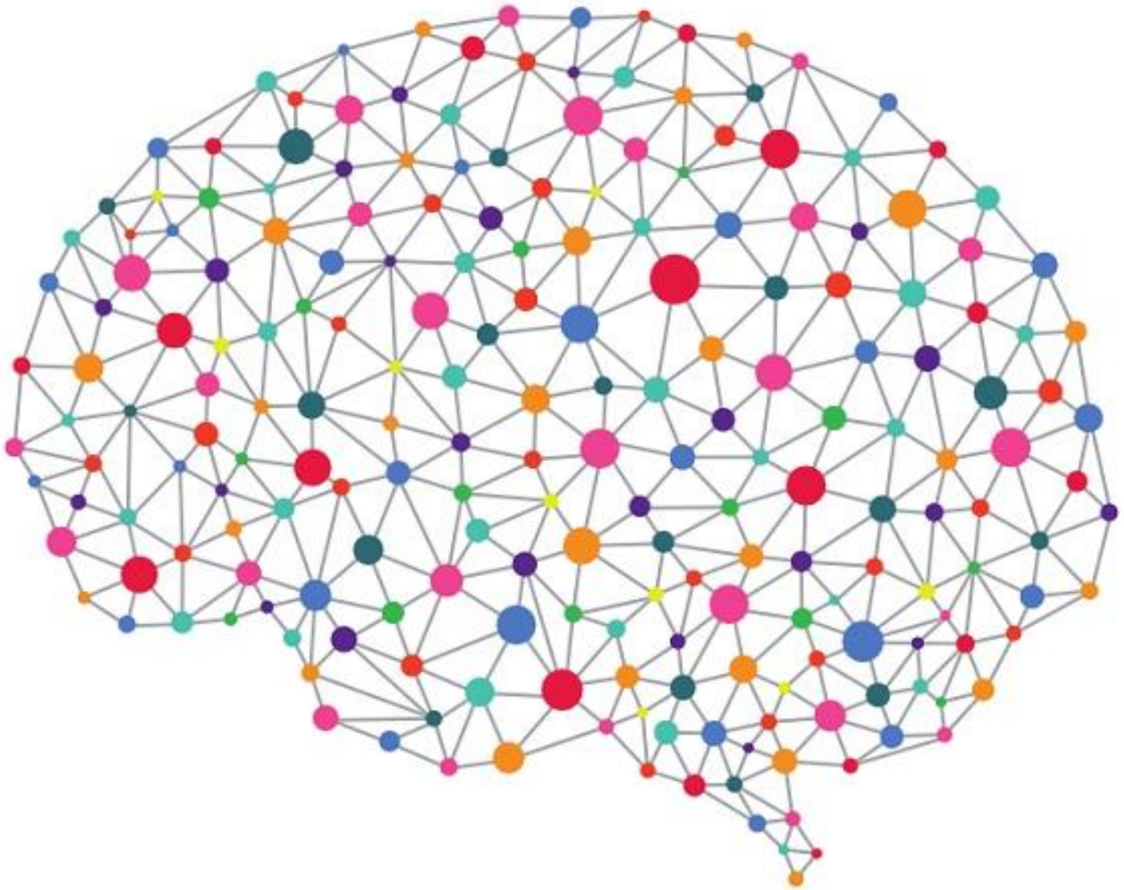
Modifier 5.0

# Deep Learning For Beginners!



IEEE HELWAN Student Branch

# Contents :

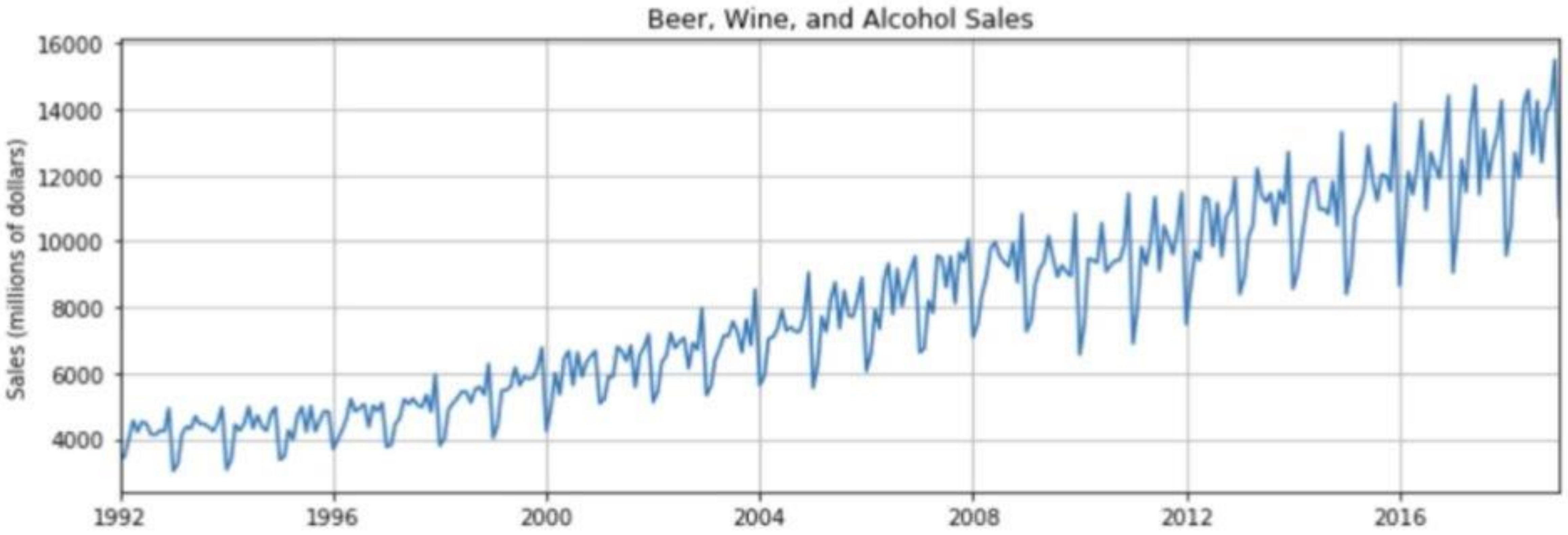
- **RNN Theory**
- **LSTMS**

- Just as CNNs were more effective for use with 2D image data, RNNs are more effective for sequence data (e.g. time-stamped sales data, sequence of text, heart beat data, etc...)

# Sequential Data :

- Predict the next word in a sentence spoken or written.
- Stock Price Predictions.
- Predict your sales in the next month or year.
- *Predict the next frame in a video.*

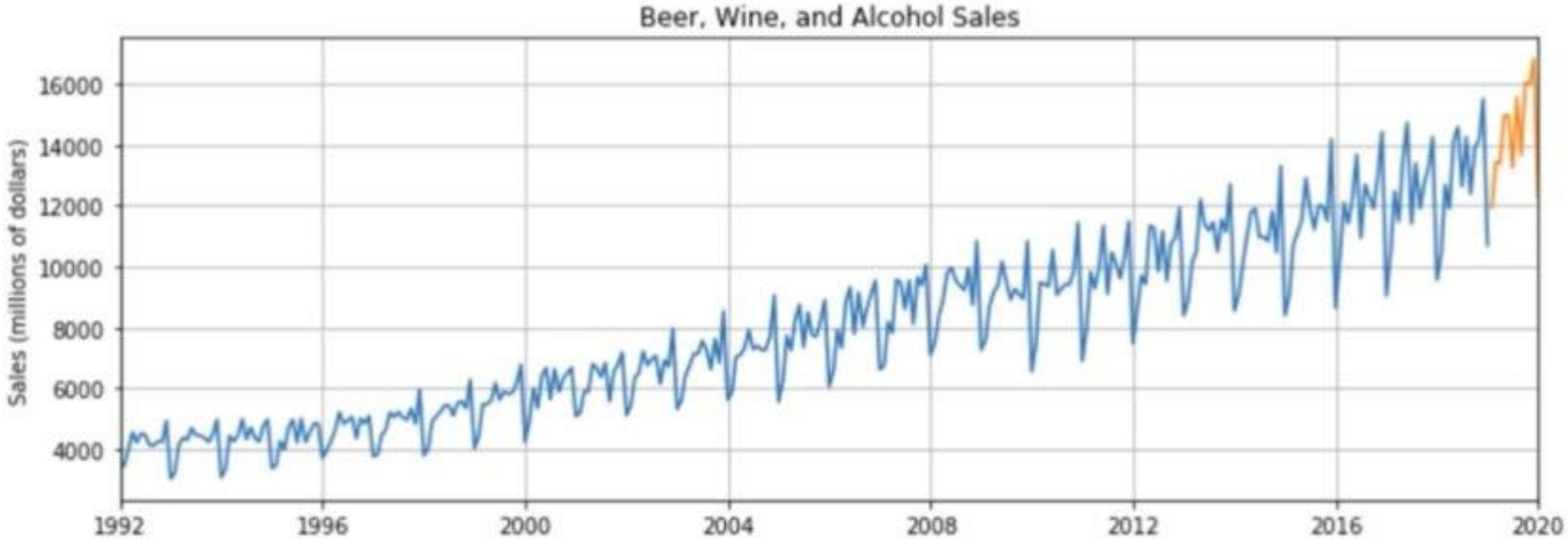
# Time Series Data



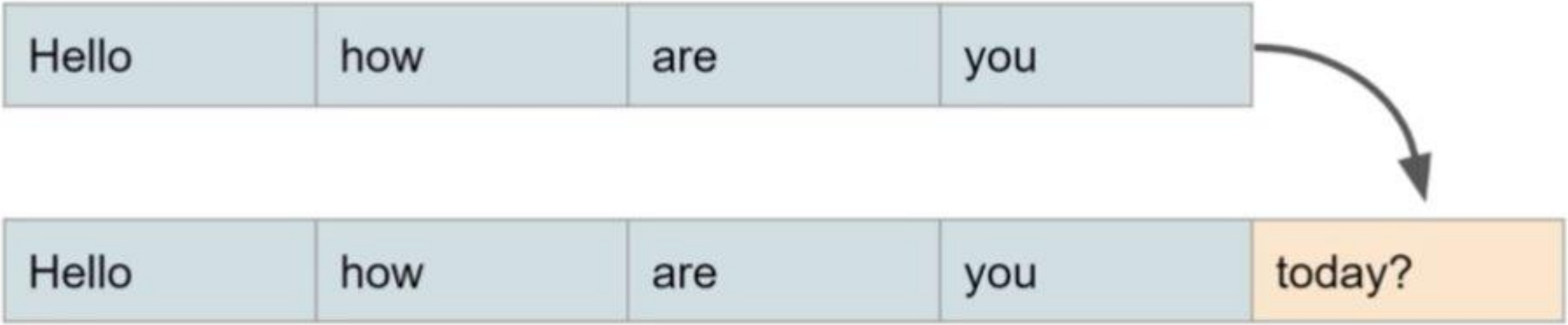

IEEE ComSoc

Machine Learning

- Time Series



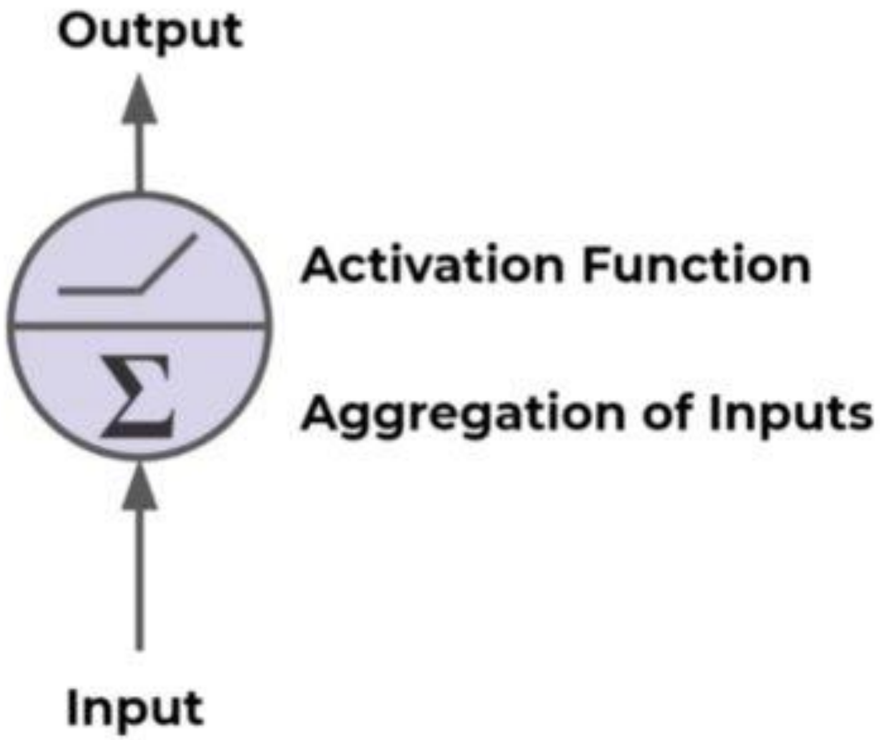
Beer, Wine, and Alcohol Sales
Sales (millions of dollars)
16000
14000
12000
10000
8000
6000
4000
1992
1996
2000
2004
2008
2012
2016
2020



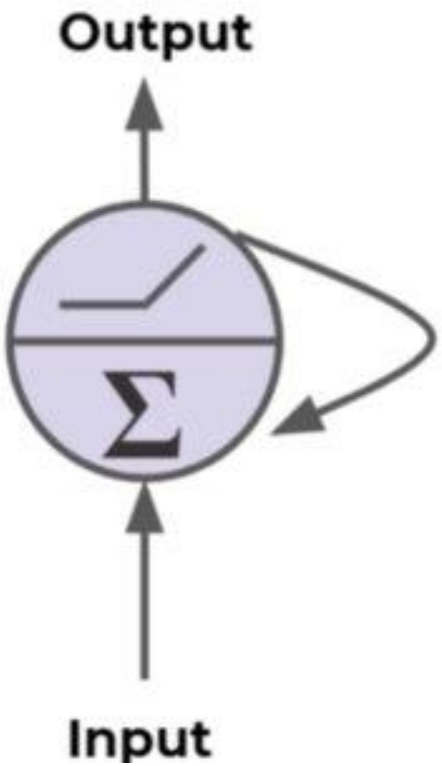

IEEE ComSoc
Machine Learning

# Recurrent Neural Network

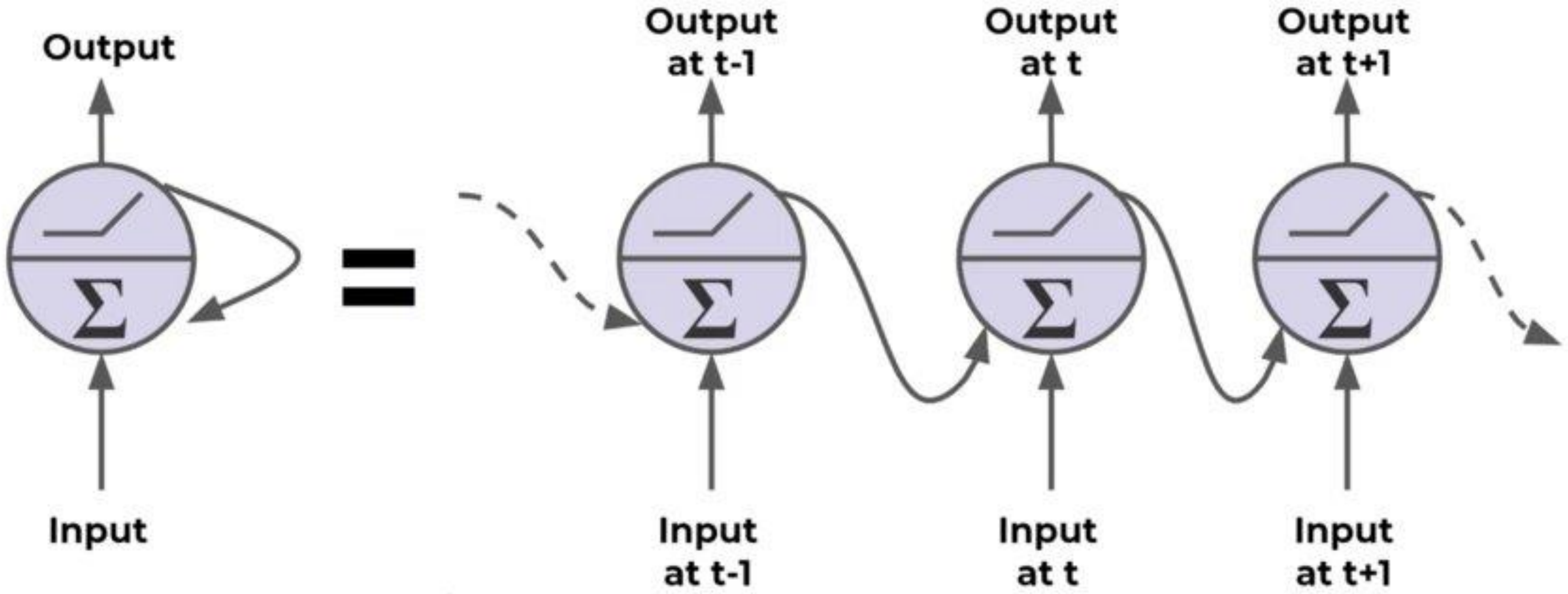
- Normal Neuron in Feed Forward Network



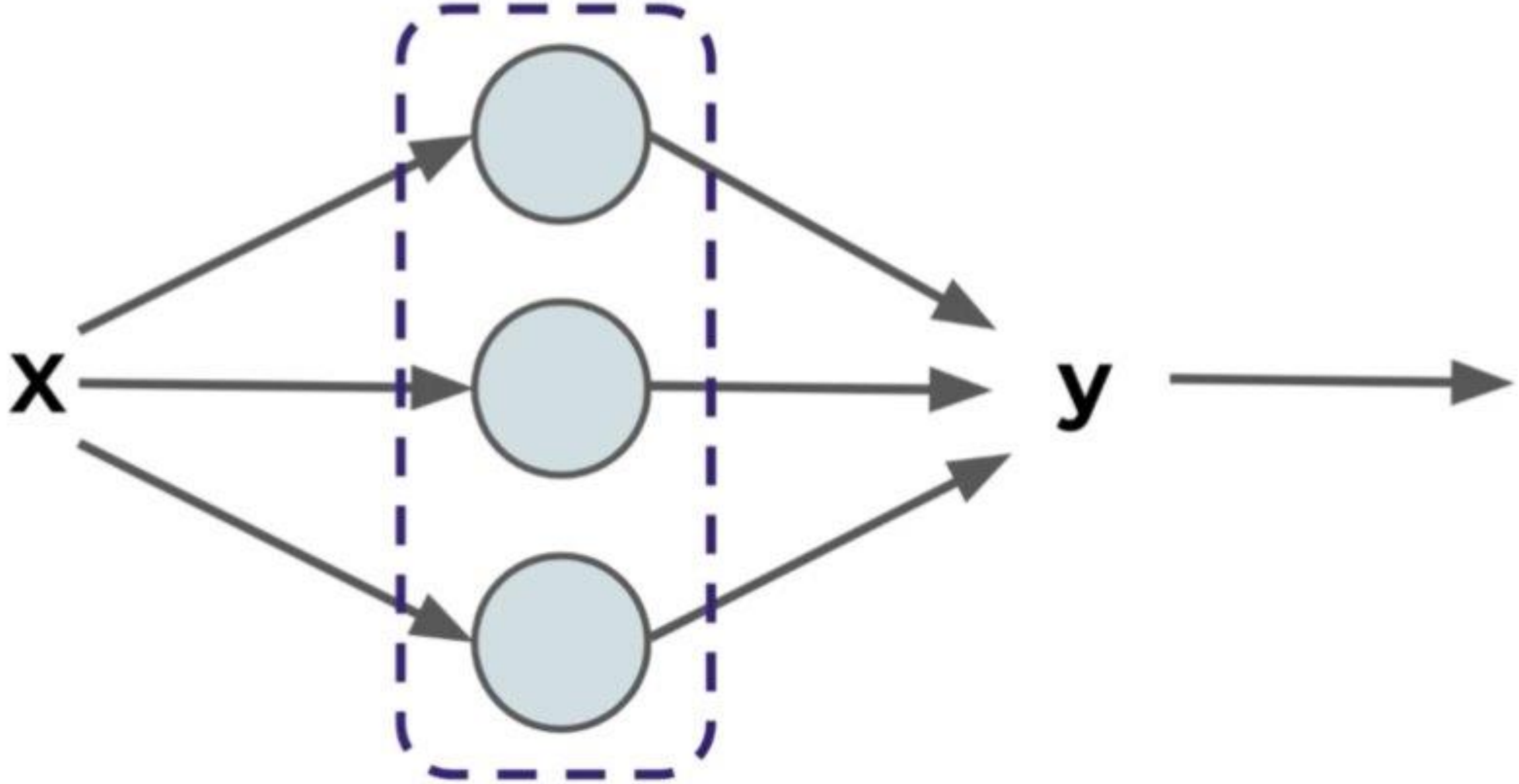
- Recurrent Neuron
  - Sends output back to itself!
  - Let's see what this looks like over time!



- Recurrent Neuron

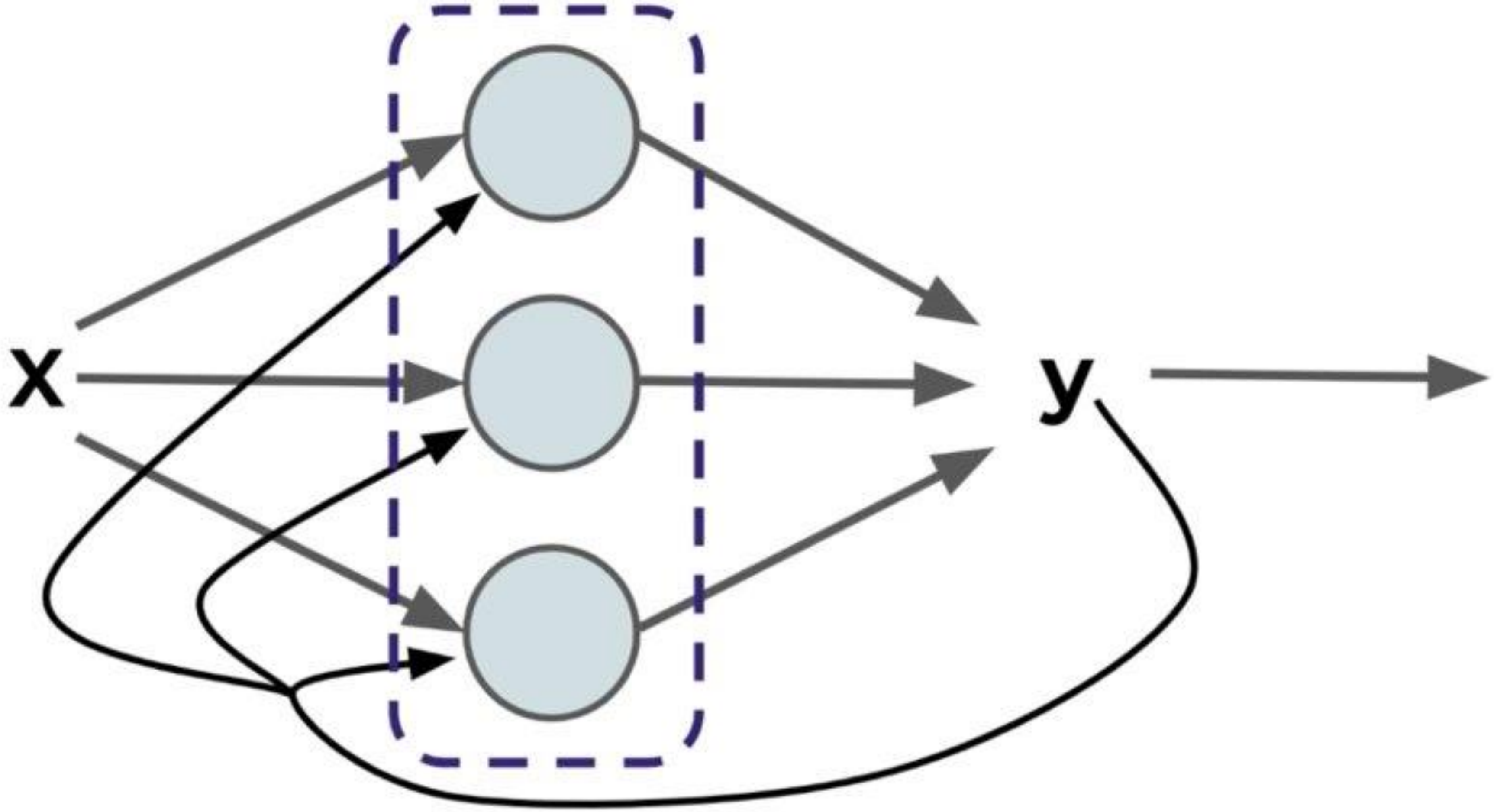


- ANN Layer with 3 Neurons:



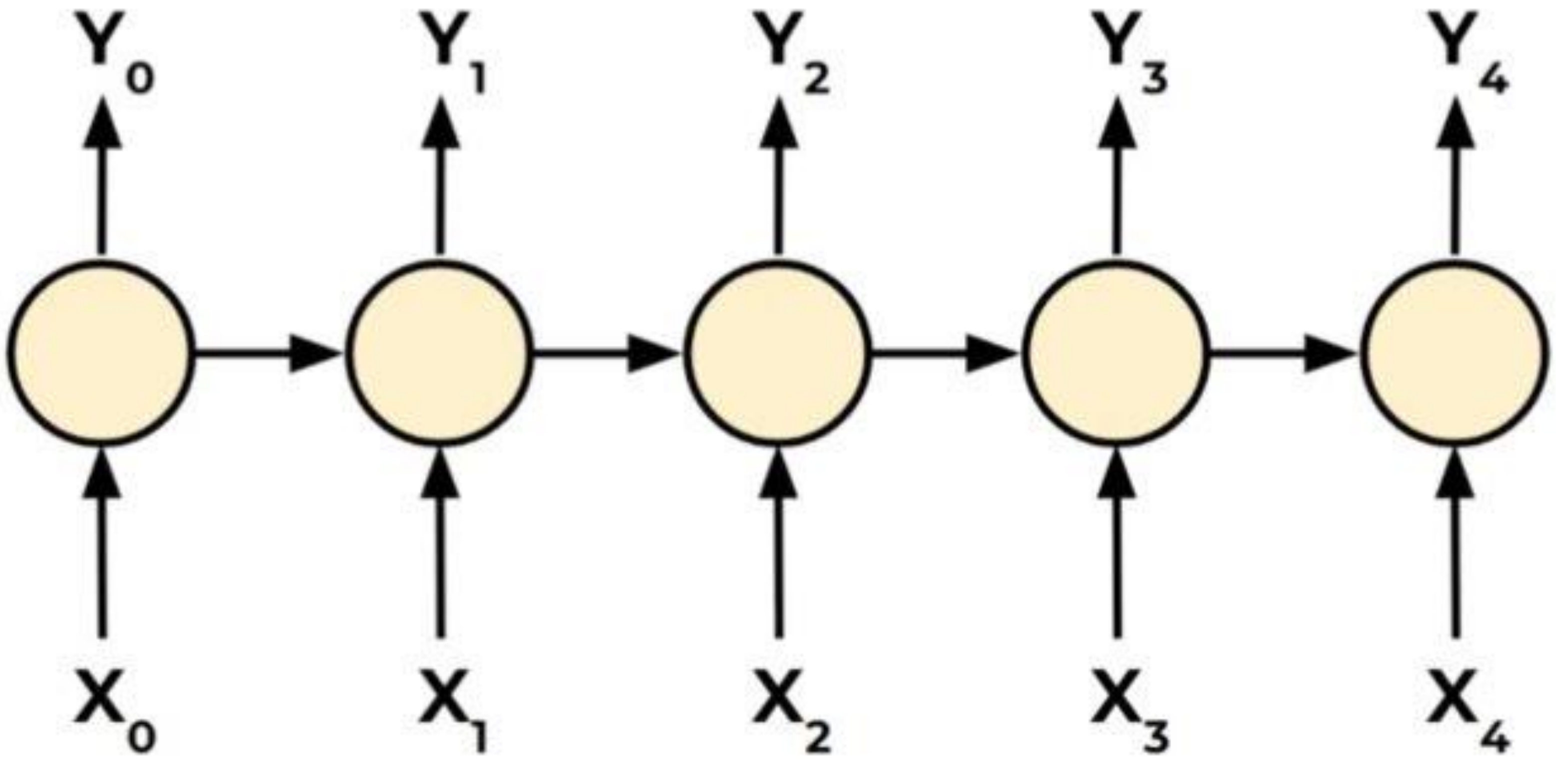

Machine Learning

- RNN Layer with 3 Neurons:

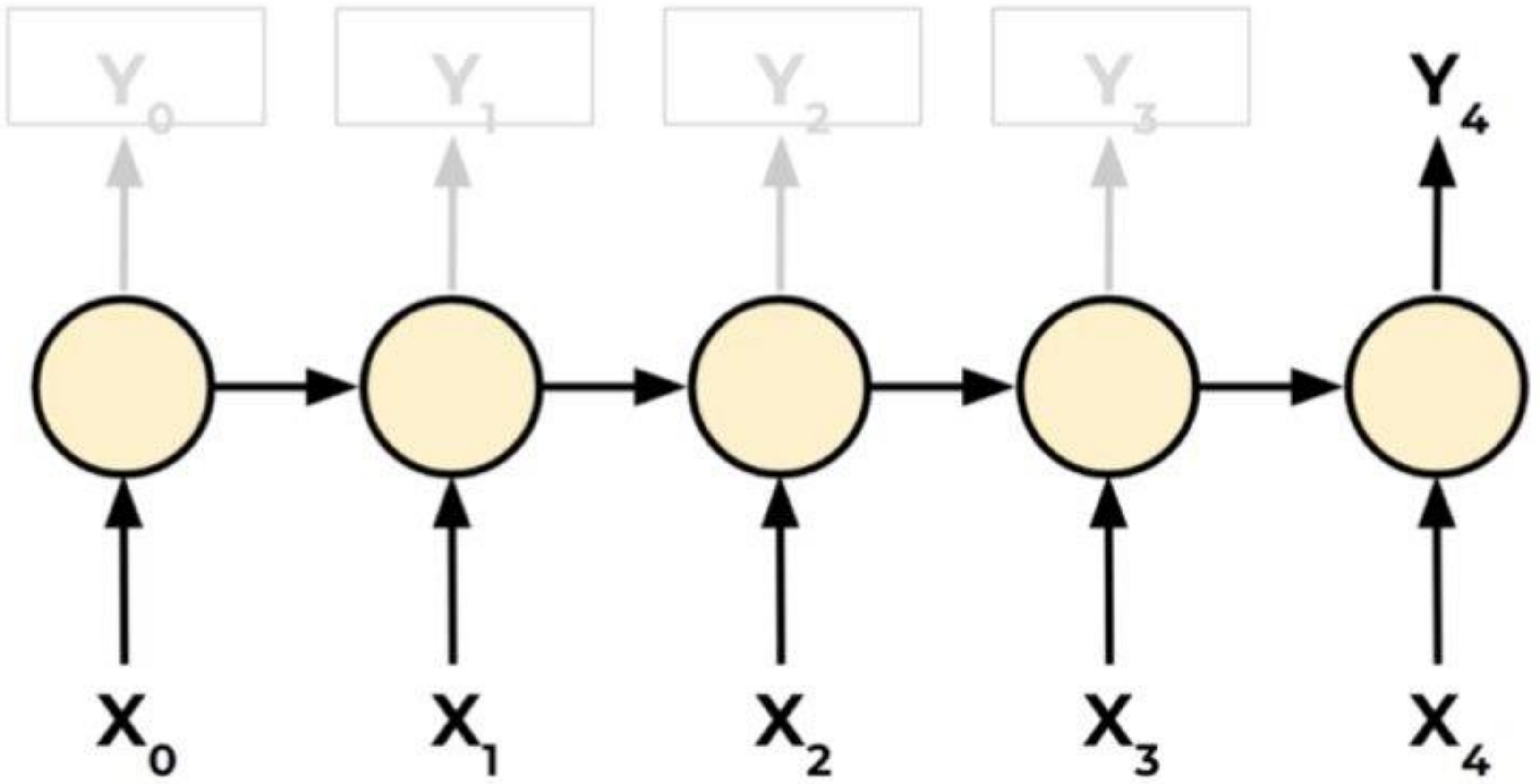


RNNs are flexible with their architecture.

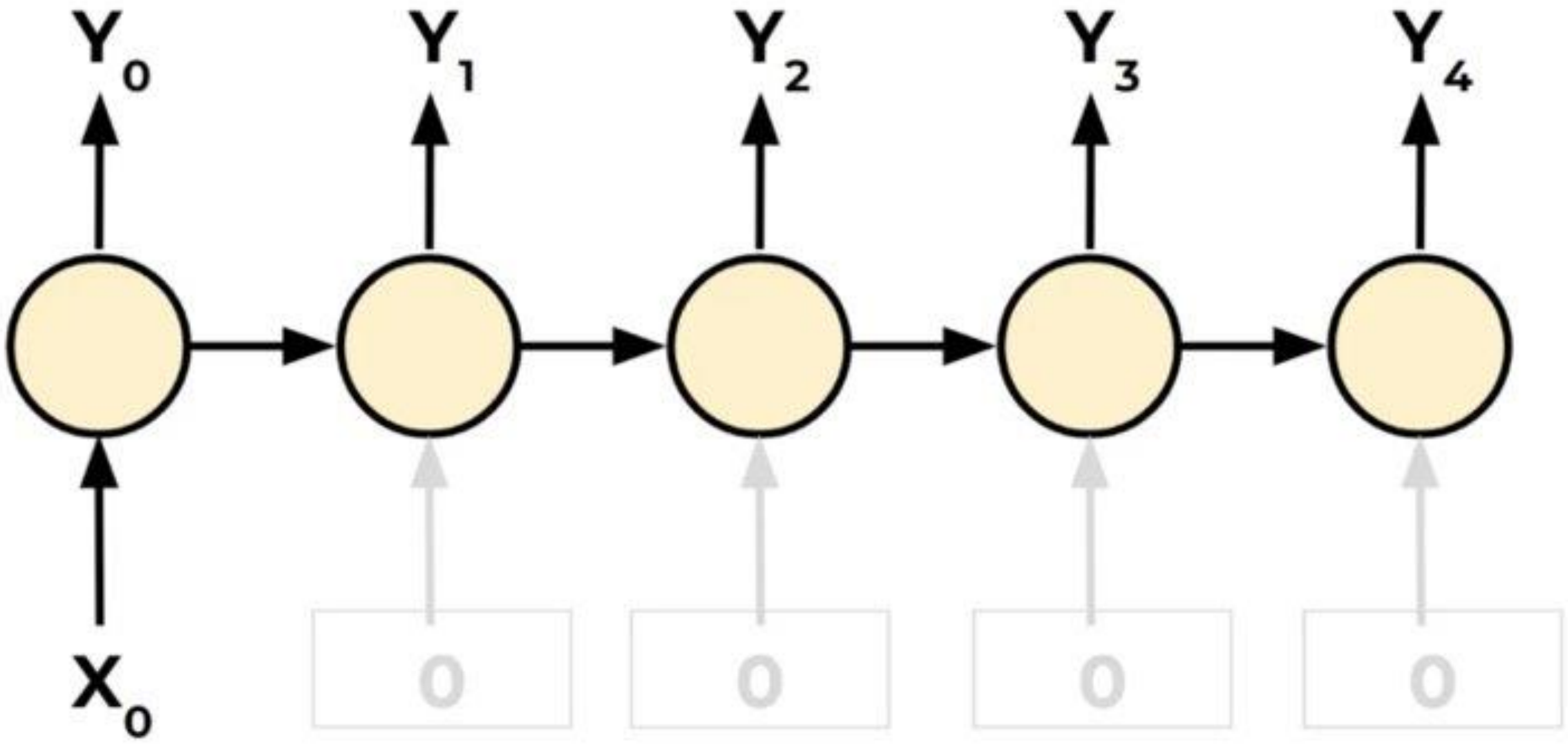
- Sequence to Sequence (Many to Many)



- Sequence to Vector (Many to One)



- Vector to Sequence (One to Many)




IEEE ComSoc

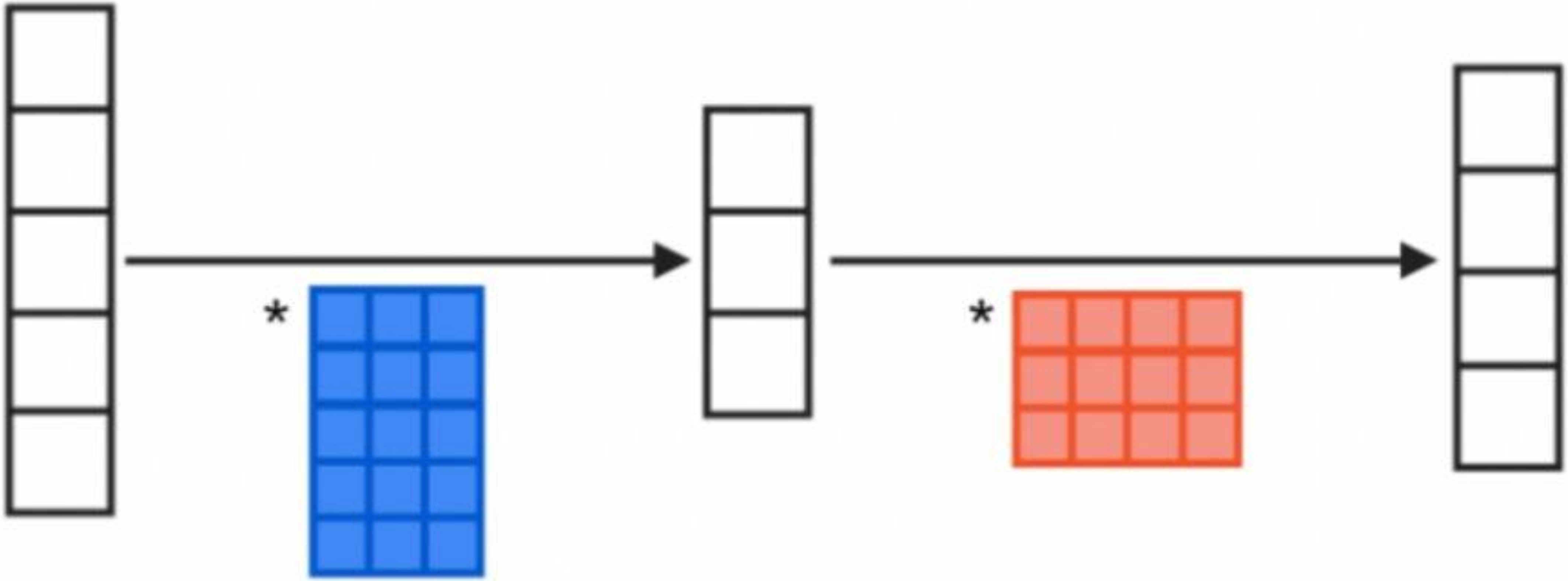
Machine Learning

**one to many:** image -> caption sentence

**many to one:** sentence -> sentiment (positive / negative label)

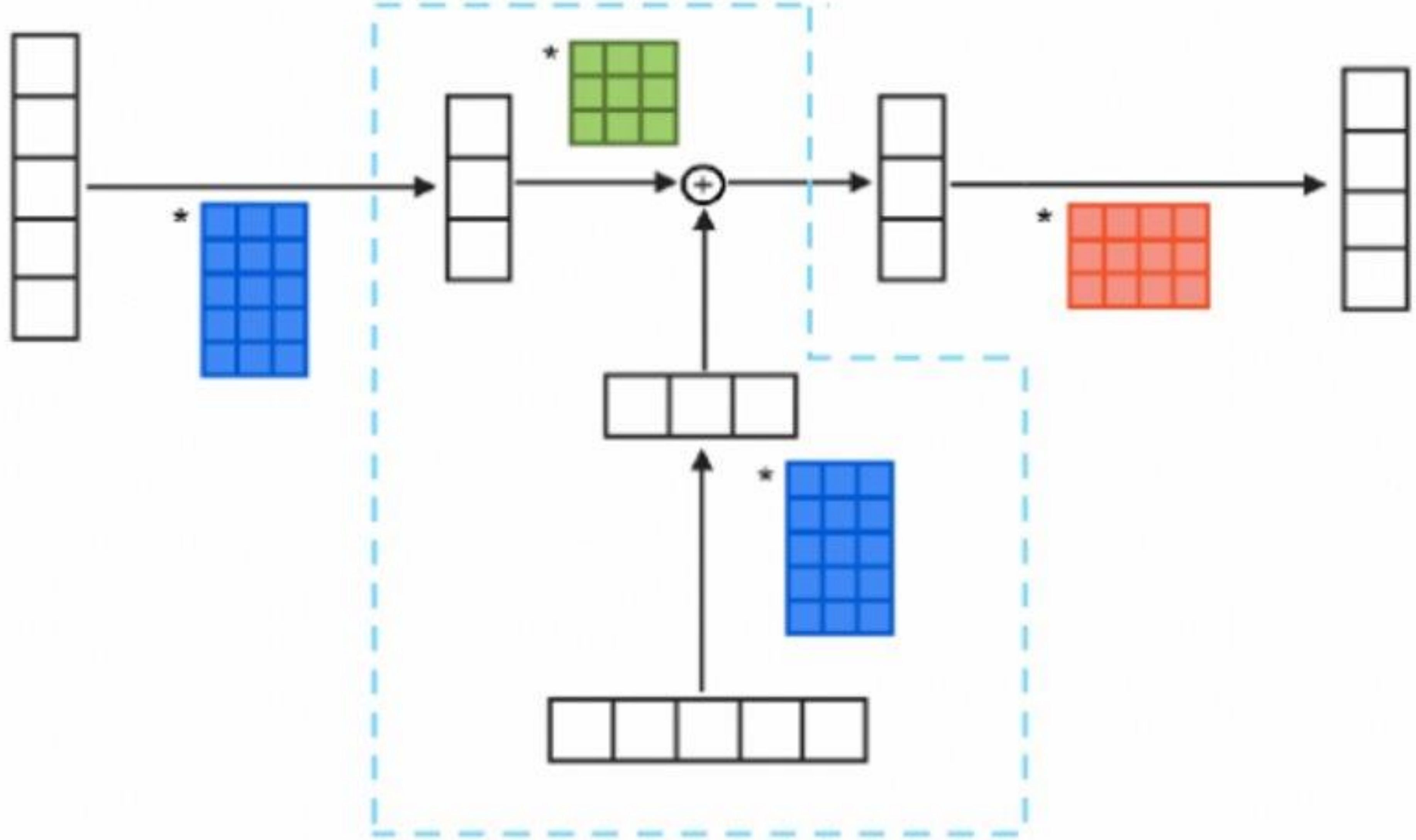
**many to many:** a sentence in English -> a sentence in Turkish

**the other many to many:** frames of video -> coordinates of bounding boxes around an object



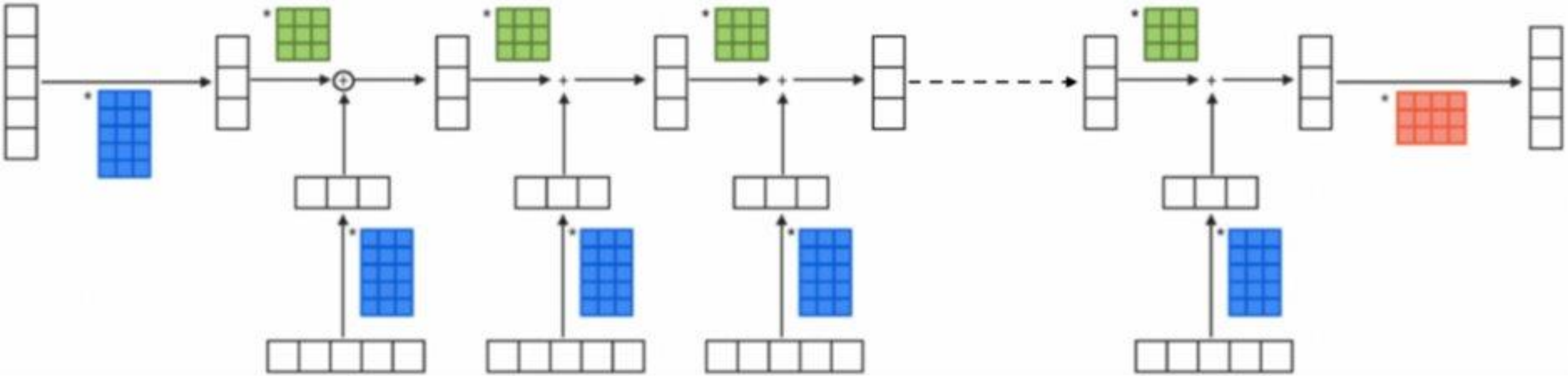

IEEE ComSoc

Machine Learning



IEEE ComSoc

Machine Learning



IEEE ComSoc Machine Learning

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

new state

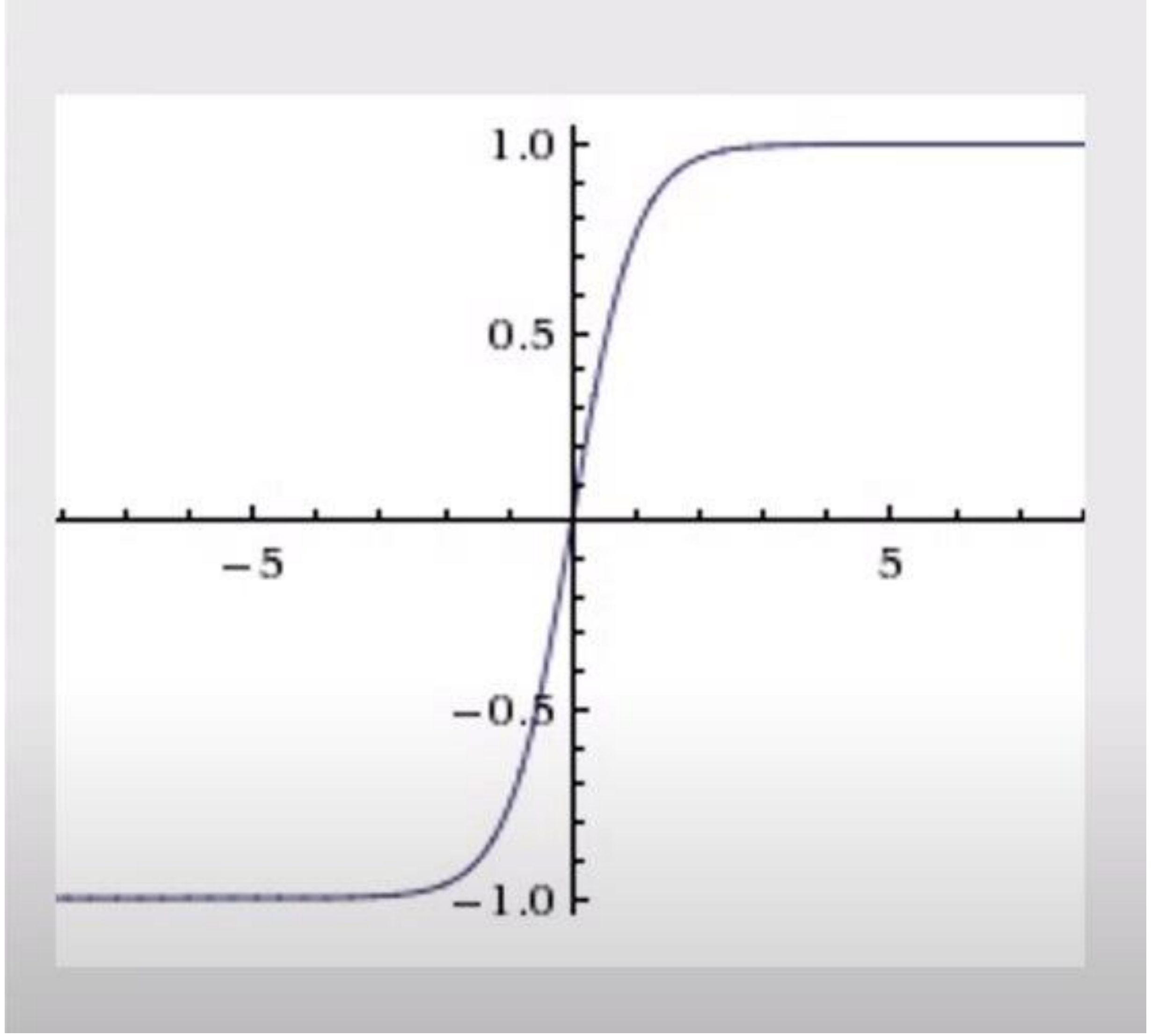
old state

input vector at some time step

some function with parameters W

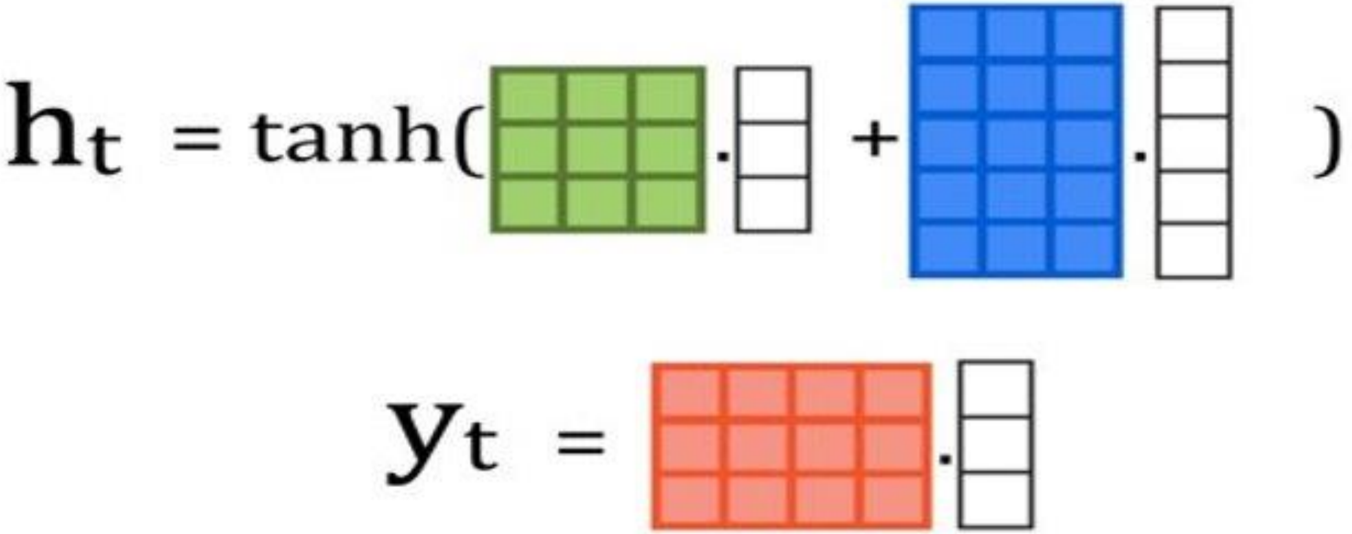
TanH

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



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

$$y_t = W_{hy}h_t$$



- A basic RNN has a major disadvantage, we only really “remember” the previous output.
- It would be great it we could keep track of longer history, not just short term history.

"In France, I had a great time and I learnt some of the ______ language."

our parameters are not trained to capture long-term dependencies, so the word we predict will mostly depend on the previous few words, not much earlier ones

- Another issue that arises during training is the “vanishing gradient”.
- Let’s explore vanishing gradients in more detail before moving on to discussing LSTM (Long Short Term Memory Units).

# backpropagation through time:

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^{2} \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \underbrace{\frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}}_{\text{Contributions of } W \text{ in previous timesteps to the error at timestep } t}$$

we're multiplying a lot of **small numbers** together.

**so what?**

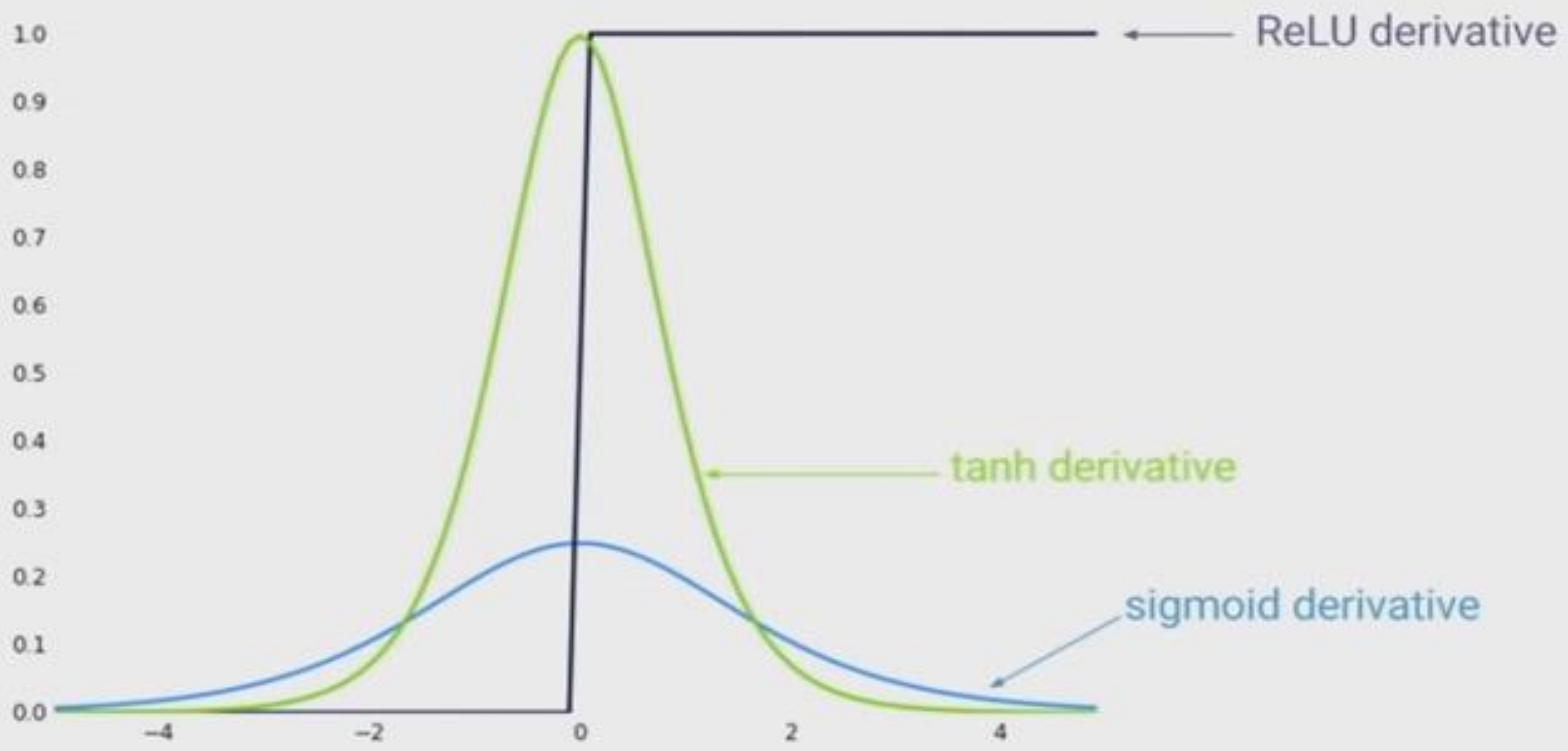
errors due to further back timesteps have increasingly **smaller gradients.**

# How to solve the Vanishing Gradient problem?

Solutions :

- Use another activation function
- Weights initialization (xavier init)
- Use long short term memory (LSTMs)

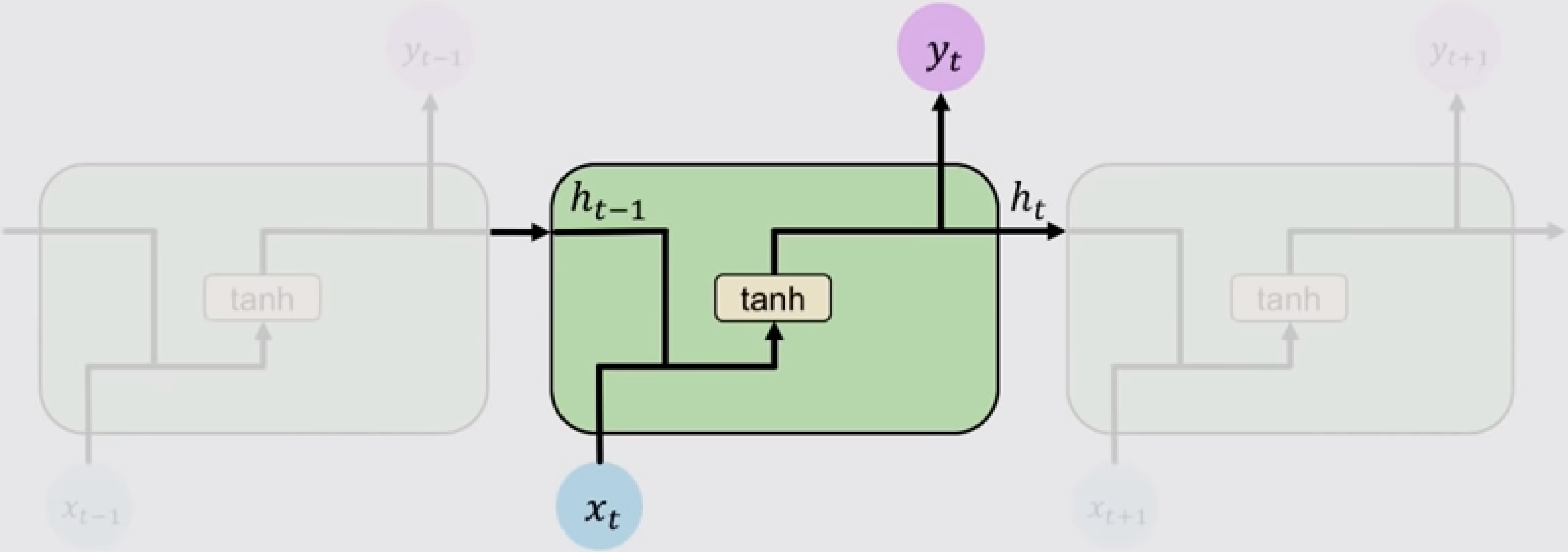
# solution #1: **activation functions**



# Long Short -Term memory cells

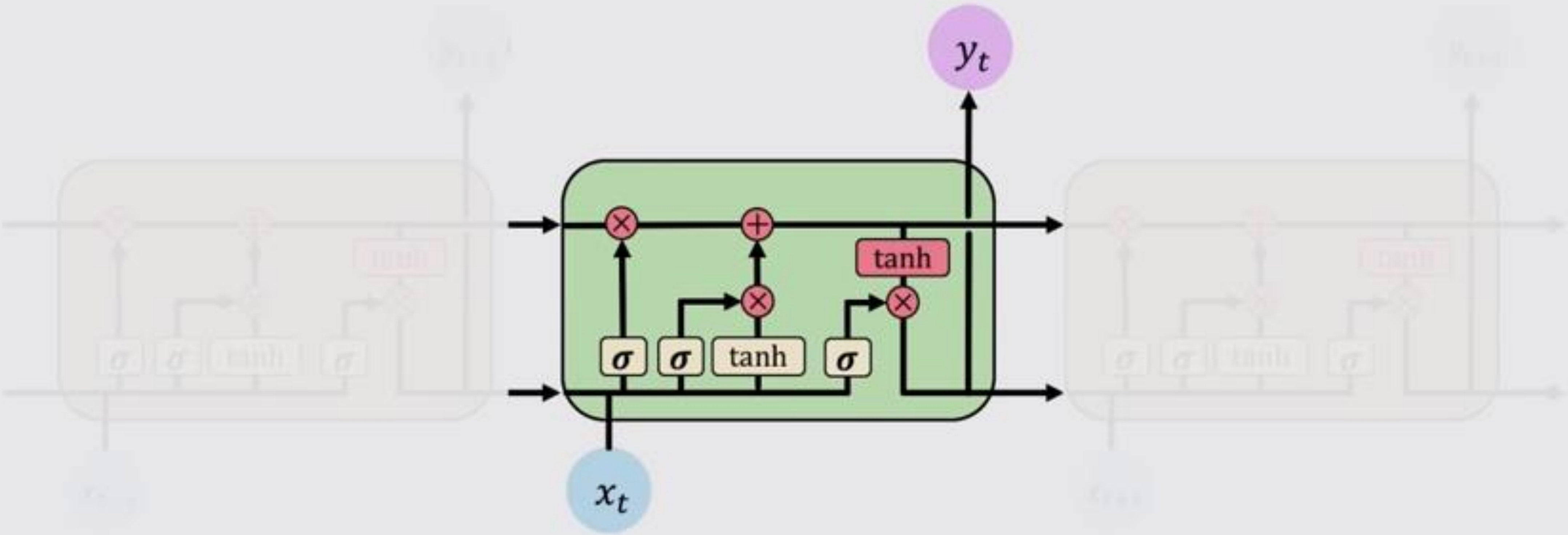
# Standard RNN

In a standard RNN, repeating modules contain a **simple computation node**

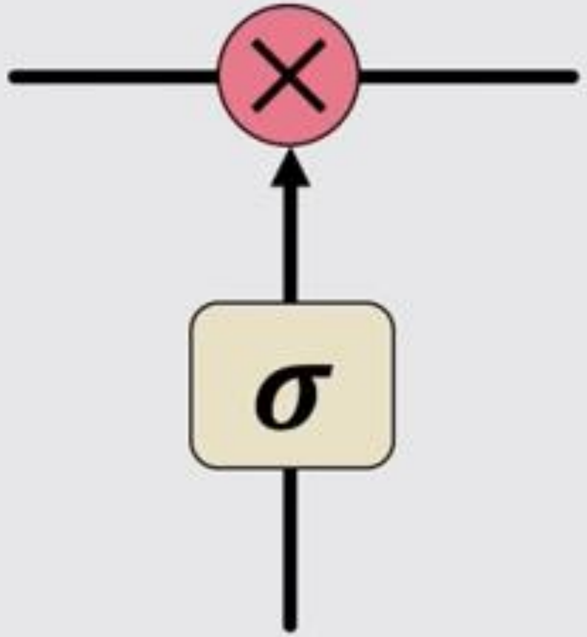


# Long Short Term Memory (LSTMs)

LSTM modules contain **computational blocks** that **control information flow**



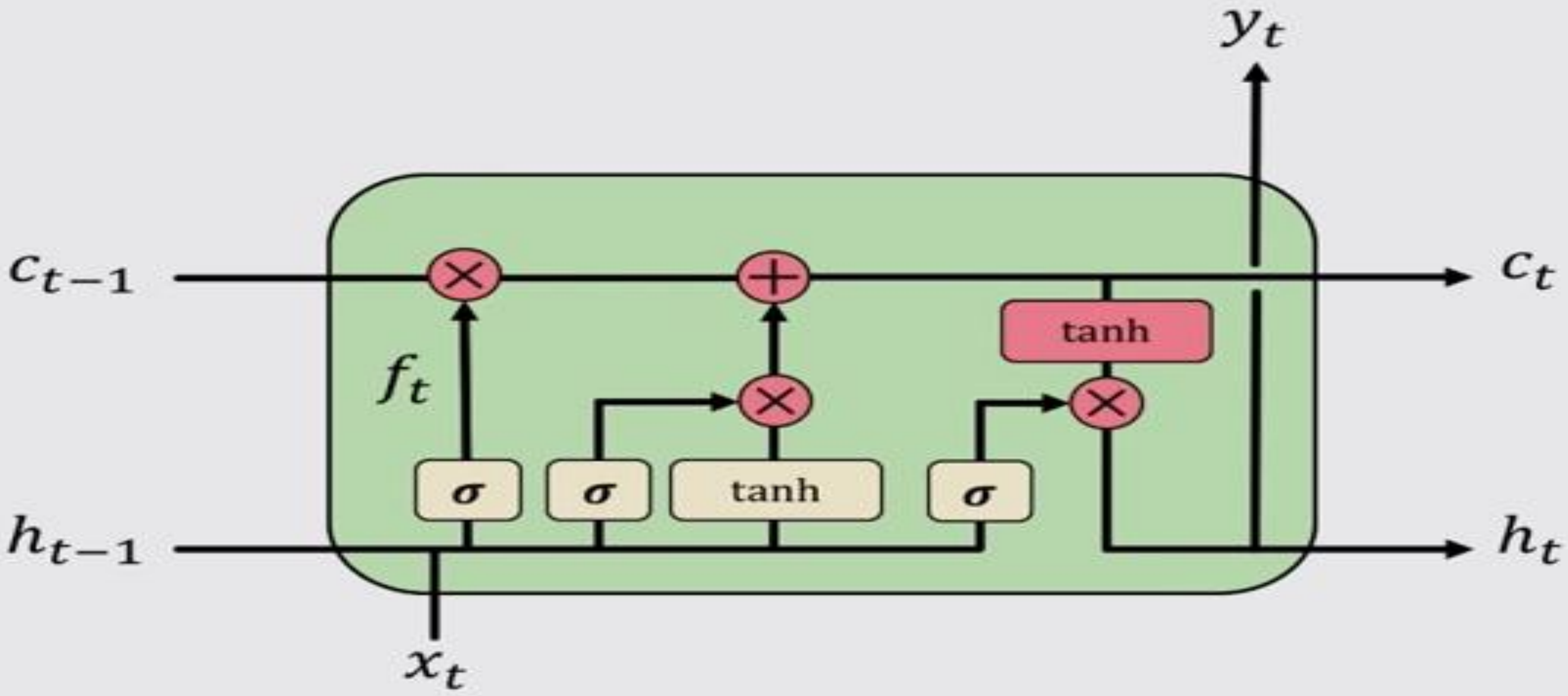
Information is **added** or **removed** through structures called **gates**




Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

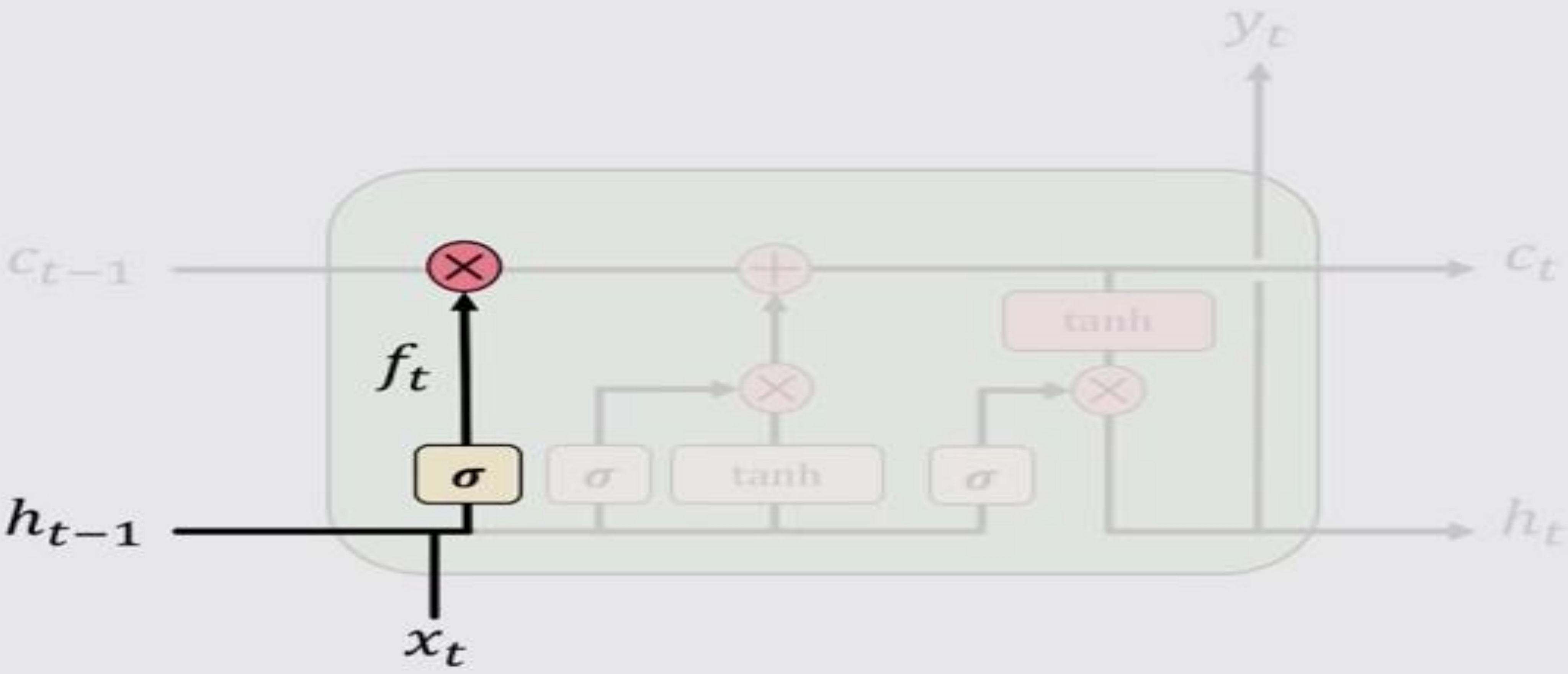
# How do LSTMs work?

## 1) Forget 2) Store 3) Update 4) Output



**1) Forget** 2) Store 3) Update 4) Output

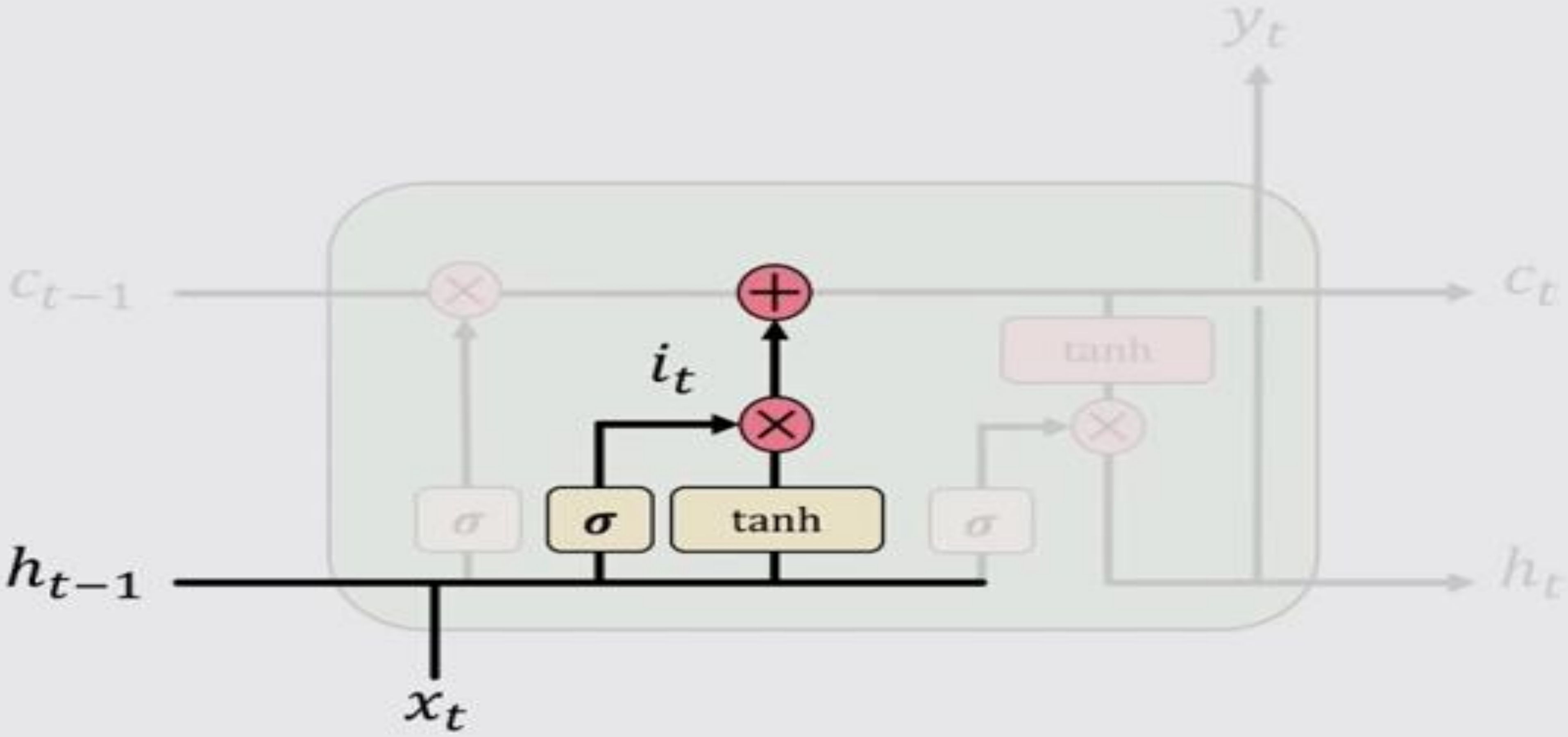
LSTMs **forget irrelevant** parts of the previous state




IEEE ComSoc Machine Learning

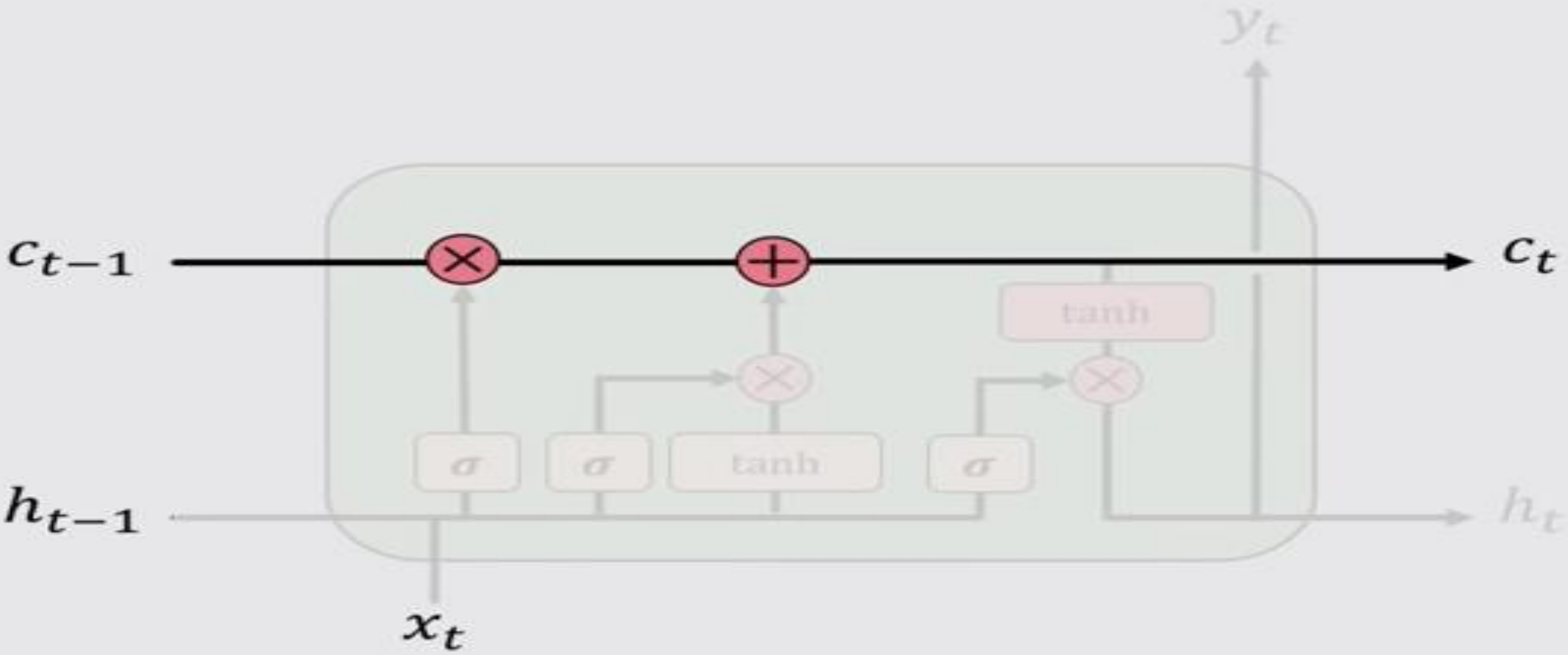
1) Forget **2) Store** 3) Update 4) Output

LSTMs **store relevant** new information into the cell state



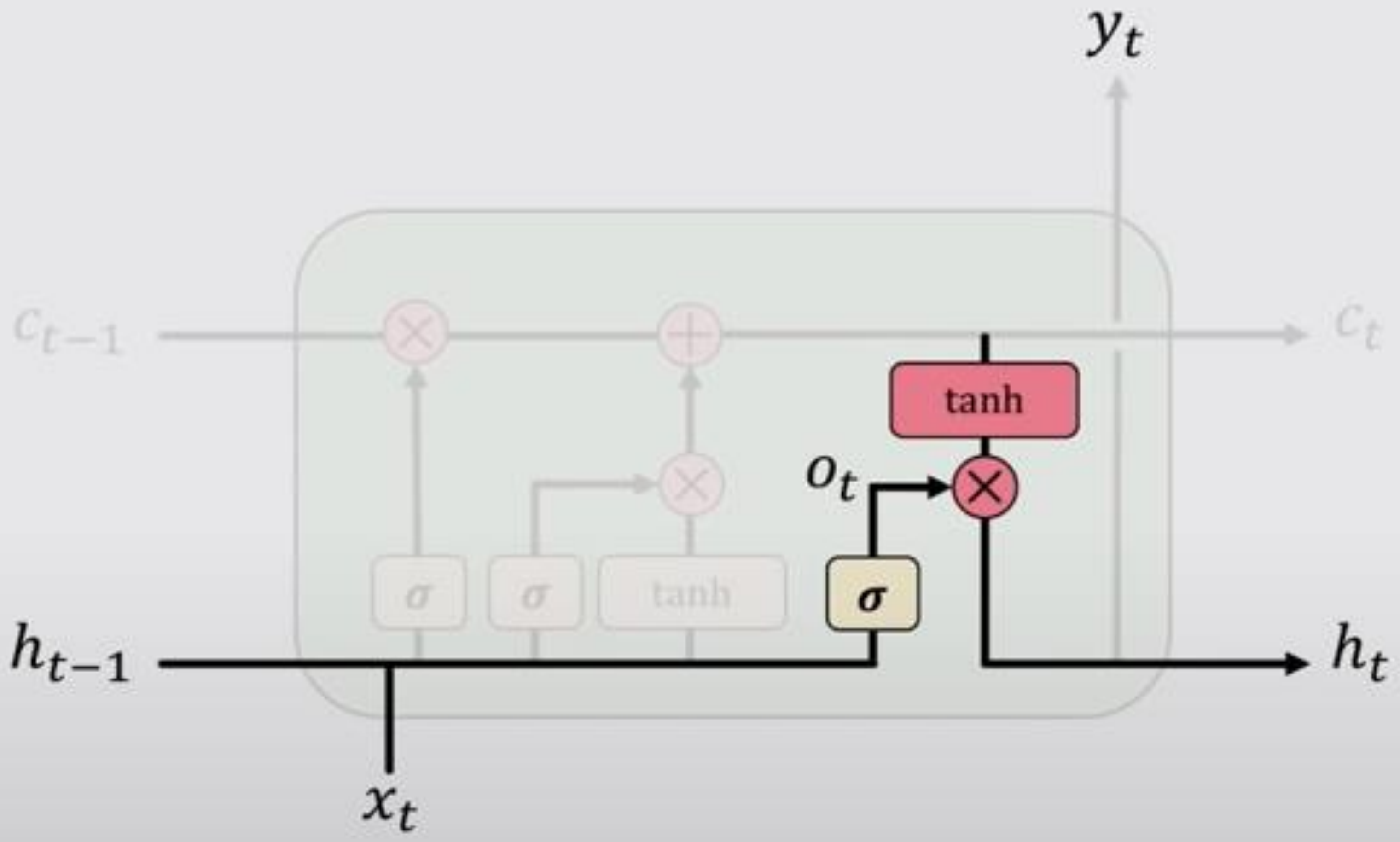
1) Forget 2) Store **3) Update** 4) Output

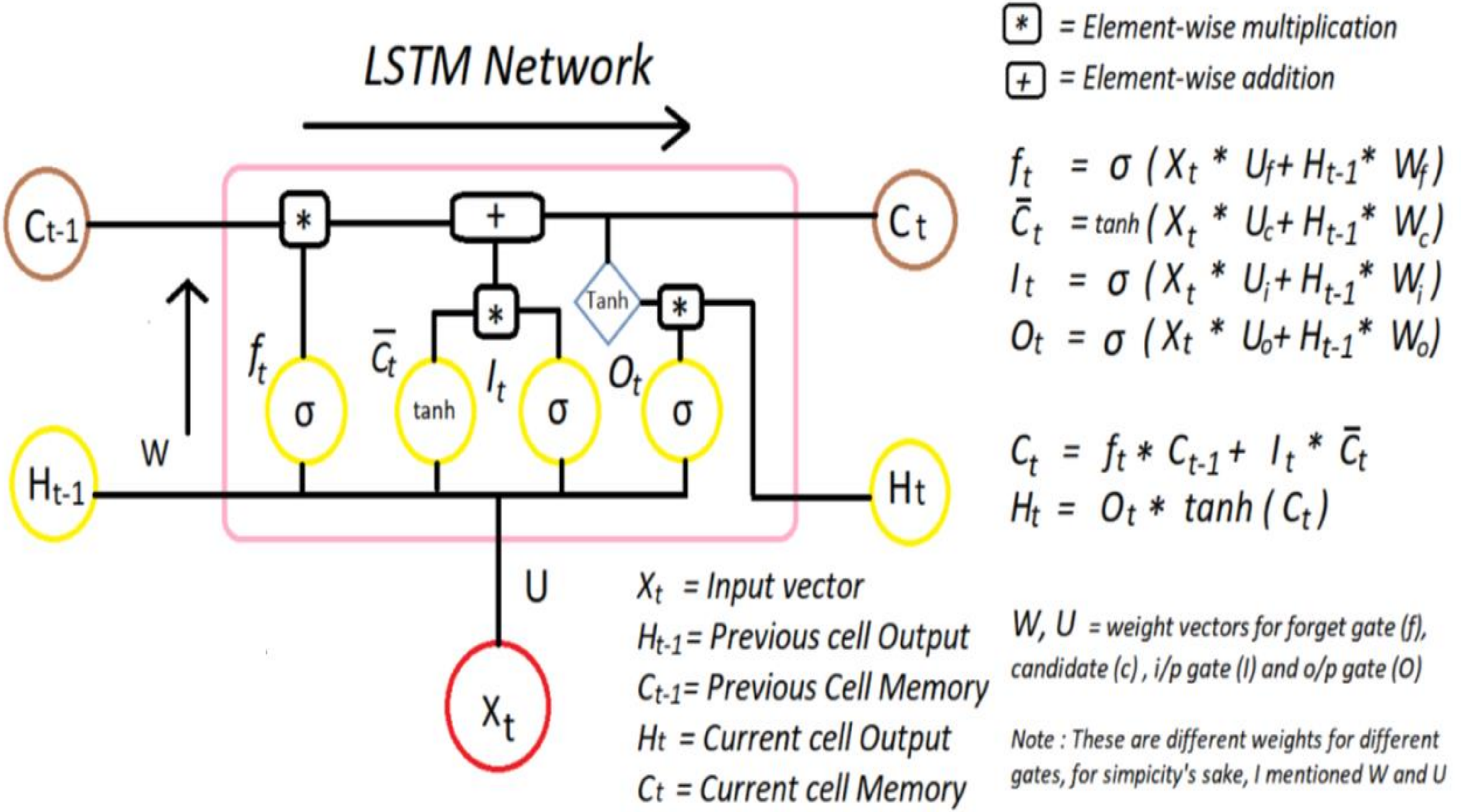
LSTMs **selectively update** cell state values



1) Forget 2) Store 3) Update **4) Output**

The **output gate** controls what information is sent to the next time step





LSTM Network
$C_{t-1}$
$H_{t-1}$
W
U
$X_t$
$f_t$
$\bar{C}_t$
$I_t$
$O_t$
σ
tanh
σ
σ
Tanh
$C_t$
$H_t$
* = Element-wise multiplication
+ = Element-wise addition
$f_t = \sigma\,(X_t * U_f + H_{t-1} * W_f)$
$\bar{C}_t = \tanh\,(X_t * U_c + H_{t-1} * W_c)$
$I_t = \sigma\,(X_t * U_i + H_{t-1} * W_i)$
$O_t = \sigma\,(X_t * U_o + H_{t-1} * W_o)$
$C_t = f_t * C_{t-1} + I_t * \bar{C}_t$
$H_t = O_t * \tanh\,(C_t)$
$X_t$ = Input vector
$H_{t-1}$ = Previous cell Output
$C_{t-1}$ = Previous Cell Memory
$H_t$ = Current cell Output
$C_t$ = Current cell Memory
W, U = weight vectors for forget gate (f), candidate (c), i/p gate (I) and o/p gate (O)
Note : These are different weights for different gates, for simpicity's sake, I mentioned W and U

# LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with **uninterrupted gradient flow**

Let's Code !

IEEE ComSoc

Machine Learning

# Where to go from here :

- **How to deploy your models (flask ,dockor …)**
- **Autoencoders**
- **GANS (GENERATIVE ADVERSERIAL NETWORKS)**
- **Self Organizing Maps**
- **Reinforcement learning**
- **FastAi Course (very very important)**
- **NLP**
- **ComputerVision**
- **Speech Processing**
- **And Moreeeeeeeeeeeeeeeeeeeeeeeeeee**


Thank you