Recurrent Neural Networks (RNNs)

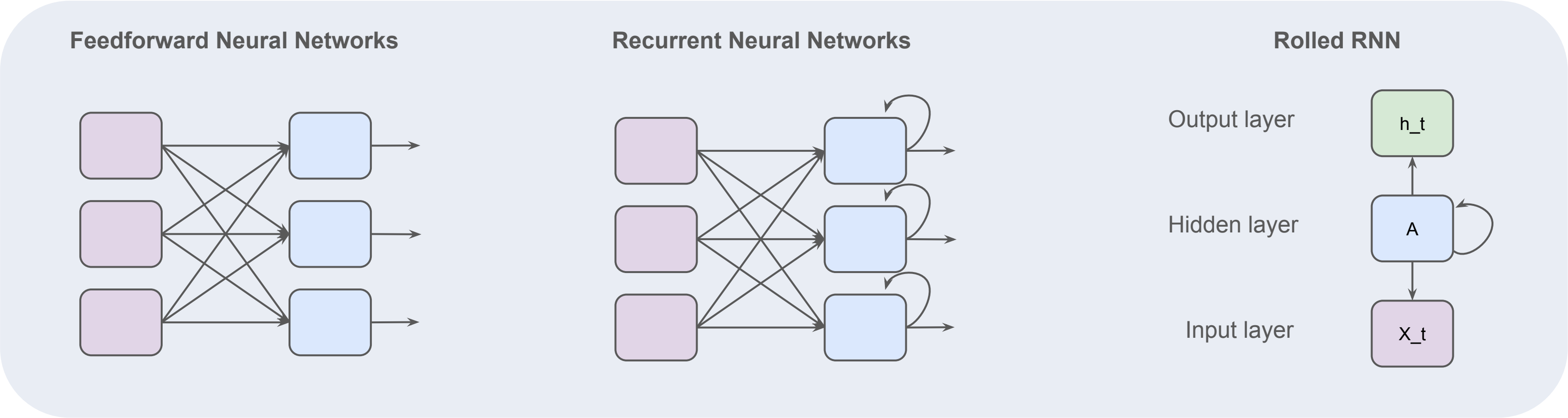
Lorena Gallego Viñarás

(logalleg@pa.uc3m.es)

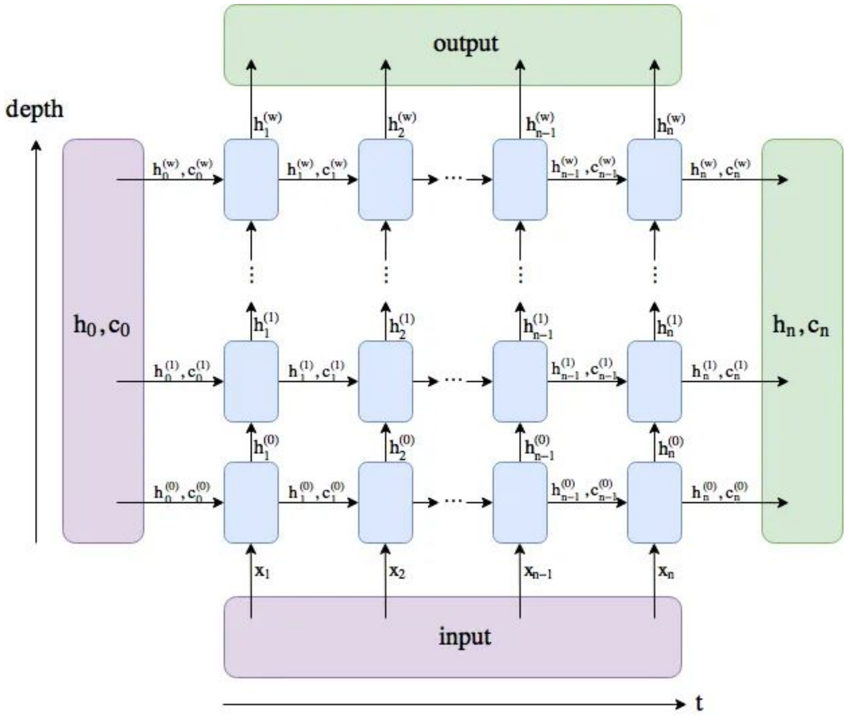
uc3m | Universidad Carlos III de Madrid

What are RNNs?

- Designed to **process sequential data** by maintaining a **memory of previous inputs**.
- RNNs **capture temporal dependencies** — the output at time t depends on previous steps.
- They use a **hidden state** passed across time steps, enabling the network to “remember” past context.
- RNNs **share weights** across all time steps.
- Trained via **Backpropagation Through Time (BPTT)**



Unrolled RNNs



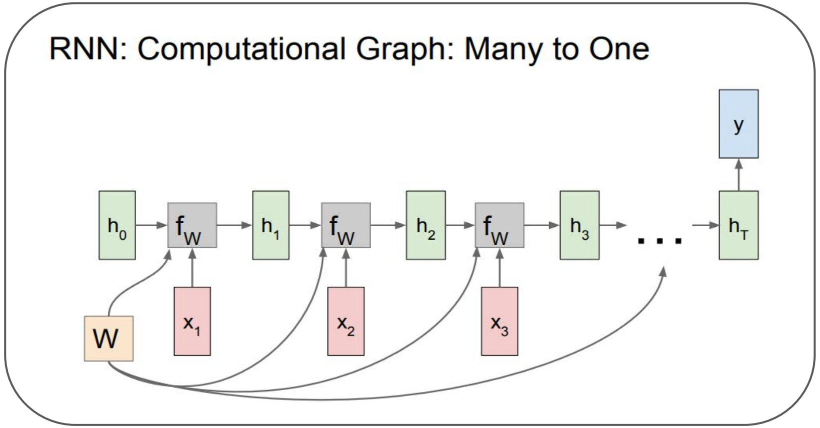
- **Horizontal Axis (Time – t):**
  Each column corresponds to a single time step (x_1, x_2, … , x_n)

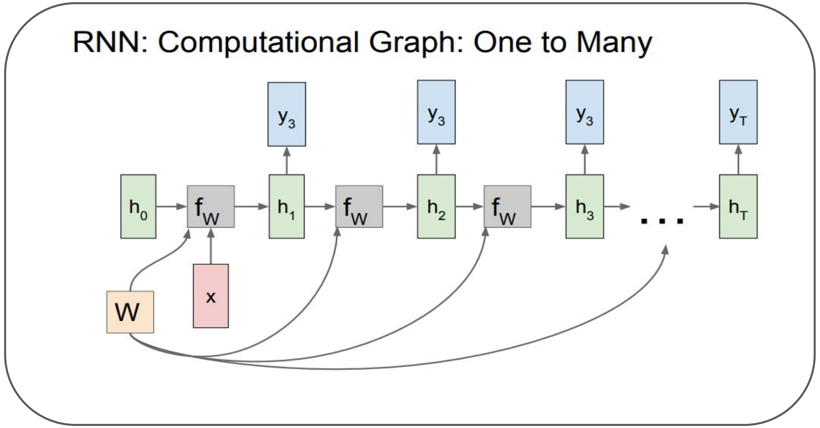
- **Vertical Axis (Depth – Layers):**
  Each row represents a different RNN layer. Inputs are processed layer by layer, from bottom to top.

- **States:**
  - $h_t^{(l)}$: Hidden state at time t, layer l
  - $c_t^{(l)}$: Cell state at time t, layer l (only in LSTM)

- RNN has two outputs - out and hidden.

Types of RNNs

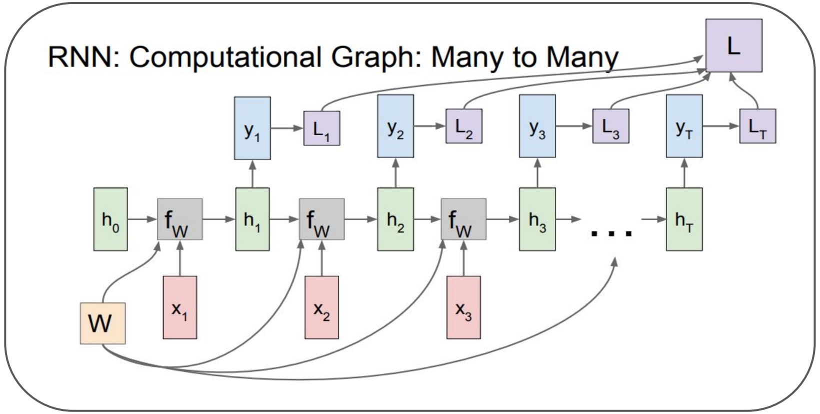




$$h_t = f_W(h_{t-1}, x_t)$$

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

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



Applications of RNNs

- Language Modeling & Translation
- Sentiment Analysis
- Speech Recognition & Synthesis
- Time-Series Forecasting
- Music Generation
- Video & Gesture Recognition
- Robotics & Autonomous Systems
- Dialogue Systems & Storytelling

Problems of traditional RNNs

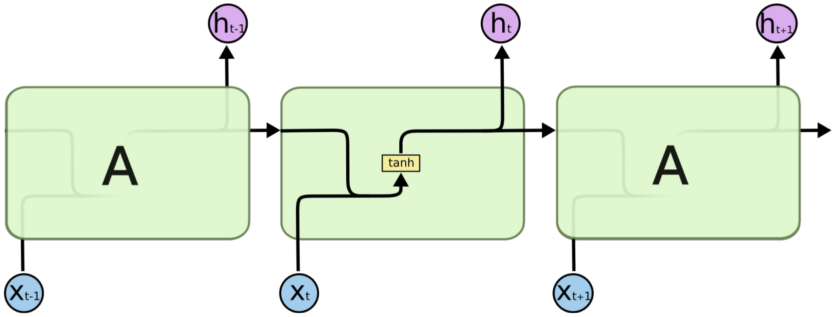
- ✖ Vanishing gradients
- ✖ Exploding gradients
- ✖ Long-term memory fade
- ✖ Limited global context
- ✖ Low parallelism

Solutions?

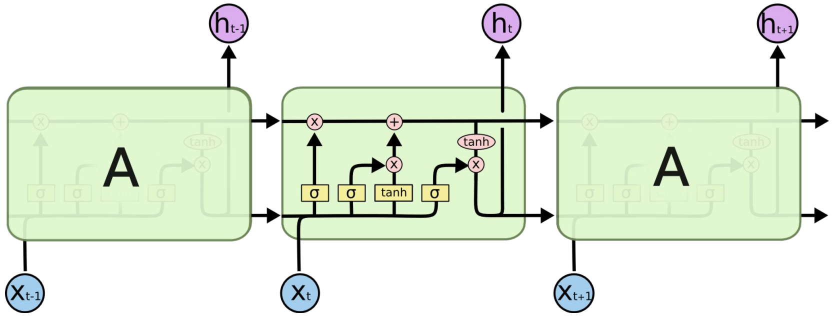
- **Reduce** network **depth** to lower complexity.
- Techniques such as **gradient clipping**, **learning rate scheduling**, and careful **regularization**
- Use advanced architectures like **LSTM** or **GRU.**

LSTM (Long Short-Term Memory)

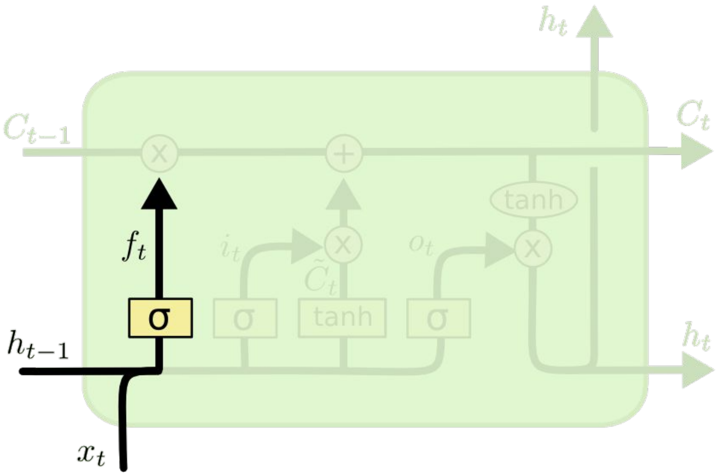
Standard RNN



LSTM



LSTM (Long Short-Term Memory)



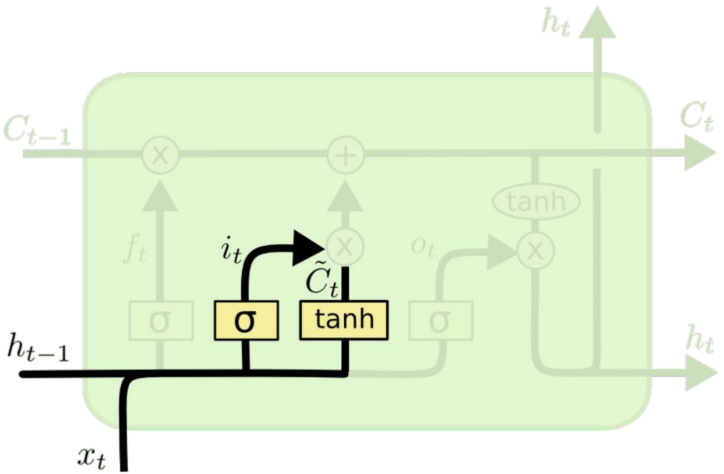
Forget gate

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

Decides **what** information to **discard**.

LSTM (Long Short-Term Memory)

uc3m



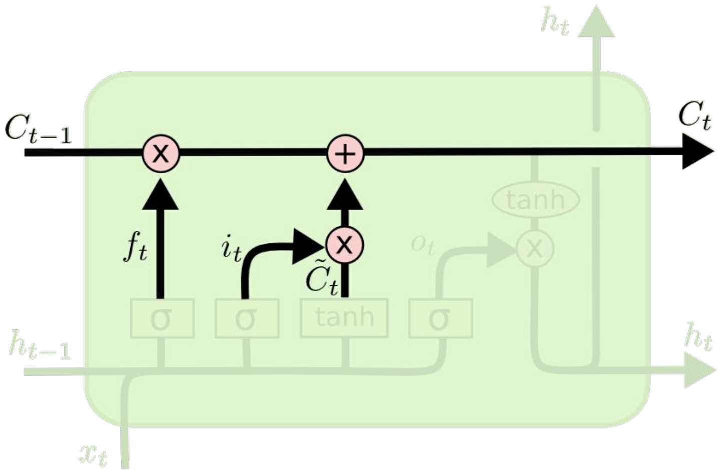
Input gate

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

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

Determines **what** information to **store**.

LSTM (Long Short-Term Memory)

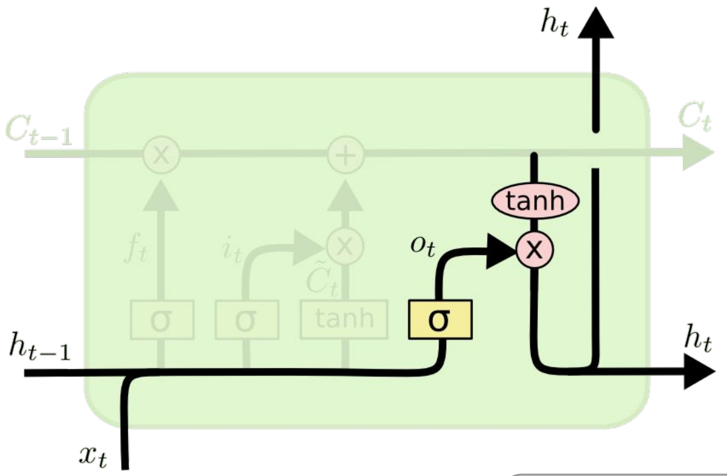


Input gate

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

Determines **what** information to **store**.

LSTM (Long Short-Term Memory)



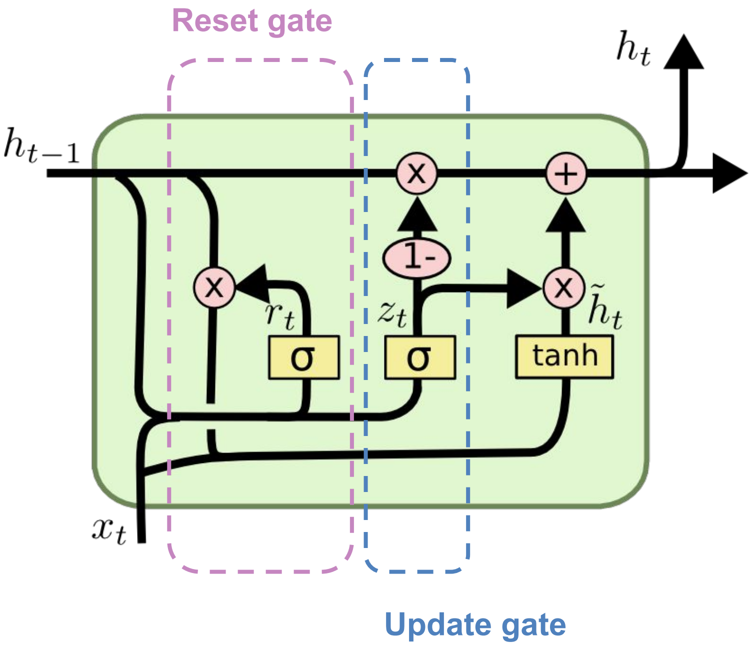
Output gate

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

$$h_t = o_t * \tanh\left(C_t\right)$$

Controls **what** information **is sent** to the next layer

GRU (Gated Recurrent Unit)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Reset gate: decides **how much** of the information from the past **should be kept** for future steps
- Update gate: determines **how much** past information **should be forgotten.**

Comparative table

Aspect	RNN	LSTM	GRU
Architecture	- Simple recurrent structure - Vanishing gradient issue	- Memory cells - Input, forget, output gates	- Fewer gates - Simpler than LSTM
Learning Capability	- Short-term dependencies only	- Good for long-term dependencies	- Handles both short & long dependencies
Training Complexity	- Easy to implement - Unstable gradients	- Complex - Slow to train	- Simpler than LSTM - Faster training
Typical Applications	- Basic sequence modeling - Stock prediction	- Language modeling - Translation	- Speech & video processing
Advantages	- Easy to understand	- Effective for long sequences	- Lightweight - Comparable to LSTM
Challenges	- Vanishing gradient - Poor long-term memory	- Slow training - Many parameters	- Sometimes slightly less accurate

PyTorch implementation

Import useful libraries...

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import torch.nn.functional as F
import numpy as np
```

... and now, let's get hands on!

Bibliography

https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

https://www.kaggle.com/code/kanncaa1/recurrent-neural-network-with-pytorch/notebook

https://medium.com/data-science/pytorch-basics-how-to-train-your-neural-net-intro-to-rnn-cb6ebc594677

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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

https://github.com/udacity/deep-learning-v2-pytorch/blob/master/recurrent-neural-networks/time-series/Simple_RNN.ipynb

https://medium.com/aimonks/recurrent-neural-network-working-applications-challenges-f445f5f297c9