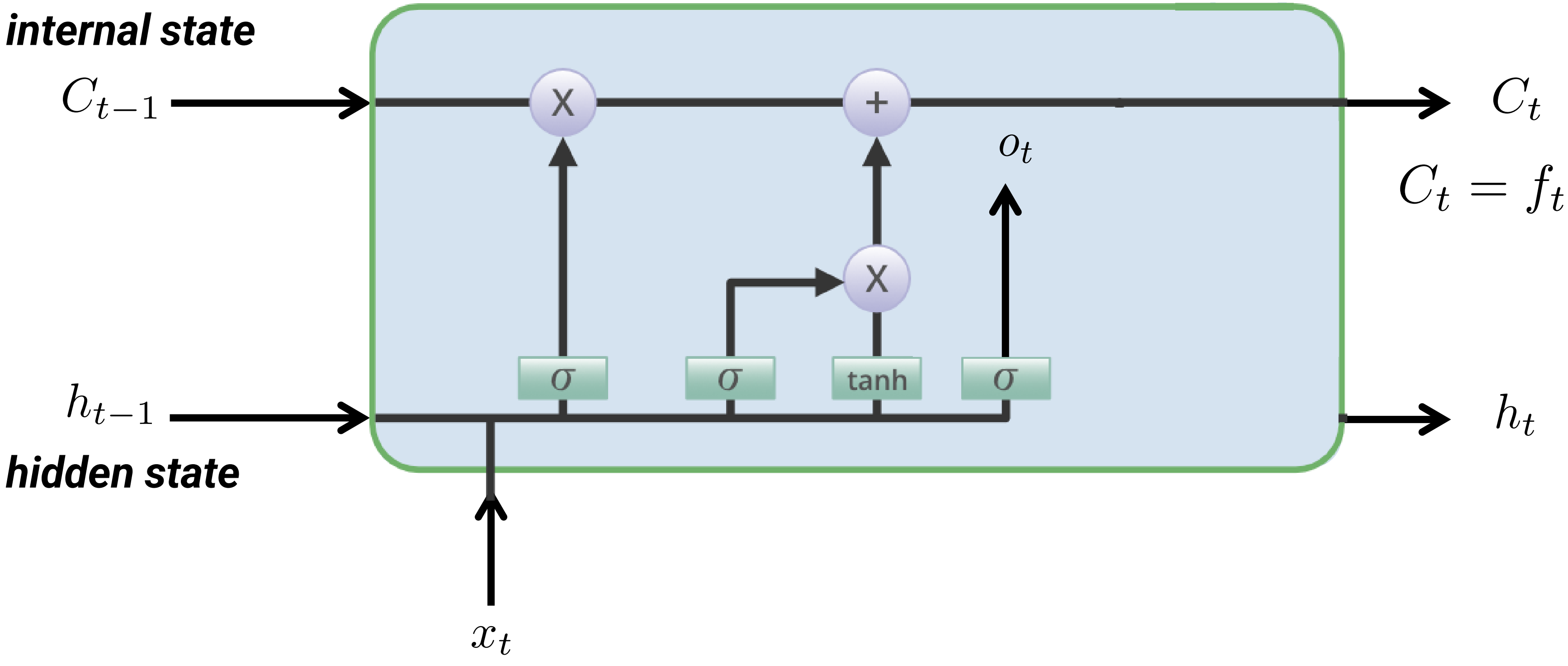
Input gate $i_t = \sigma(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i)$

$o_t = \sigma(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o)$ **Output gate**

$\tilde{C}_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$ **Input node**

Long Short-Term Memory (LSTM) Networks

Formulation – Determining Next (Internal) State:



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

Forget gate $f_t = \sigma(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f)$

Input gate $i_t = \sigma(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i)$

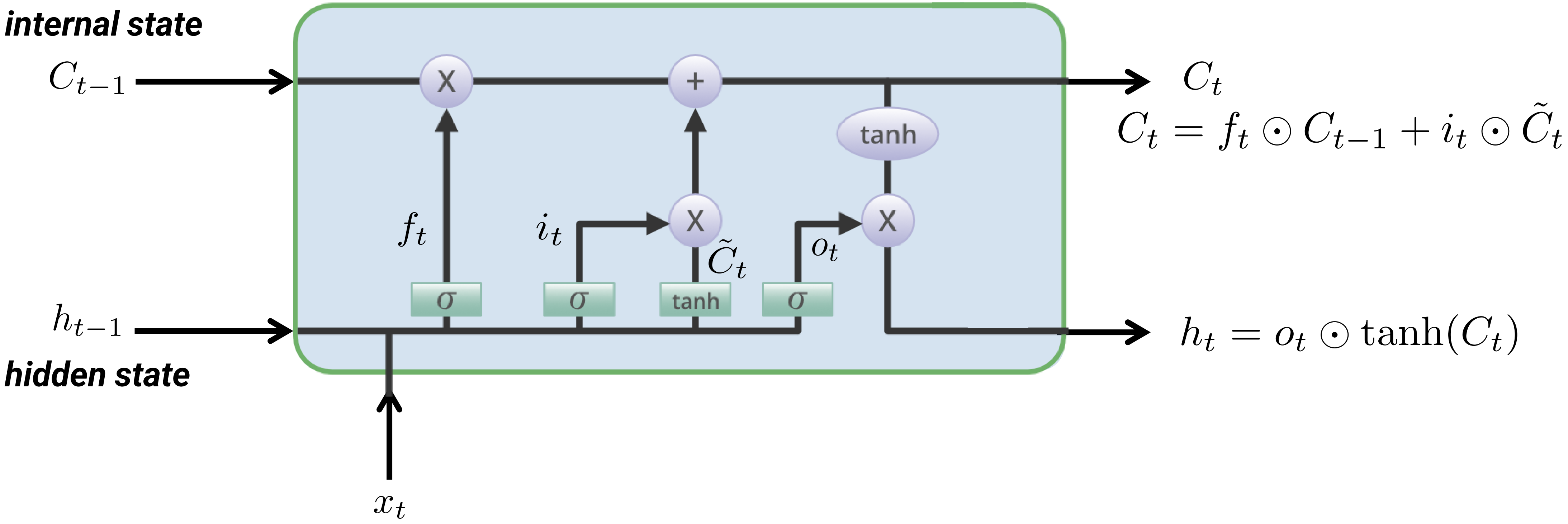
$o_t = \sigma(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o)$ **Output gate**

$\tilde{C}_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$ **Input node**

$\odot$ – element wise multiplication

Long Short-Term Memory (LSTM) Networks

Formulation – Output Gate Action and Next Hidden State:



Forget gate $f_t = \sigma(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f)$

$o_t = \sigma(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o)$ **Output gate**

Input gate $i_t = \sigma(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i)$

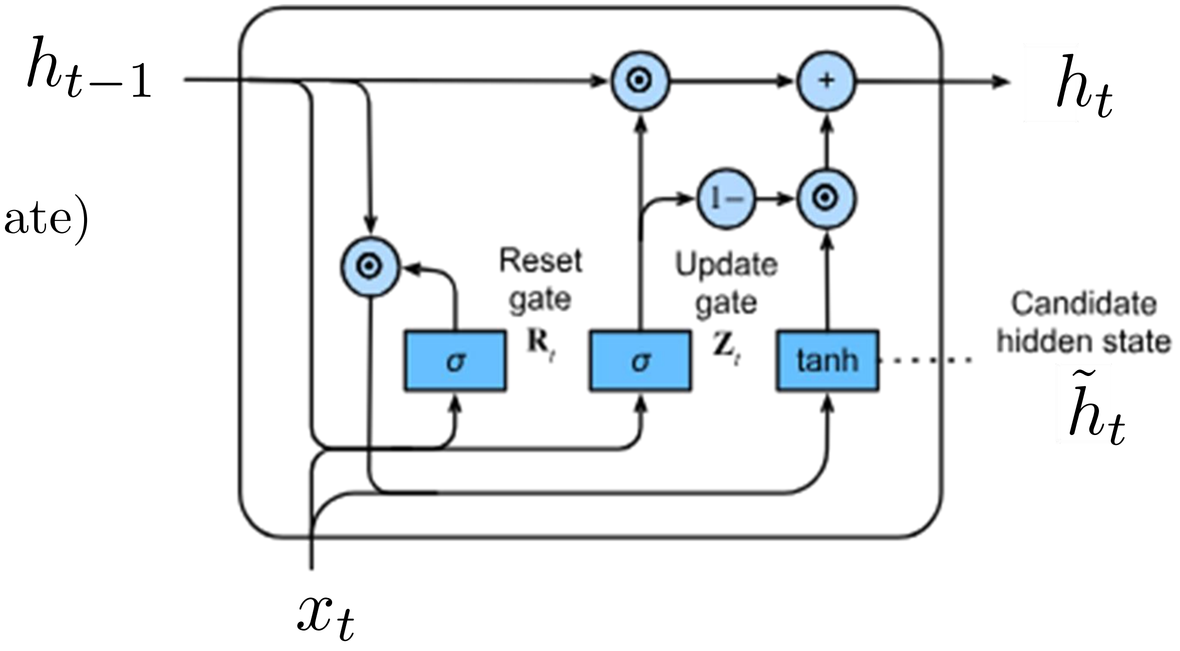
$\tilde{C}_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$ **Input node**

Long Short-Term Memory (LSTM) Networks

Variations and Extensions – Gated Recurring Units:

GRUs simplify LSTMs by combining the forget and input gates into a single update gate. The key equations are:

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \quad \text{(Reset Gate)}$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \quad \text{(Candidate State)}$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \quad \text{(Update Gate)}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad \text{(Final Hidden State)}$$



Key Idea: GRUs control how much past information to keep (z_t) and how much to reset (r_t).