

# Recurrent Neural Networks ( RNN )

# Introduction to RNN

## Why We Need RNN?

- RNNs are used for sequential data analysis.
- When we need to make a prediction based on previous data and the current data, we use RNNs because they make use of sequential information.
- RNNs are called **recurrent** because they perform the same task for every element of a sequence.
- RNNs have a **memory** which stores the previous information

# Sequential Data

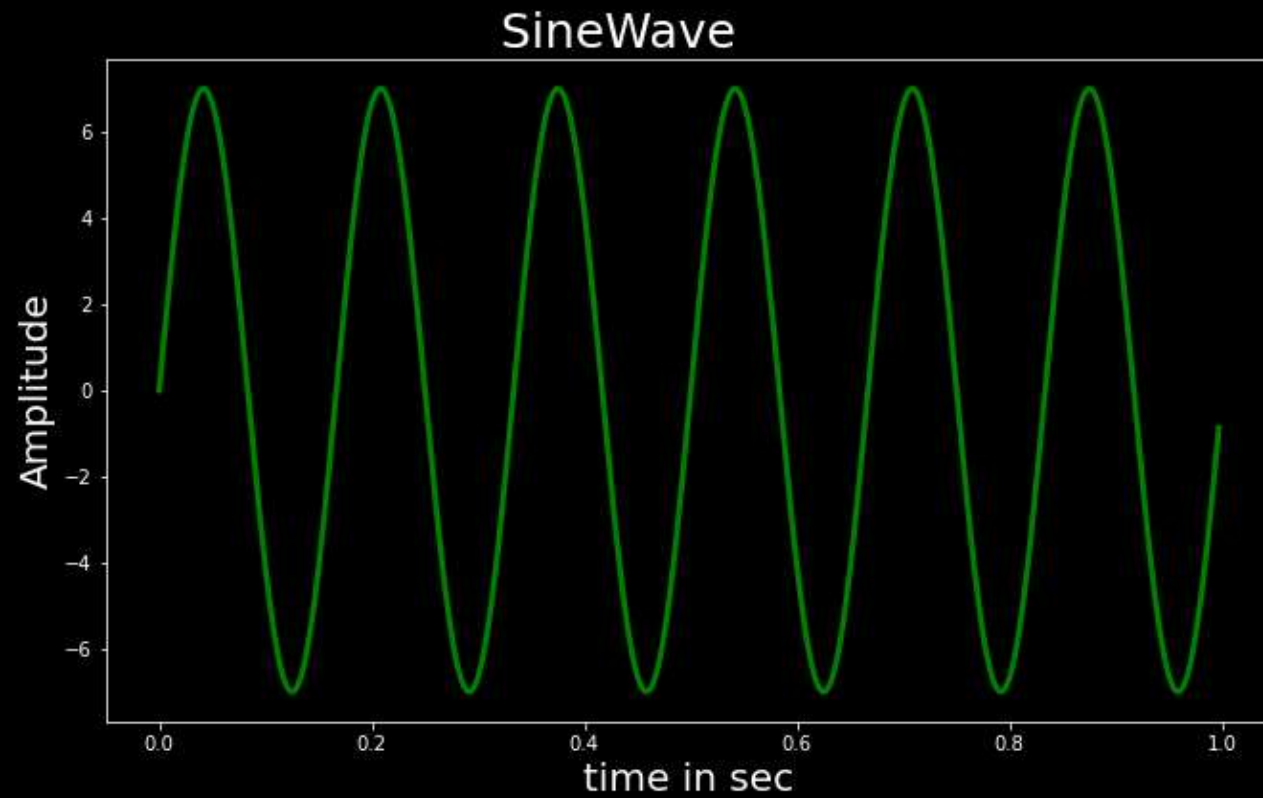
# Sequential Data

If the data samples in the dataset are dependent on the other samples and they have temporal correlation with each other then the data is said to be a Sequential data.

## Examples

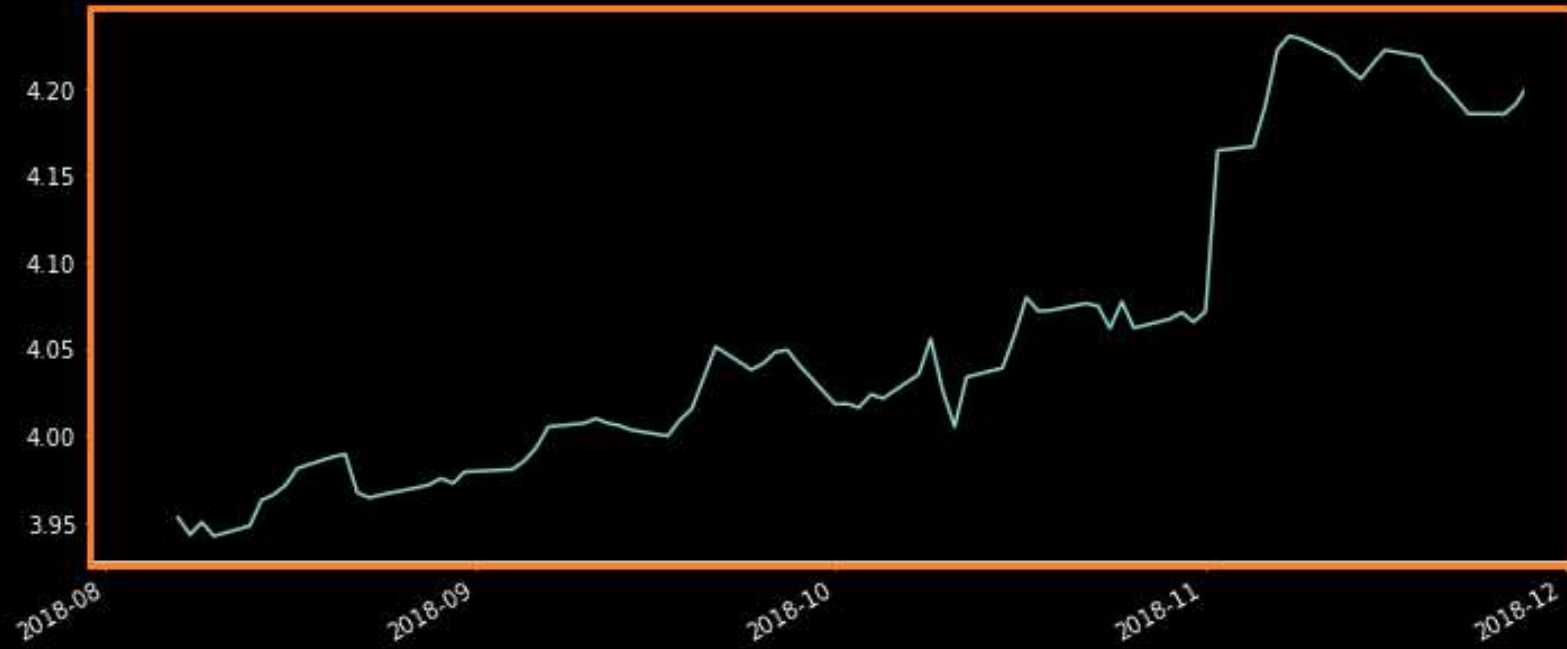
- Time Series data
- Text Data

# Time Series Data

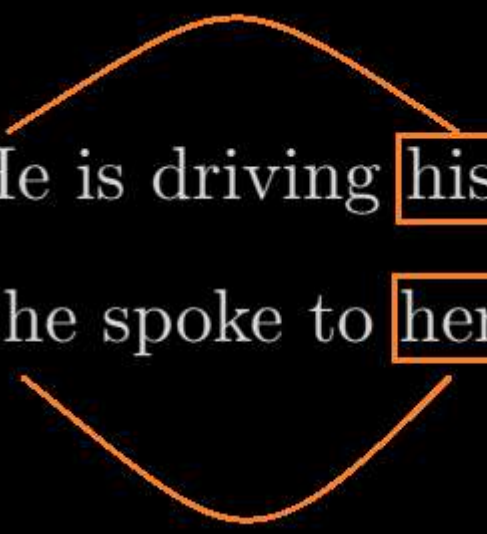


# Time Series Data

Stock Price Data



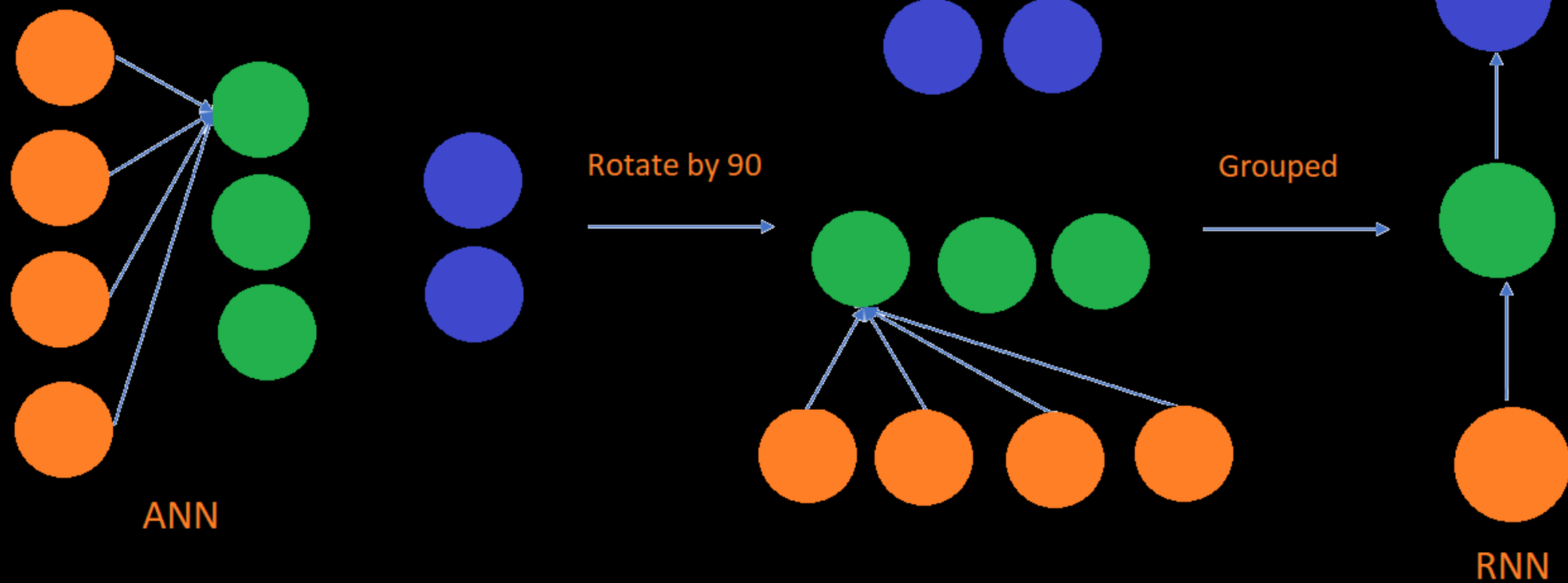
## Text Data

- He is driving his car.
  - She spoke to her mother.
- 

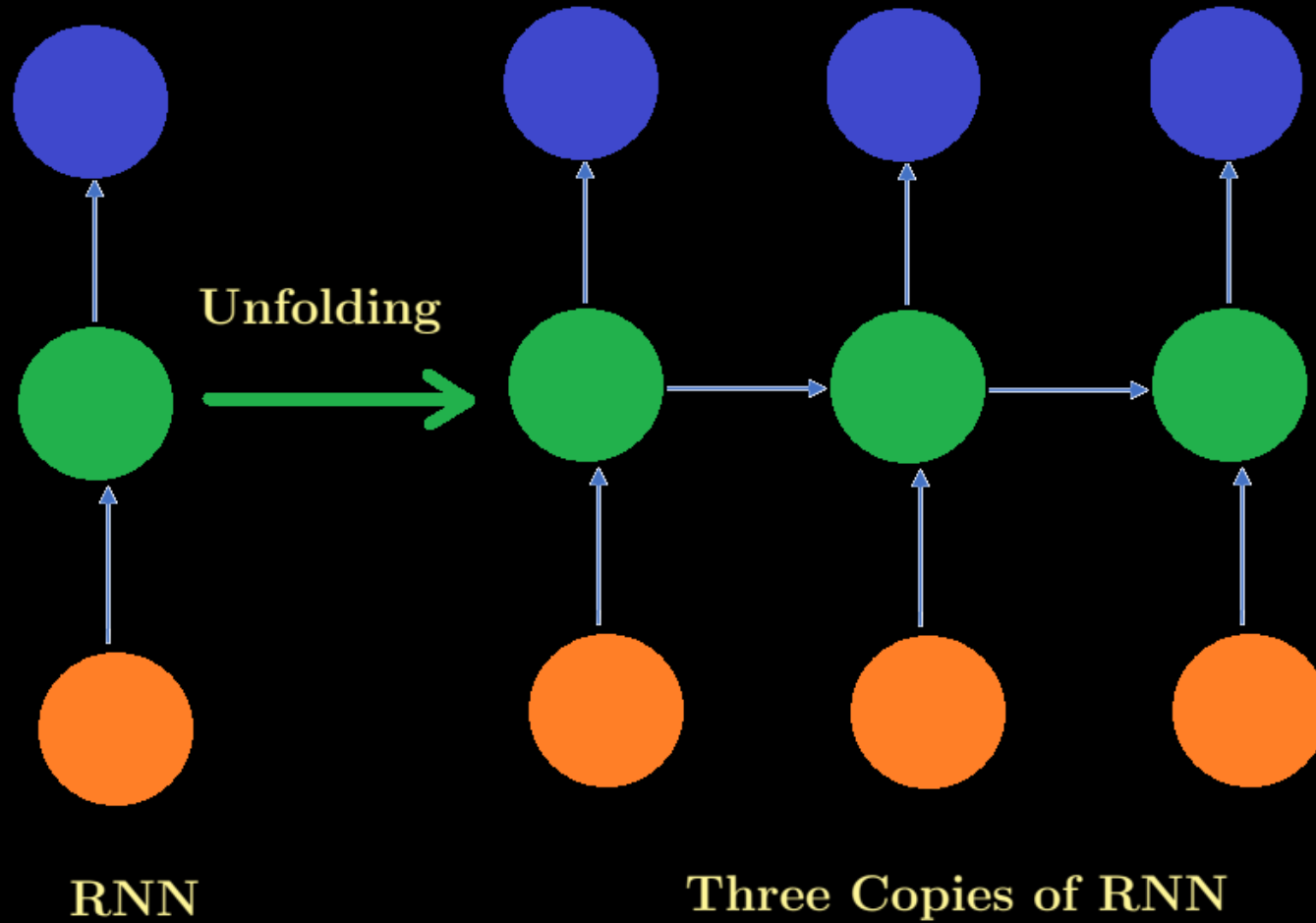


# ANN to RNN

# ANN to RNN

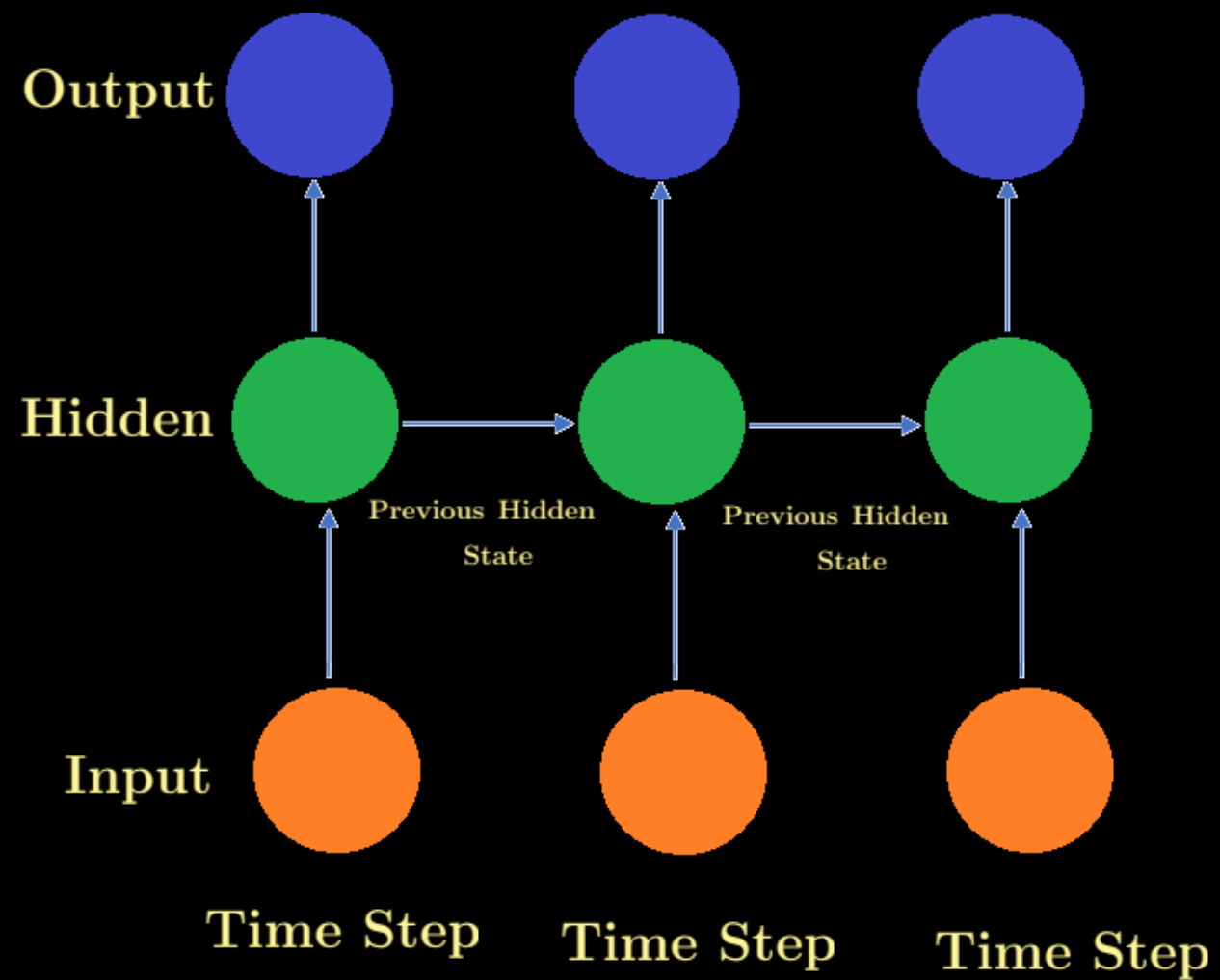


# Unfolding RNN

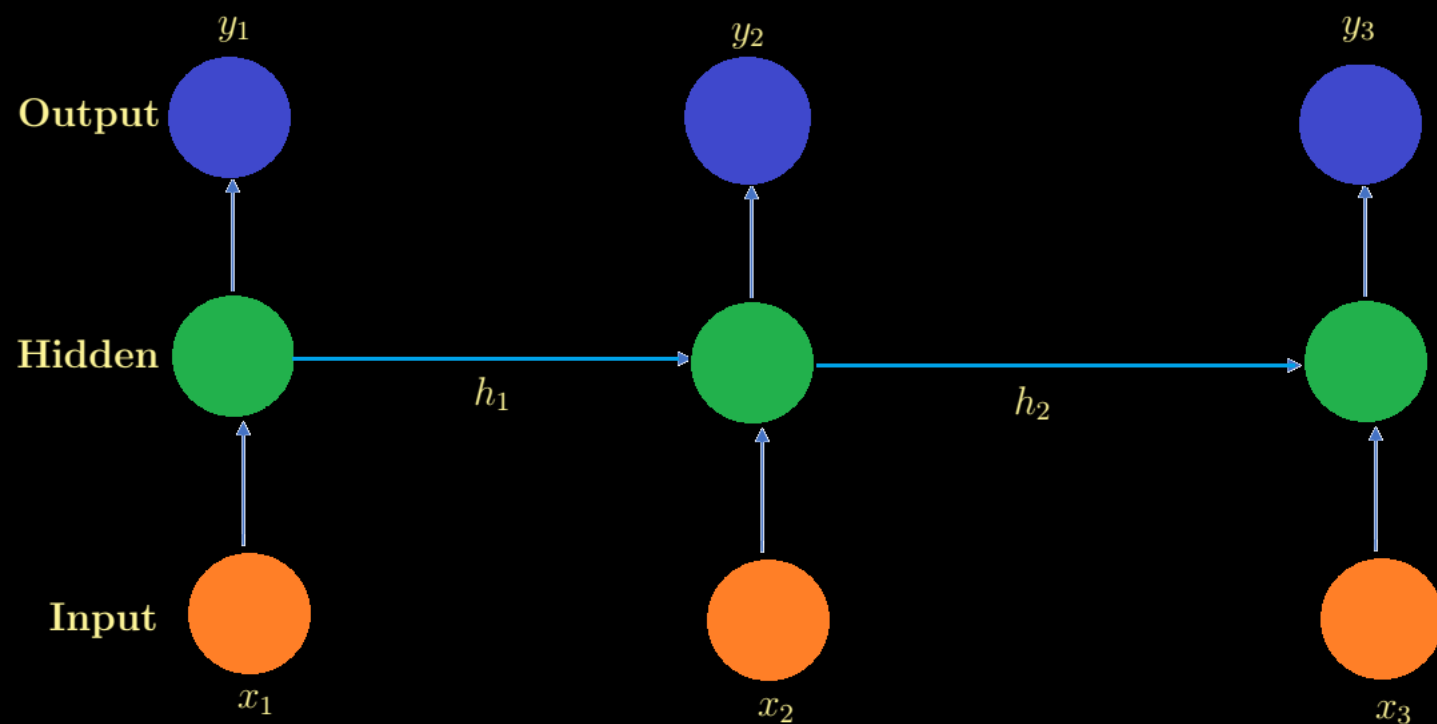


Each copy of RNN is called time step because it receives different inputs at different times

# Time Steps



# RNN Equations



$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$

$$y_t = W_y h_t$$

$$h_1 = \tanh(W_h h_0 + W_x x_1)$$

$$y_1 = W_y h_1$$

$$h_2 = \tanh(W_h h_1 + W_x x_2)$$

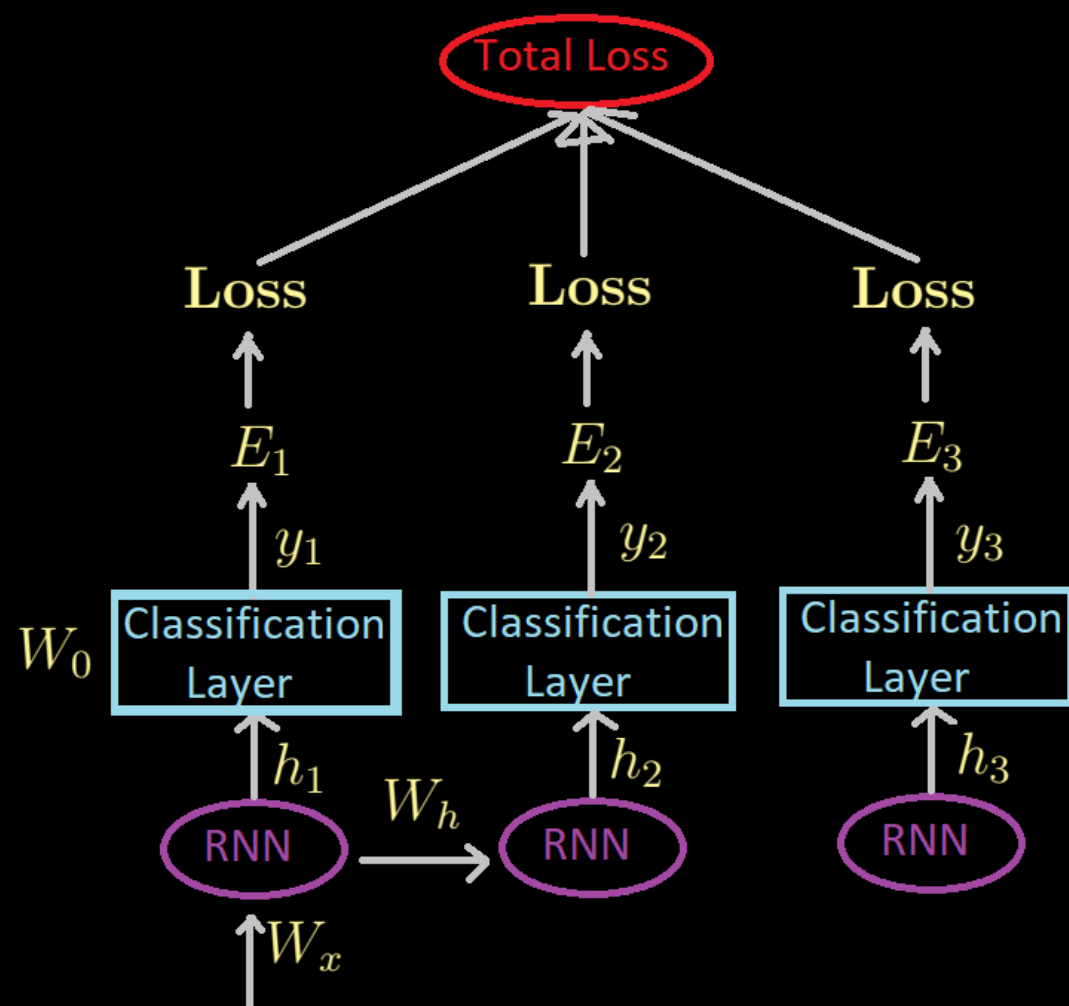
$$y_2 = W_y h_2$$

$$h_3 = \tanh(W_h h_2 + W_x x_3)$$

$$y_3 = W_y h_3$$

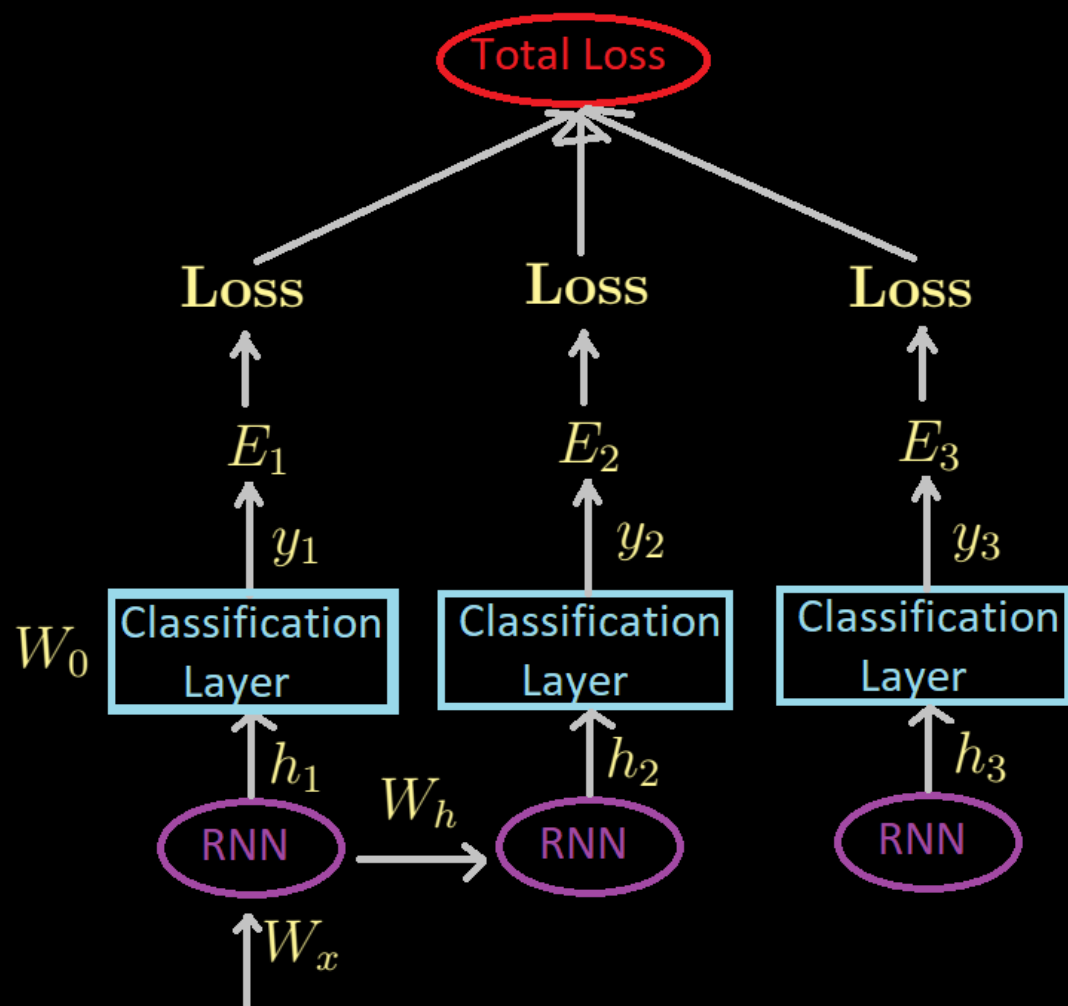
# Back Propagation Through Time ( BPTT )

# Back Propagation Through Time ( BPTT )



$W_x, W_h, W_0$  are shared  
along each copy of RNN

# Back Propagation Through Time ( BPTT )



## Updating $W_0$

$$\frac{\partial E_1}{\partial W_0} = \frac{\partial E_1}{\partial y_1} \frac{\partial y_1}{\partial W_0}$$

$$\frac{\partial E_2}{\partial W_0} = \frac{\partial E_2}{\partial y_2} \frac{\partial y_2}{\partial W_0}$$

$$\frac{\partial E_3}{\partial W_0} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial W_0}$$

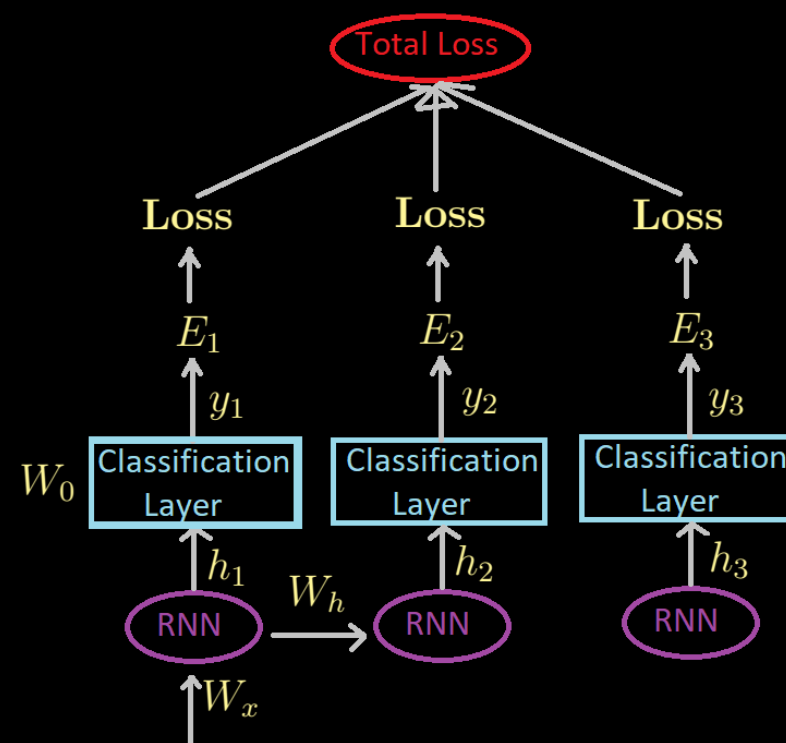


# Updating $W_h$

$$\frac{\partial E_1}{\partial W_h} = \frac{\partial E_1}{\partial y_1} \frac{\partial y_1}{\partial h_1} \frac{\partial h_1}{\partial W_h}$$

$$\frac{\partial E_2}{\partial W_h} = \frac{\partial E_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_h} + \frac{\partial E_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial W_h}$$

$$\frac{\partial E_3}{\partial W_h} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_h} + \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W_h} + \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial W_h}$$



# Vanishing Gradient Problem

$$\frac{\partial E_2}{\partial W_h} = \boxed{\frac{\partial E_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_h}} + \frac{\partial E_2}{\partial y_2} \frac{\partial y_2}{\partial h_2} \frac{\partial h_2}{\partial W_h}$$

$$= 0.2 \times 0.1 \times 0.01 \times 0.02$$

$$= 0.000004$$

# Vanishing Gradient Problem

- During the vanishing gradient problems, the early layers of RNN vanishes or do not take part in learning.
- Earlier layers are important because they extract low level features that are keys of accurate prediction.

# Vanishing Gradient Problem



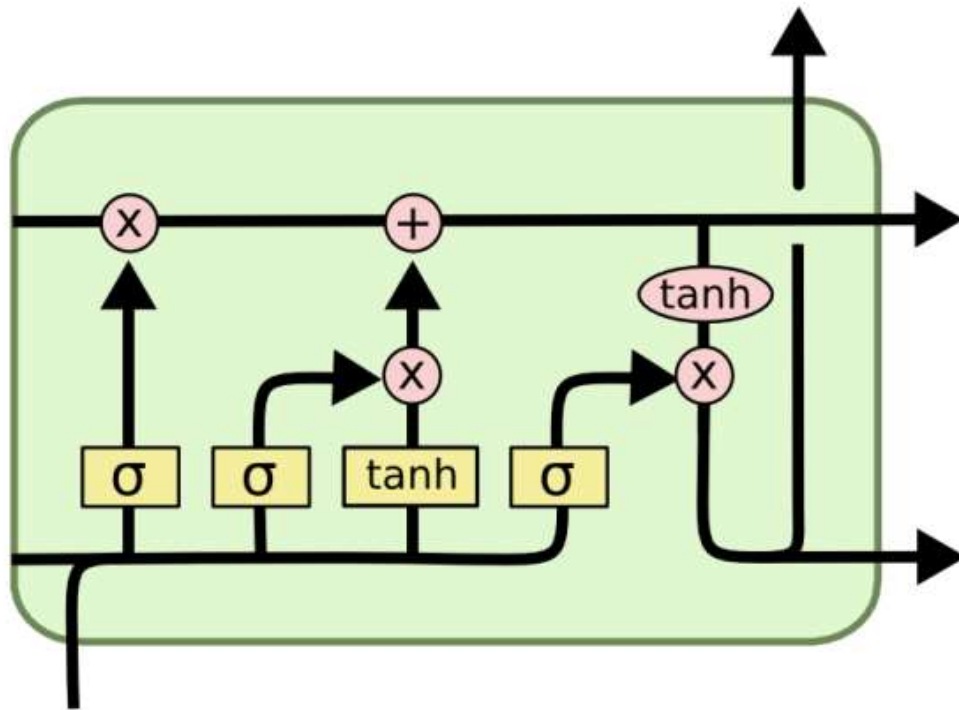
Tiger



Cat Instead of Tiger due to Vanishing Gradient

# Long-Short Term Memory ( LSTM )

# Long-Short Term Memory



## LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

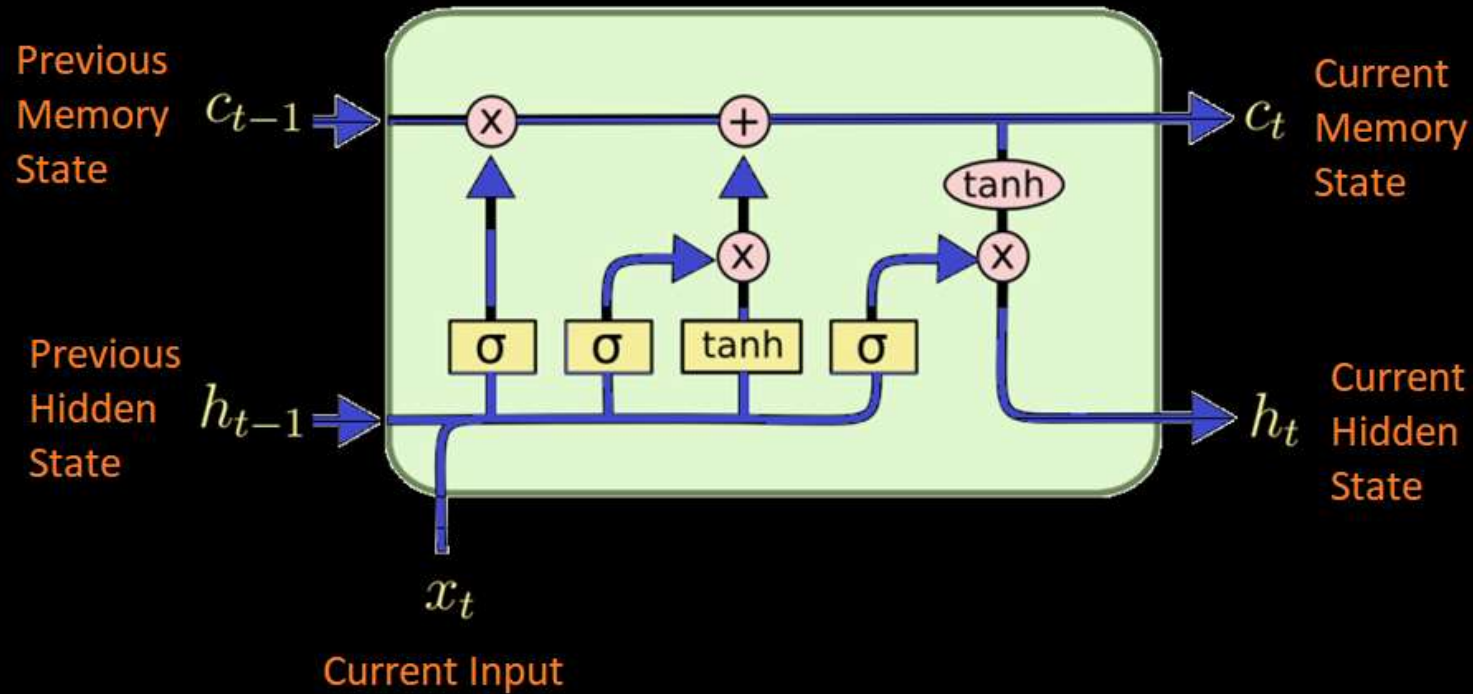
Sepp Hochreiter  
Fakultät für Informatik  
Technische Universität München  
80290 München, Germany  
hochreit@informatik.tu-muenchen.de  
<http://www7.informatik.tu-muenchen.de/~hochreit>

Jürgen Schmidhuber  
IDSIA  
Corso Elvezia 36  
6900 Lugano, Switzerland  
juergen@idsia.ch  
<http://www.idsia.ch/~juergen>

## Difference between RNN and LSTM

- The vanishing gradient problem is solved.
- LSTM has additional state called memory state which is updated at every time step.
- RNN cannot model long term dependencies but LSTM does.

# LSTM Structure for one Time Step



Input to the LSTM is the concatenation of current input  $x_t$  and the previous hidden state  $h_{t-1}$



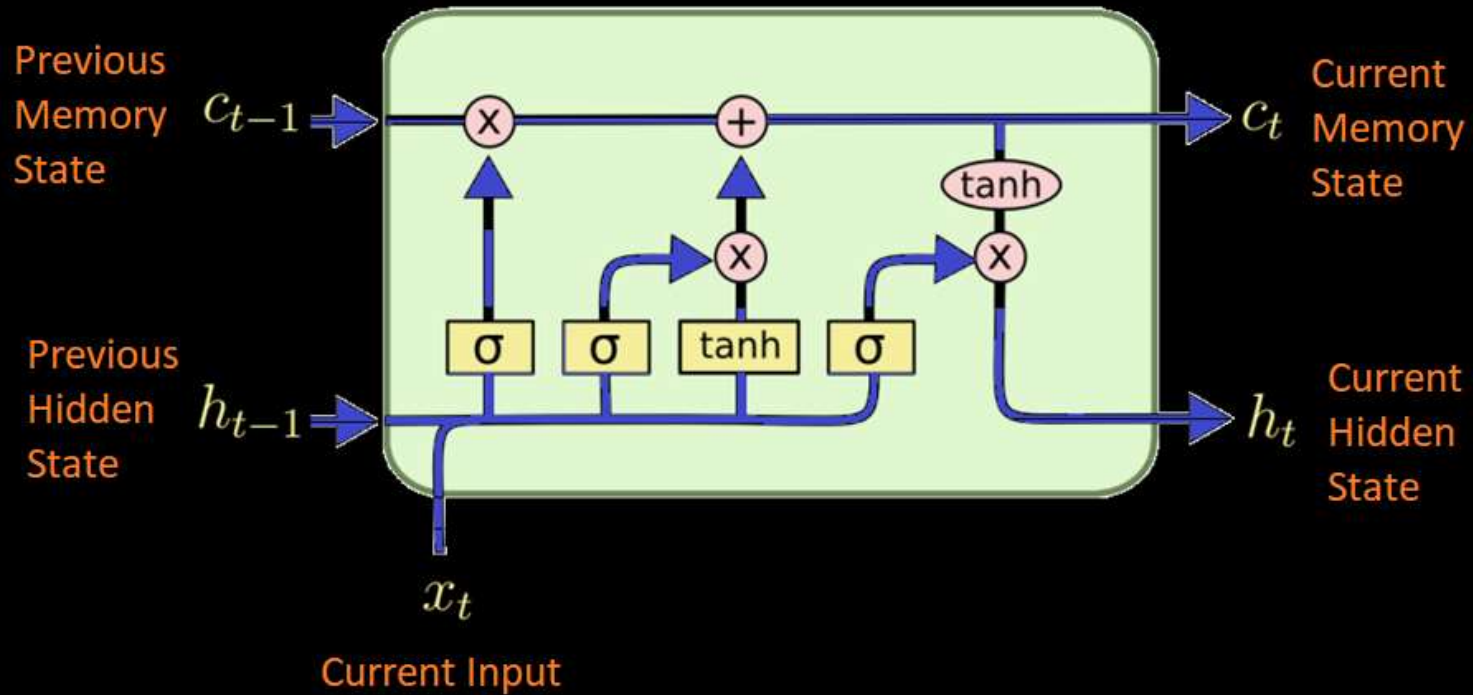
## Concatenation of $x_t$ and $h_{t-1}$

$$x_t = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad h_{t-1} = \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix}$$

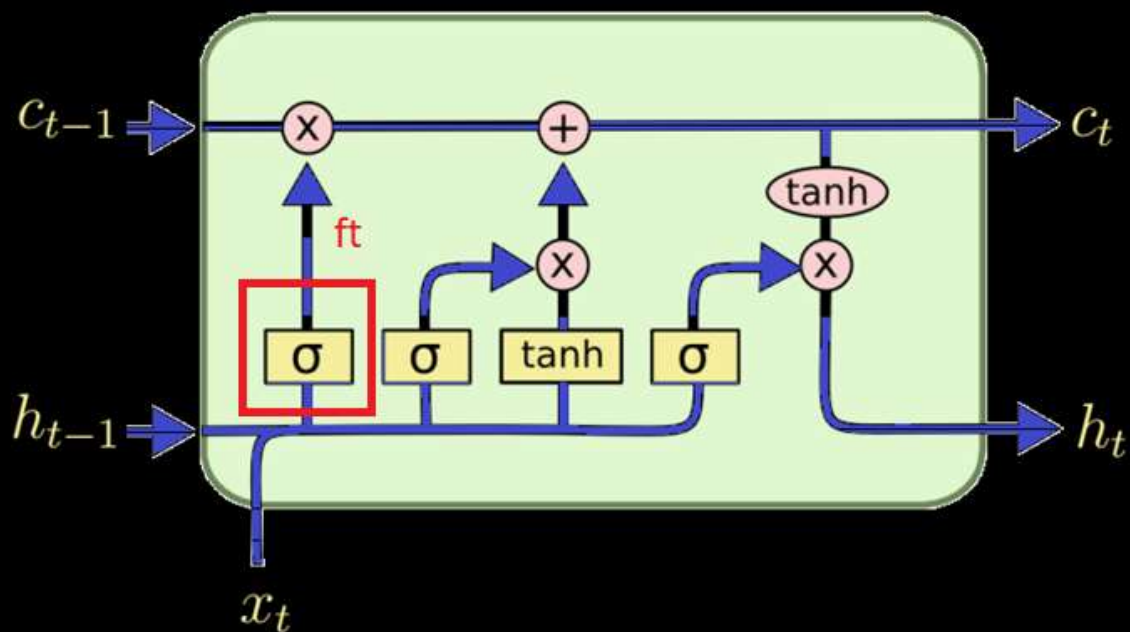
$$[x_t, h_{t-1}] = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 7 \\ 8 \\ 9 \end{bmatrix}$$

# Working of LSTM

# LSTM Structure for one Time Step



# Forget Gate



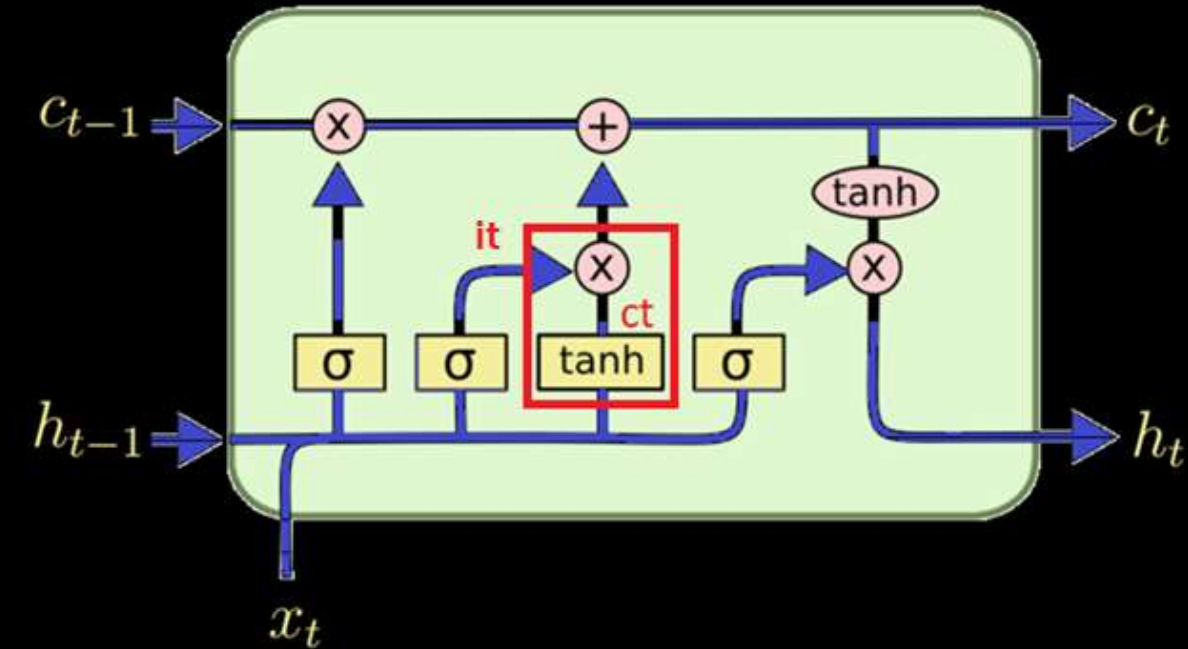
$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$f_t$  multiplies with  $c_{t-1}$  to decide what to keep

The forget gate tells LSTM how much to keep from the previous memory.

- $f_t = 0$  means forget all previous memory
- $f_t = 1$  means keep all previous memory
- $f_t = 0.5$  means keep some of the previous memory

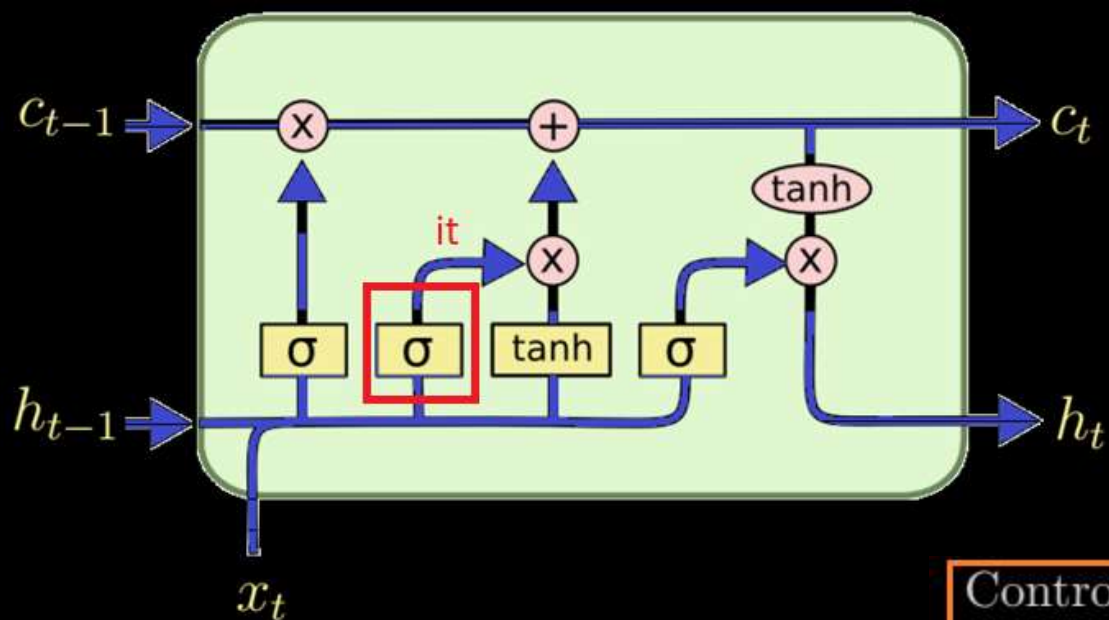
# Input Gate



$$c_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

- Input gate is used to generate new memory.
- $\tanh$  is from  $-1$  to  $1$ , therefore we can increase or decrease the memory.
- $c_t$  is the new memory that will be controlled based on the control gate output  $i_t$ .

# Control Gate

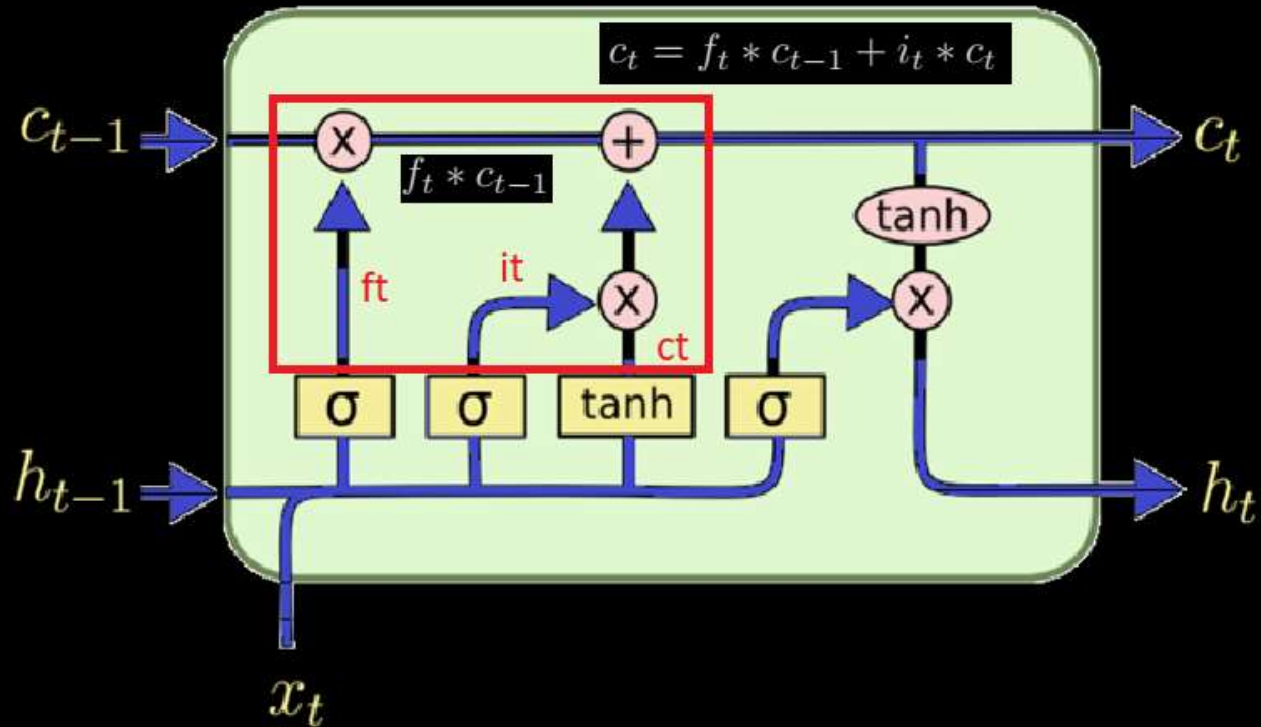


Control gate controls how much to take from the new memory.  
Control gate is the activation of input gate

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

- $i_t = 0$  means don't take anything from the generated memory
- $i_t = 1$  means take everything of the generated memory
- $i_t = 0.5$  means take something from the generated memory

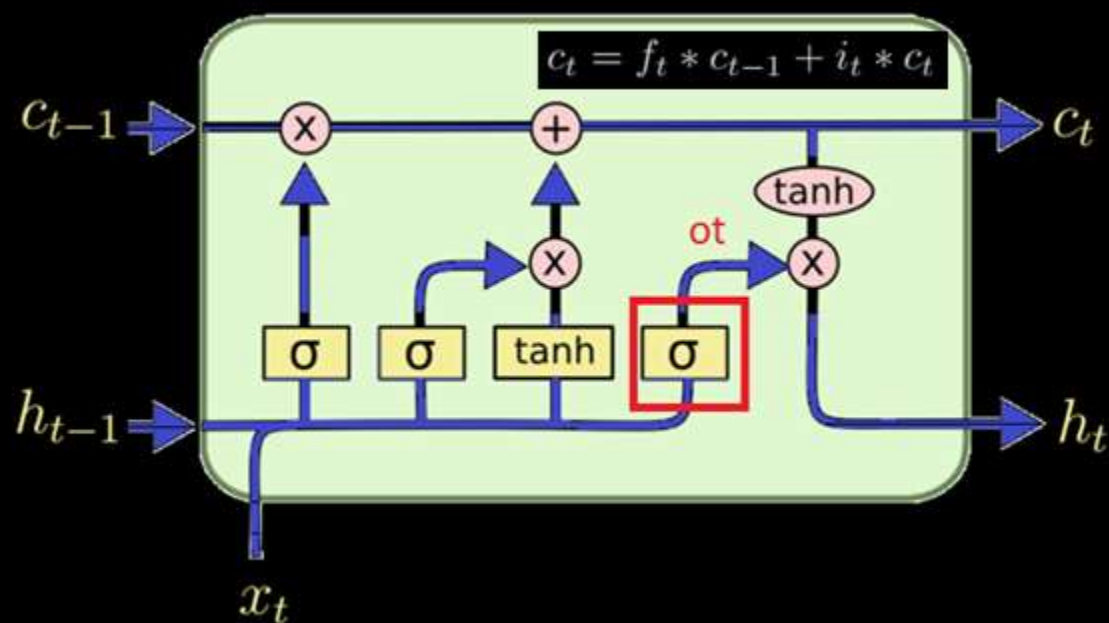
## Constructing New Memory



$$c_t = f_t * c_{t-1} + i_t * c_t$$

Add what we need from the new memory

# Output Gate



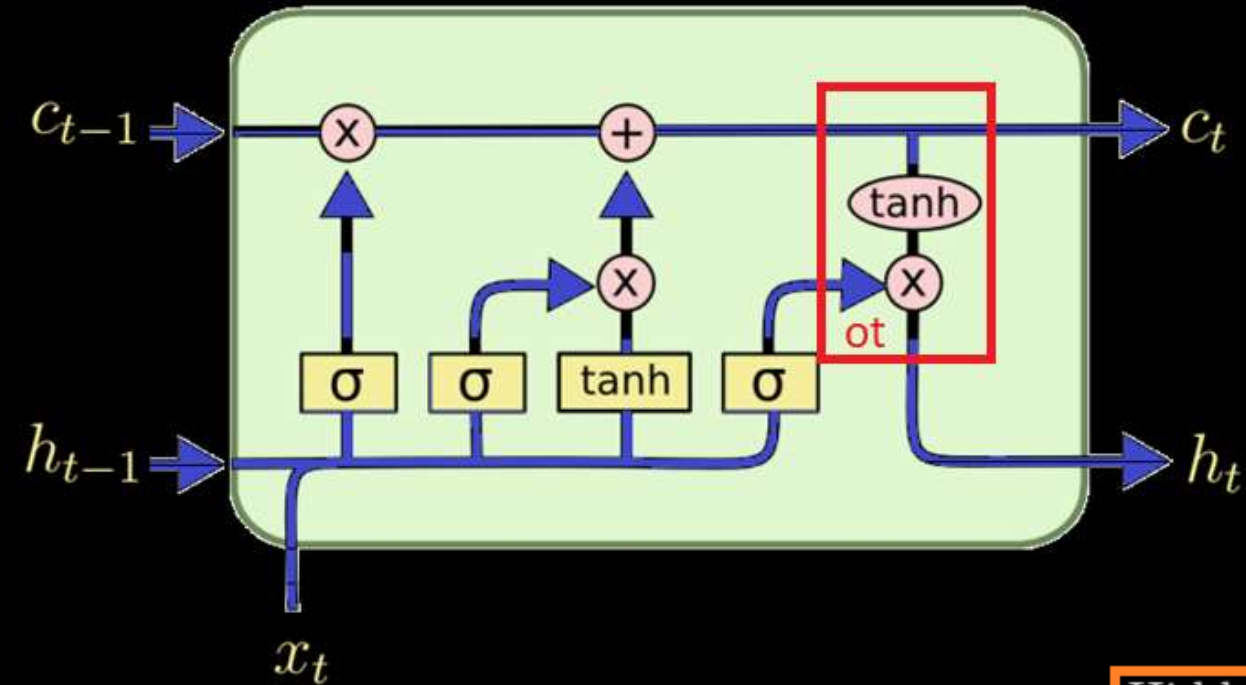
Output gate controls how much to output from the newly constructed memory.

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

- $o_t = 0$  means dont output anything from the new memory
- $o_t = 1$  means output everything of the new memory
- $o_t = 0.5$  means output something from the new memory

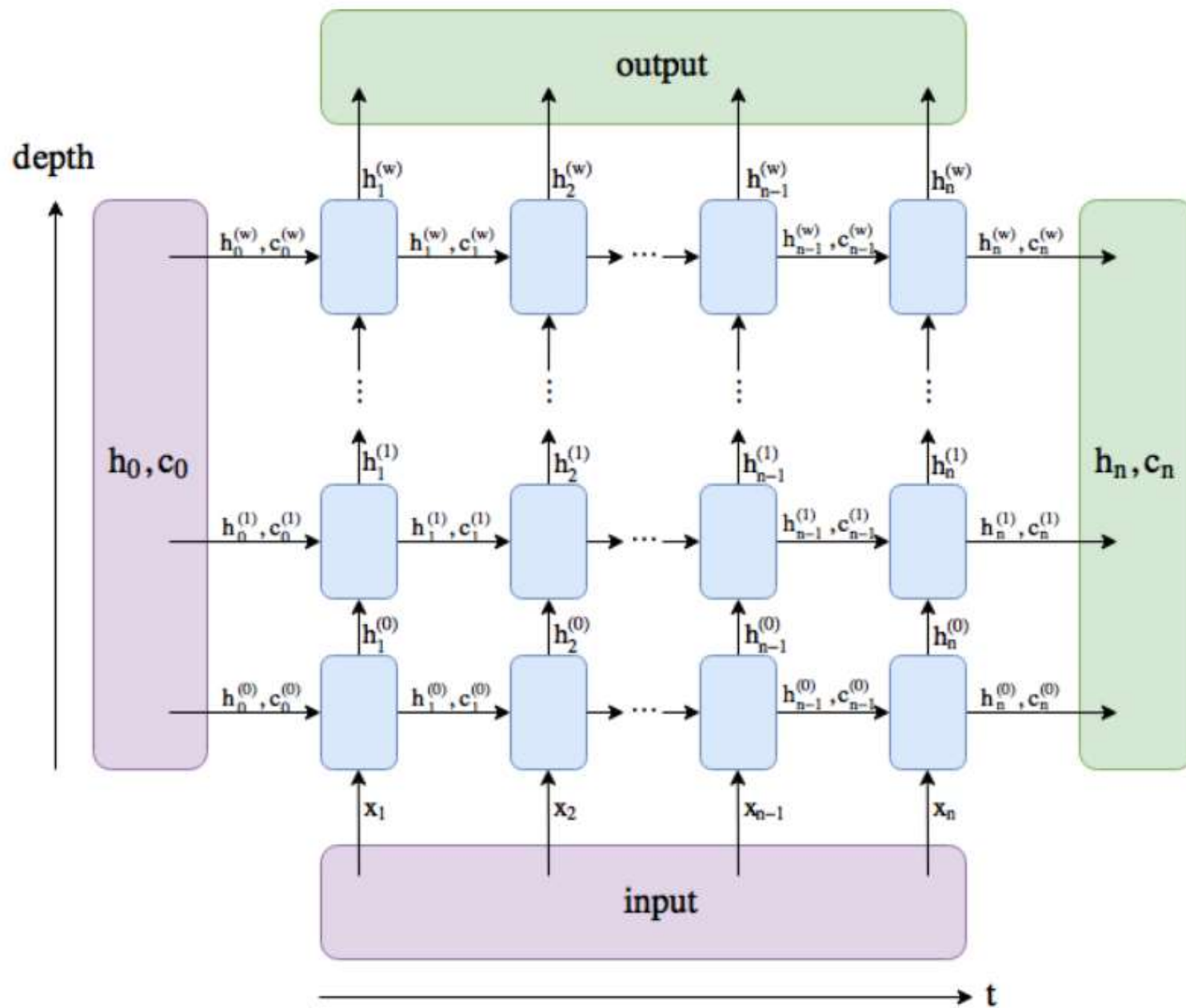


# Hidden State Output



$$h_t = o_t * \tanh(c_t)$$

Hidden state output is a controlled activation of the memory state



Concept of Batch Size, Sequence Length  
and Feature dimension  
for  
Time Series Data

# Data Dimensions for LSTM

data = (Batchsize, Sequence length, Feature dimension)

data = (N, T, D)

where,

N = Batchsize

T = Sequence Length

D = Feature Dimension OR Input size.

# Tabular Data

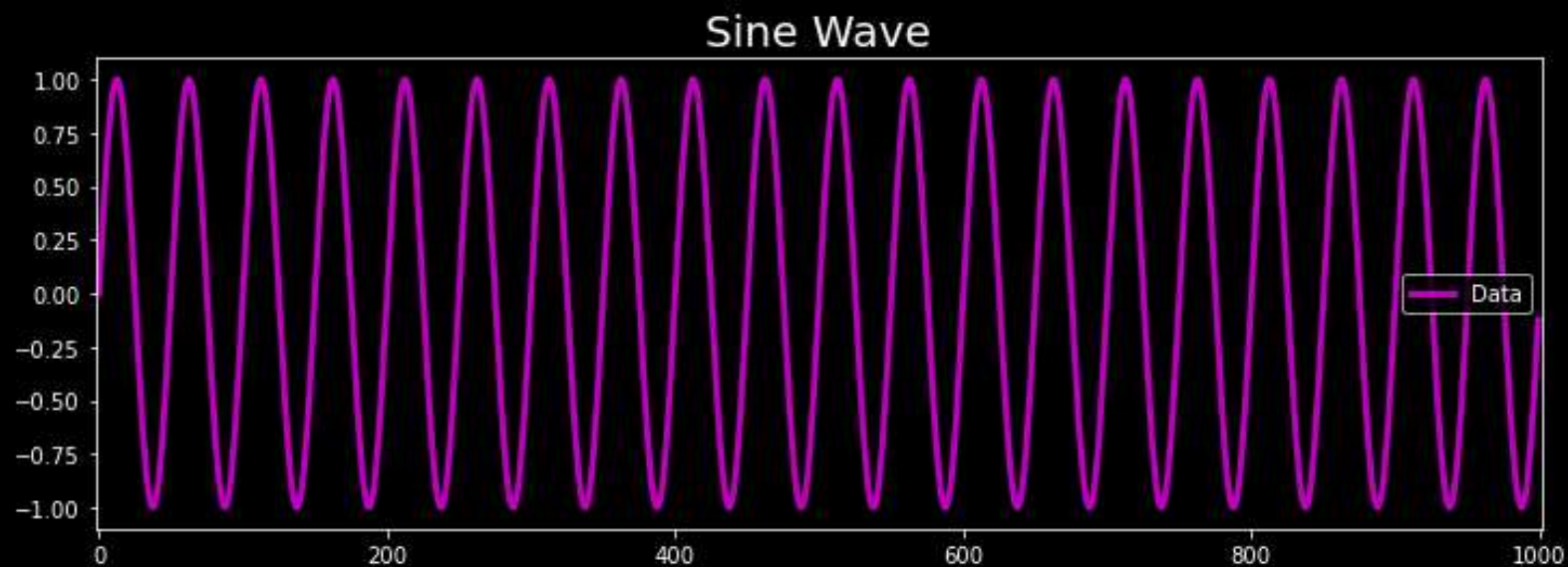
sepal_length	sepal_width	petal_length	petal_width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5.0	3.4	1.5	0.2
4.4	2.9	1.4	0.2
4.9	3.1	1.5	0.1

What is N, T and D?

$$N = 10$$

$$D = 4$$

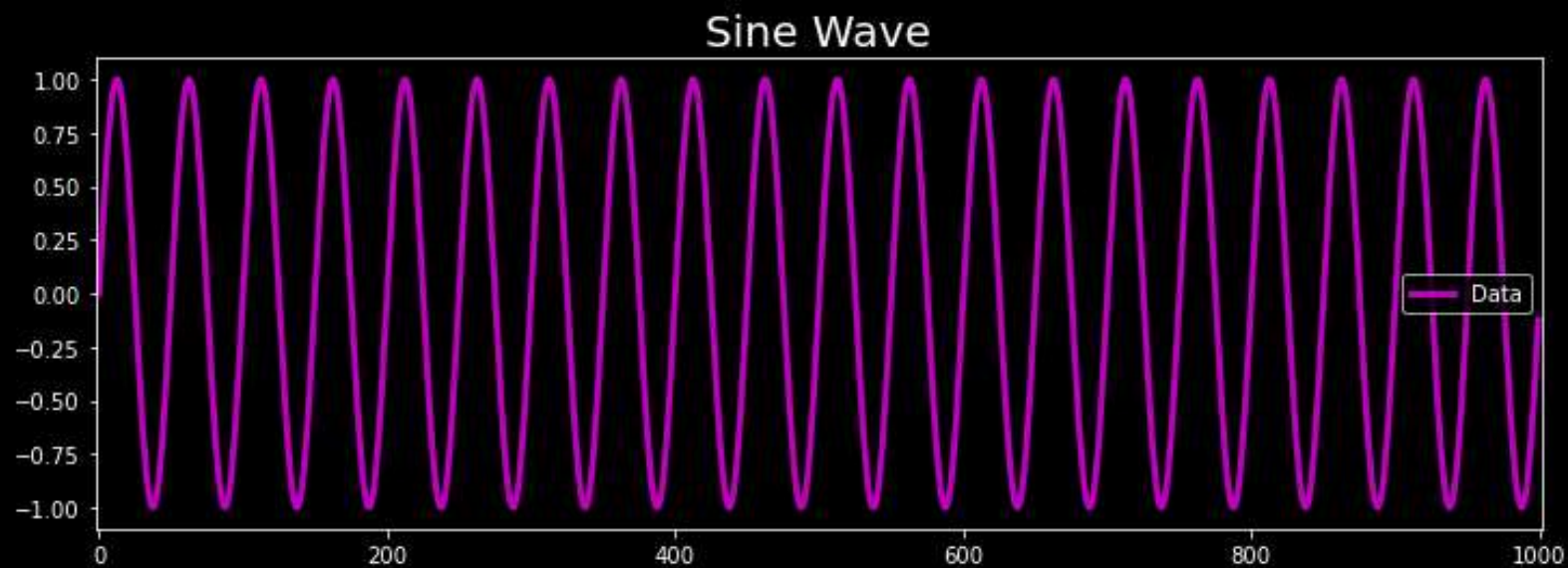
# Time Series Data



`len(data) = 1000`

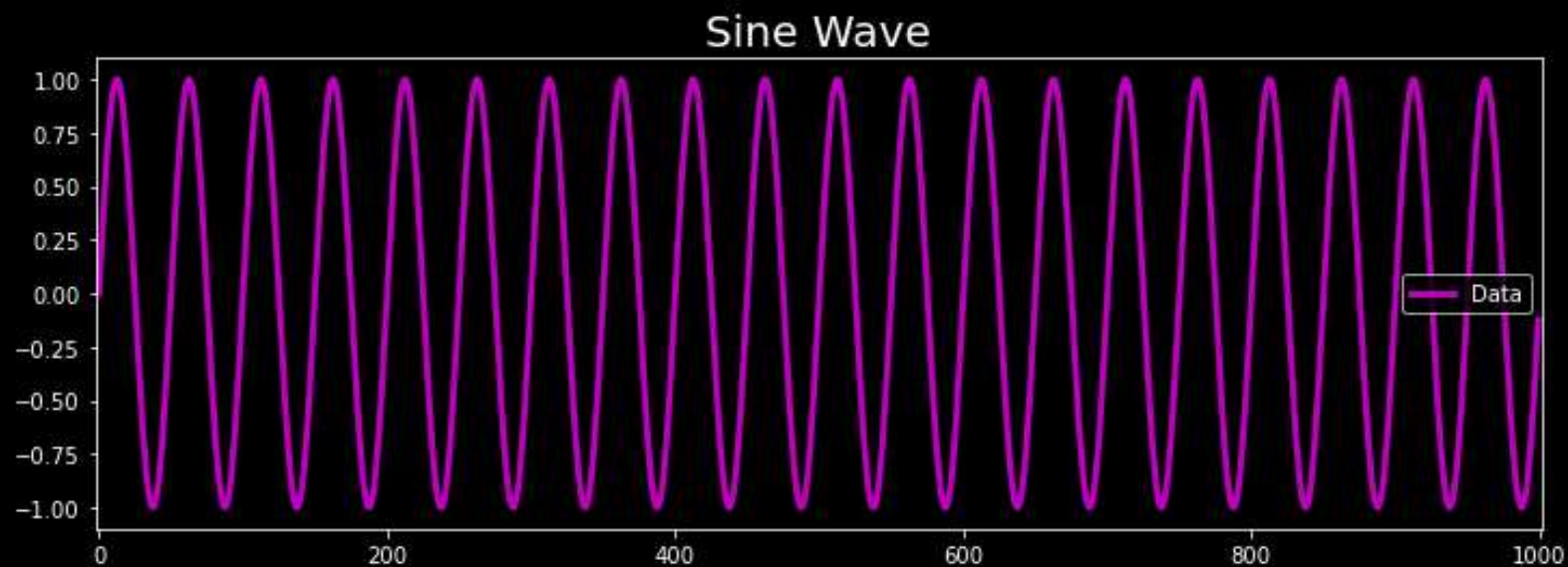
Is 1000 N, T or D?

# Time Series Data



$T = 1000 = \text{Sequence length}$

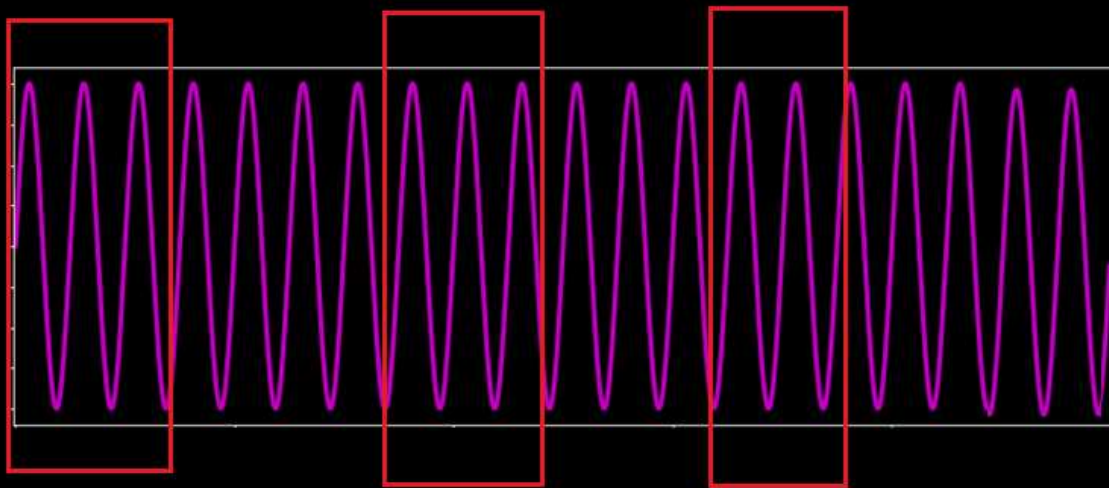
# Time Series Data



$T = 1000 = \text{Sequence length}$

What are N or D?





Window Size = 50

Sequence Length = 50

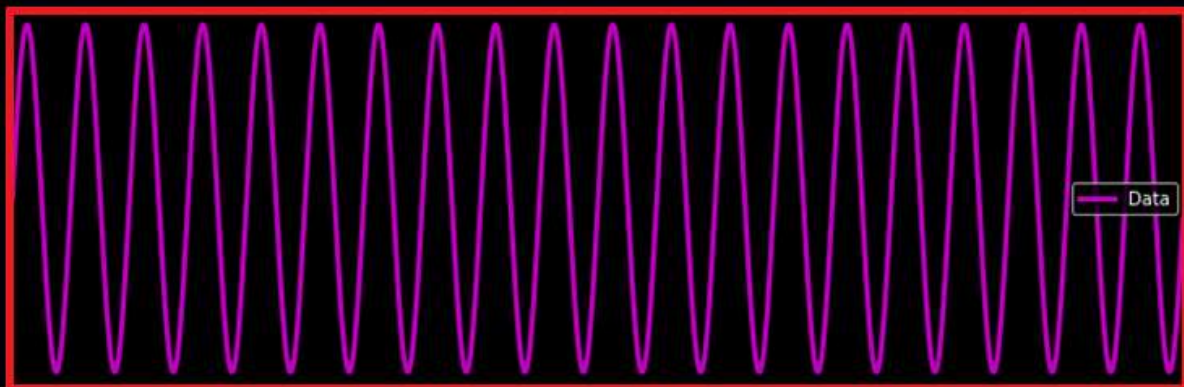
Batch size = 1 ( Training model with one window at a time)

Feature Dimension = 1

(1D data OR Single channel data OR data generated by single source)

Total samples = Number of windows of len(50)

Total samples =  $\text{len}(\text{data}) - \text{window Size} + 1 = 1000 - 50 + 1 = 951$



Window Size =  $\text{len}(\text{data}) = 1000$

Sequence Length = 1000

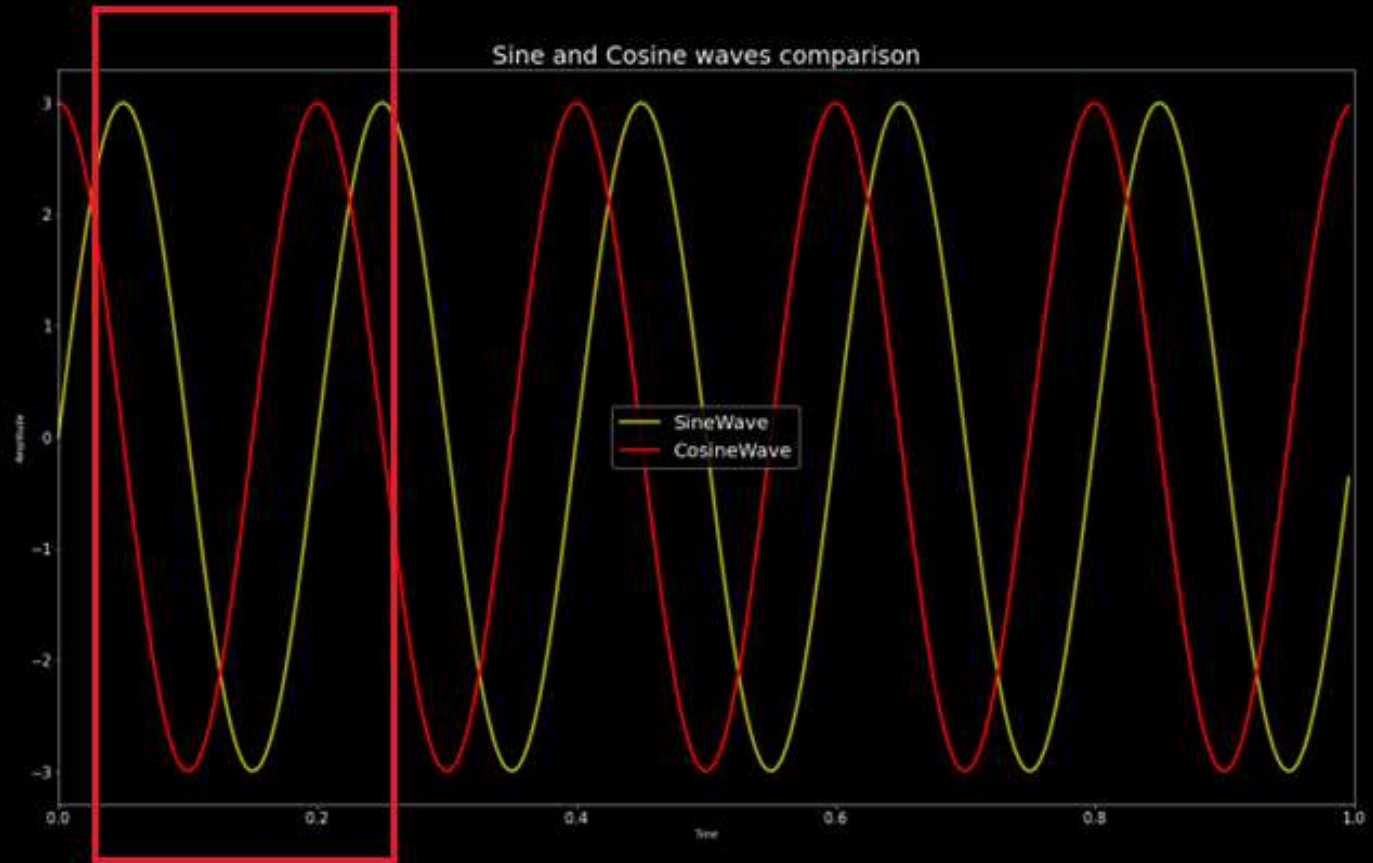
Batch size = 1 ( One Window)

Feature Dimension = 1

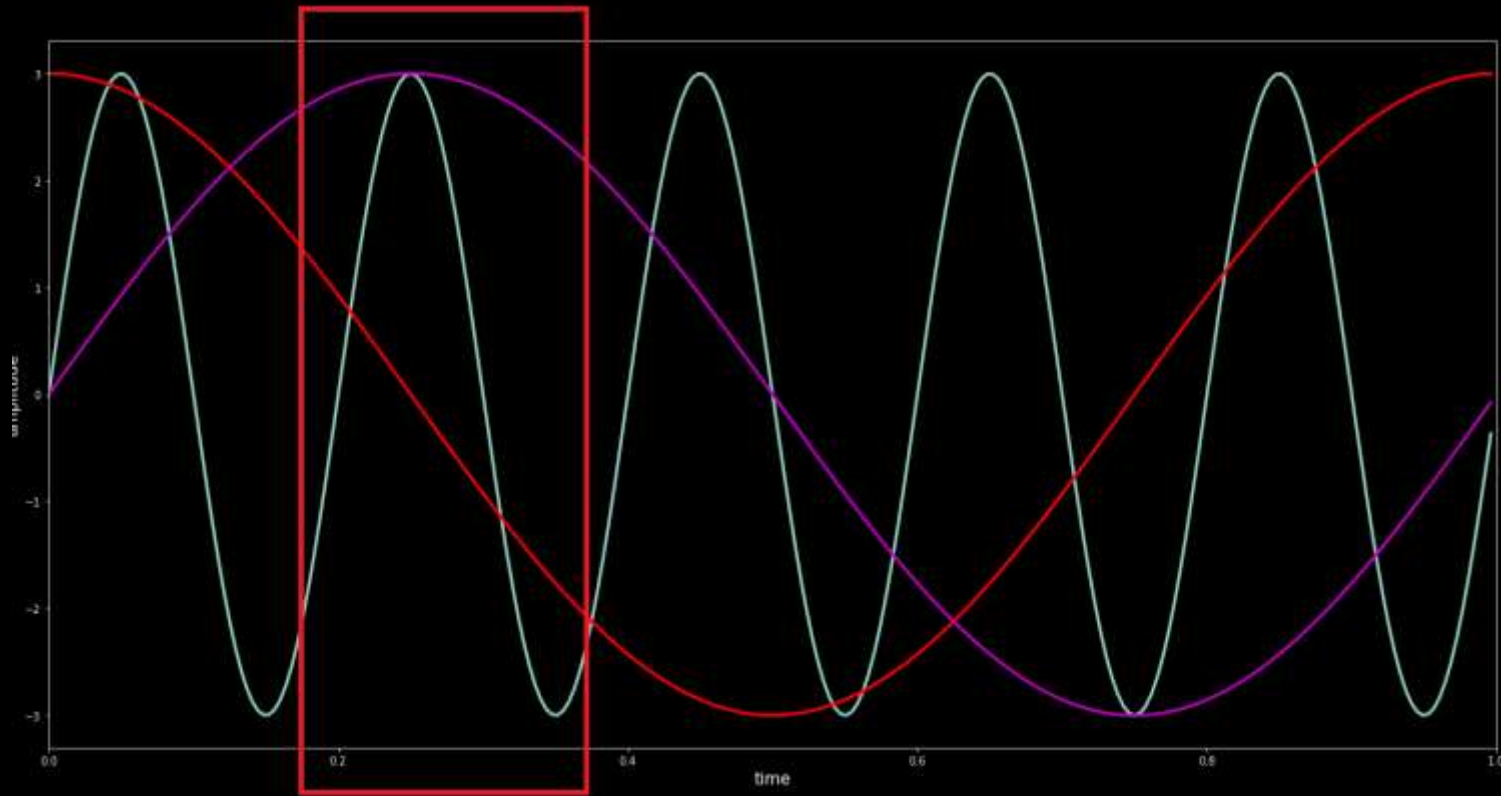
$L = \text{len}(\text{data}) - \text{window Size} + 1$

$L = 1000 - 1000 + 1 = 1$

$N = L$



**D = Feature dimension = 2**



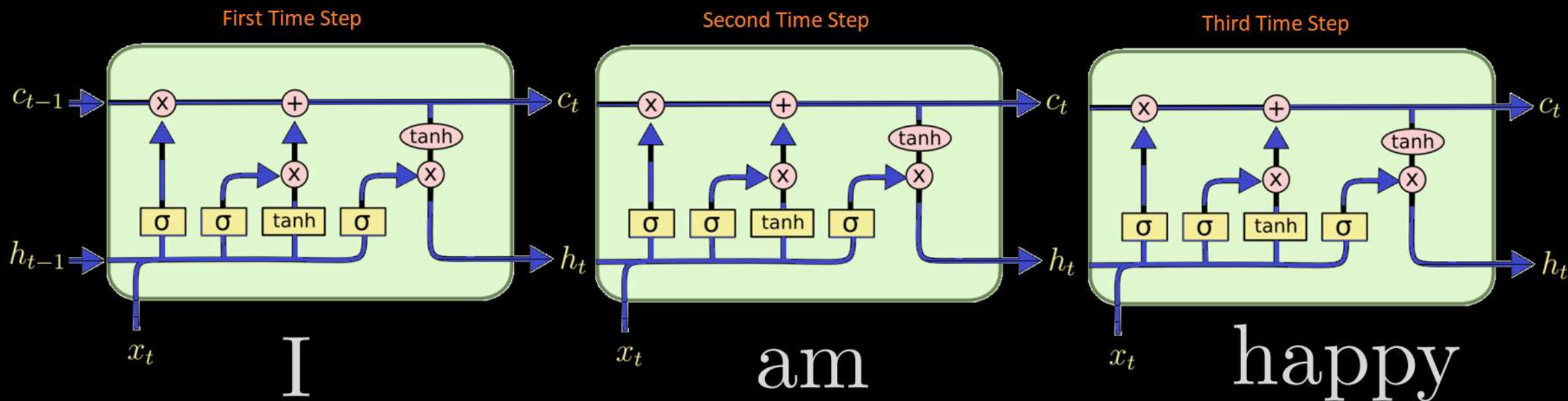
**D = Feature dimension = 3**

# Text Generation

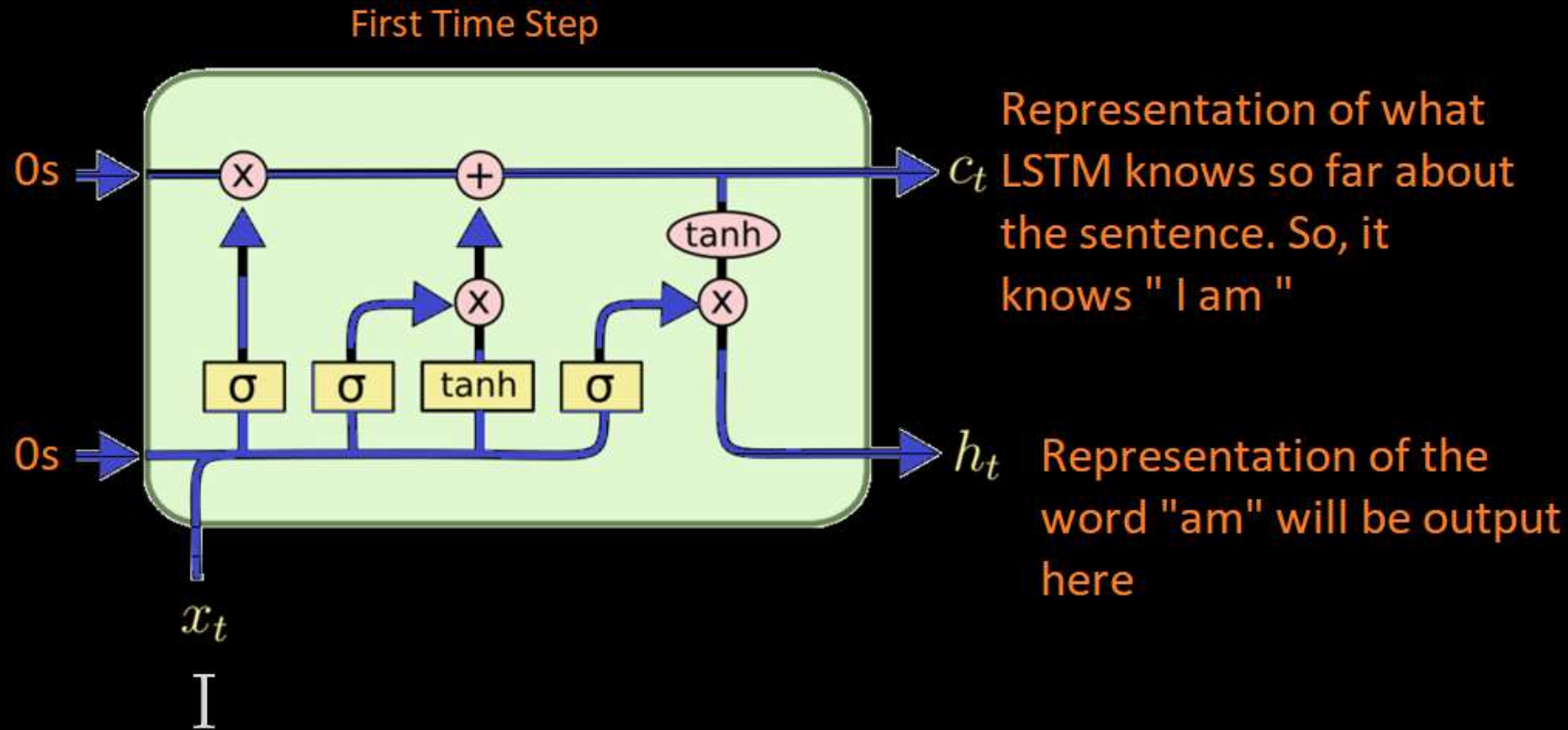
# Prediction by LSTM

I am happy

I need three time steps because sentence has three words



# First Time Step





# One Hot Encoding

- To feed text data to a neural network, we need to encode it i.e, transform the words to numbers.
- One way to implement this is to use one-hot vectors where each vector represents a specific word.



# One Hot Encoding

I am happy

$$\text{One hot vector} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

I   am   happy

One Vector for each word

## Short Comings of One Hot Vector

- It works fine if we have a small number of vocabulary
- Text with thousands of words, we would have to one hot encode all our words. Thus, it will not be computationally efficient
- The context is lost i.e correlation between the words is missing.

# Word Embeddings

- We transform words to a vector of values of real numbers. Each value of the vector represents a feature.
- Each input integer is used as the index to access a table that contains all possible vectors
- We need to specify the size of vocabulary and the size of the embeddings.

# Example of Word Embeddings

I am happy

[0, 1, 2]

I am sad

[0, 1, 3]

0	[0.5, 1.3]
1	[0.3, 1, 1]
2	[1, 2, 0.2]
3	[0.9, 1.7]

*Vocabulary Size, Embedding Size*

*Embeddings(4, 2)*

*Randomly Initialized*

0	[0.5,1.3]
1	[0.3, 1,1]
2	[1,2, 0.2]
3	[0.9, 1.7]



Embedding size represents features of each word

## Bidirectional RNN

- Understand bidirectional with Name Entity Recognition (NER)
- NER means to recognize people, company, location, date.

## Example

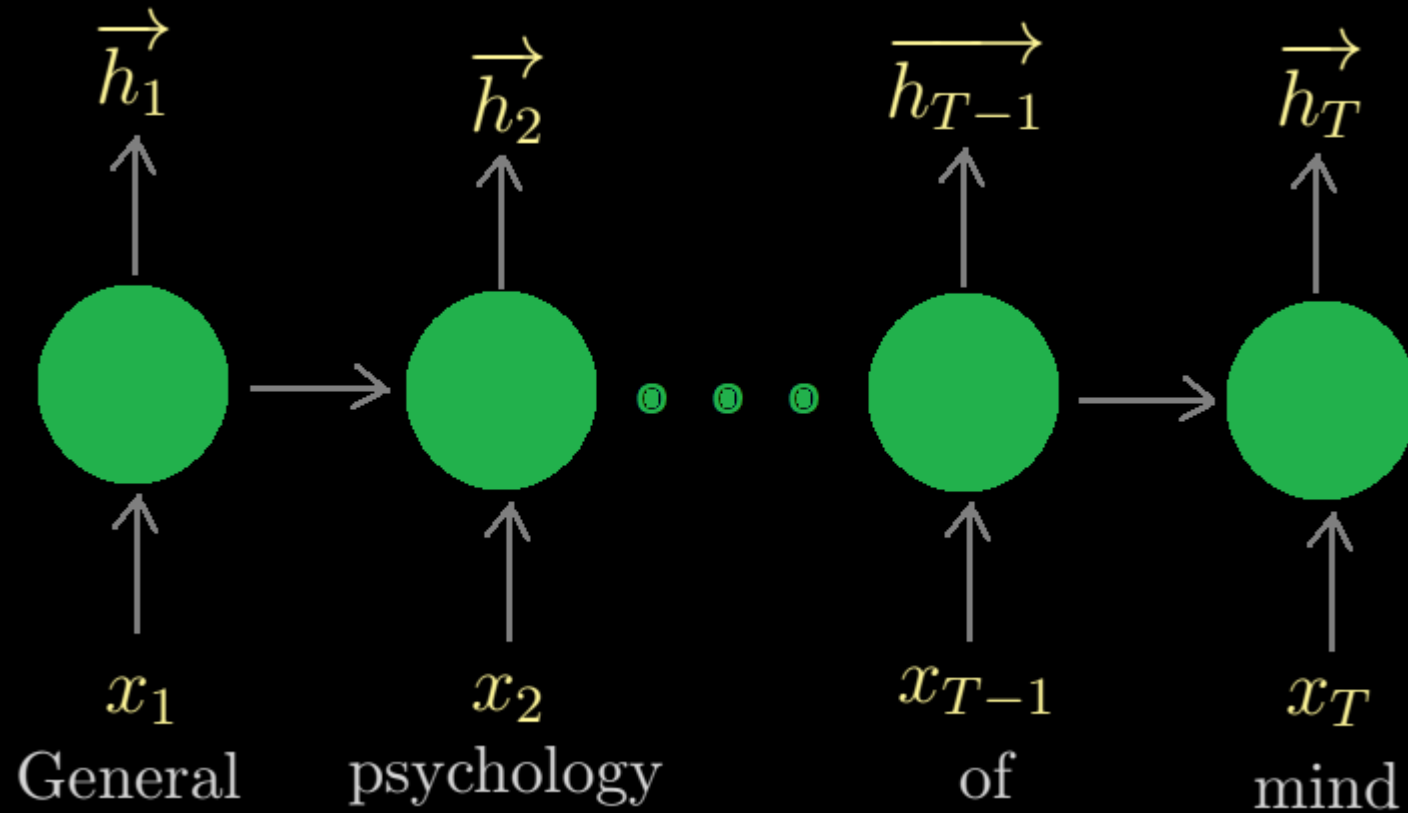
- General psychology is the study of behaviour of mind.
- General Zaid is the top most commander in the history of warfare.  
**Person**
- General are the manufacturers of Fans, AC, Refrigerators  
**Company**

## Problem

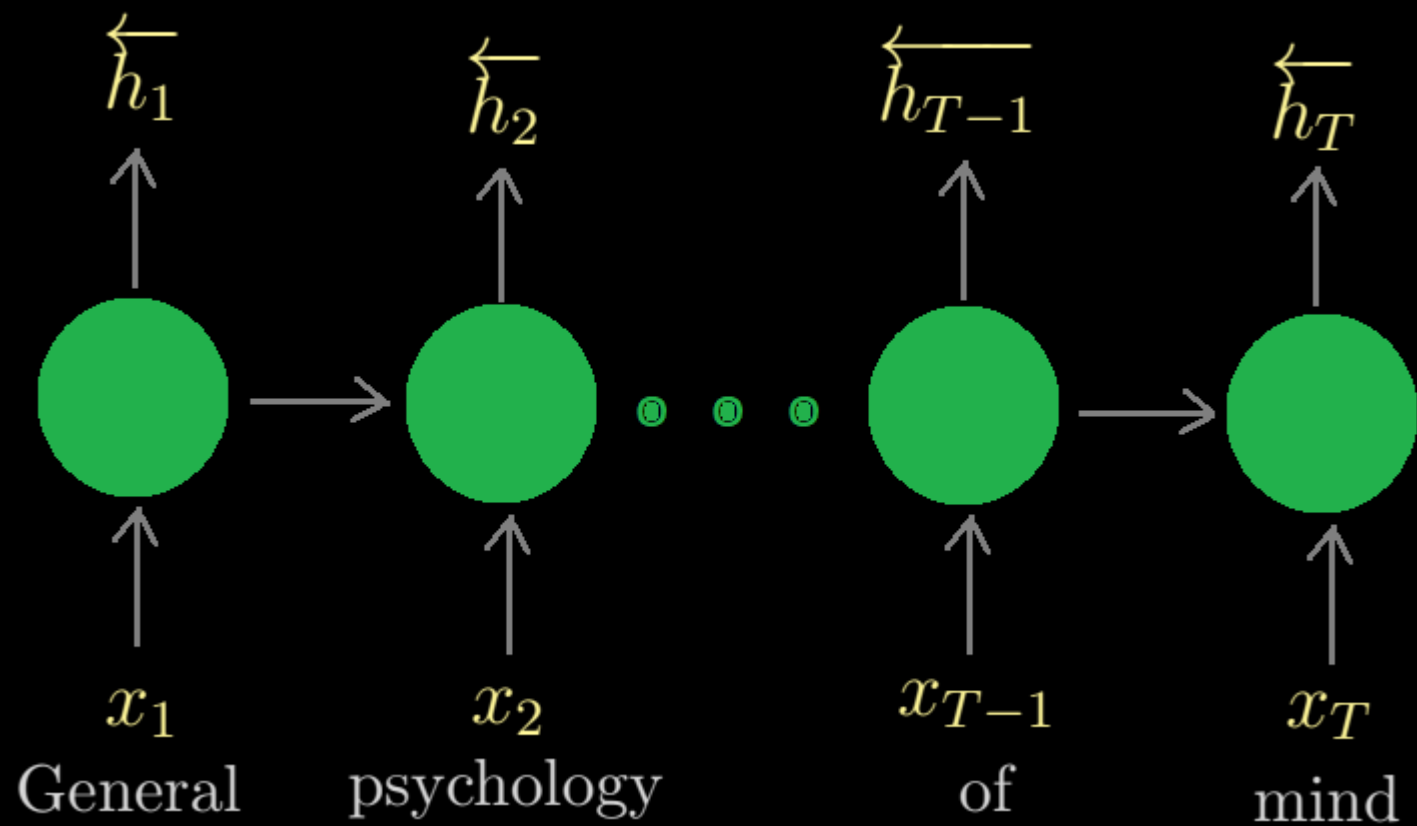
Since RNN did not see any word before **General**, therefore, it has to make prediction based on the word **General** which has the different meanings in every sentence



## Standard RNN to read the sequence in forward direction



## Second RNN to read in reverse direction



Opposite arrows show that we are looking the sequence in reverse and this solved problem about the world general.

In bidirectional RNN, we calculate hidden states and concatenate them.

$$h_t = \left[ \overrightarrow{h_t}, \overleftarrow{h_t} \right]$$

## Size of RNN Units

The hidden size of forward and backward RNN is  $T \times H$

After concatenation. the size becomes  $T \times 2H$

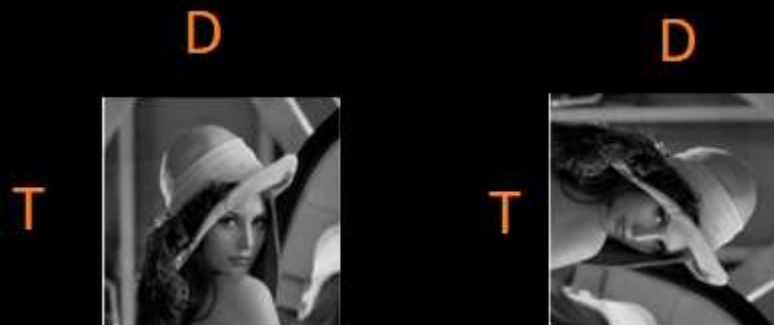
# Image Classification With Bidirectional RNN

## First Method

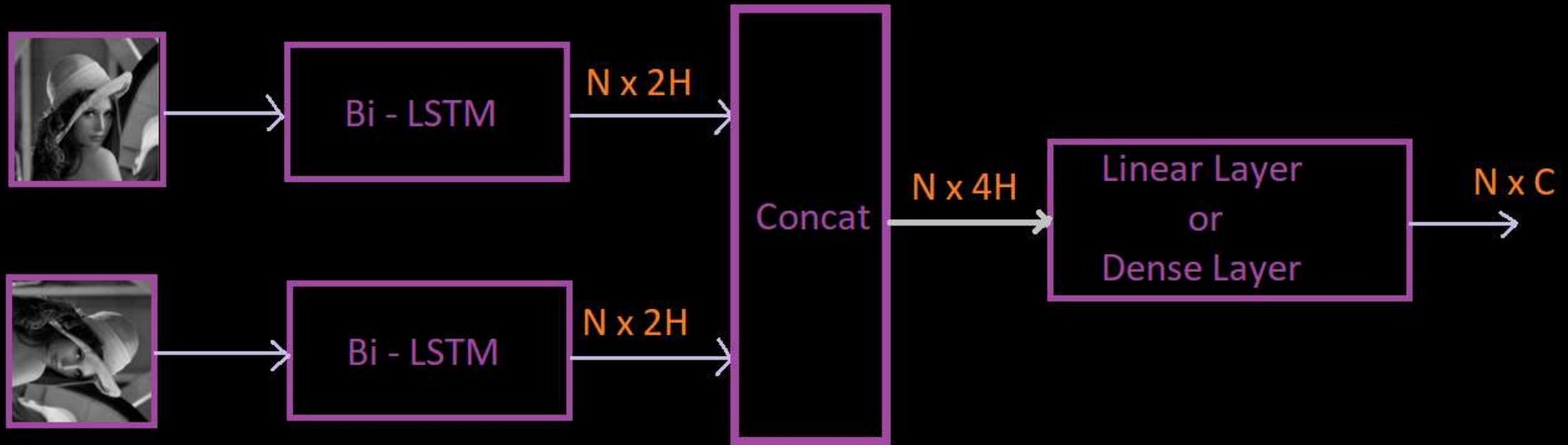
- No change in code, just a layer wrapper.
- `Bidirectional(LSTM(Hidden_Size))`

## Second Method : Dual Bidirectional LSTM

- First Bidirectional RNN moves from top to bottom and bottom to top.
- Rotate the image and run the bidirectional RNN, hence we cover all the four directions.
- We do this because image is like 2D time series of pixels and we can select any dimension as T and D.



# Implementation



Thank you !



Thank you !

