M: 4/Oct/2020 10:30 AM

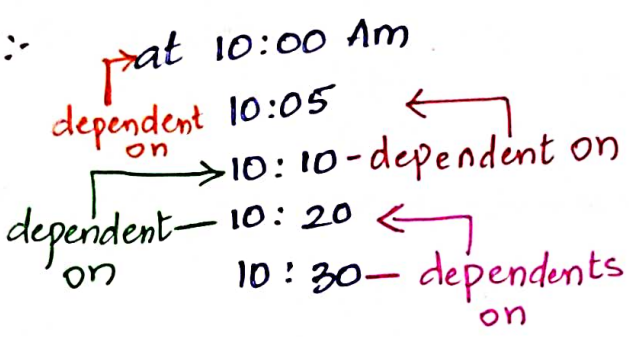
Recurrent Neural networks

↓

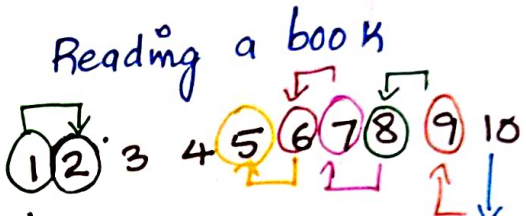
Time Series

Que:- Why can't we use a normal neural network ? for time series data ?

Ex:-



Ex:2 Reading a book



10 page doesn't dependent on 1st page.

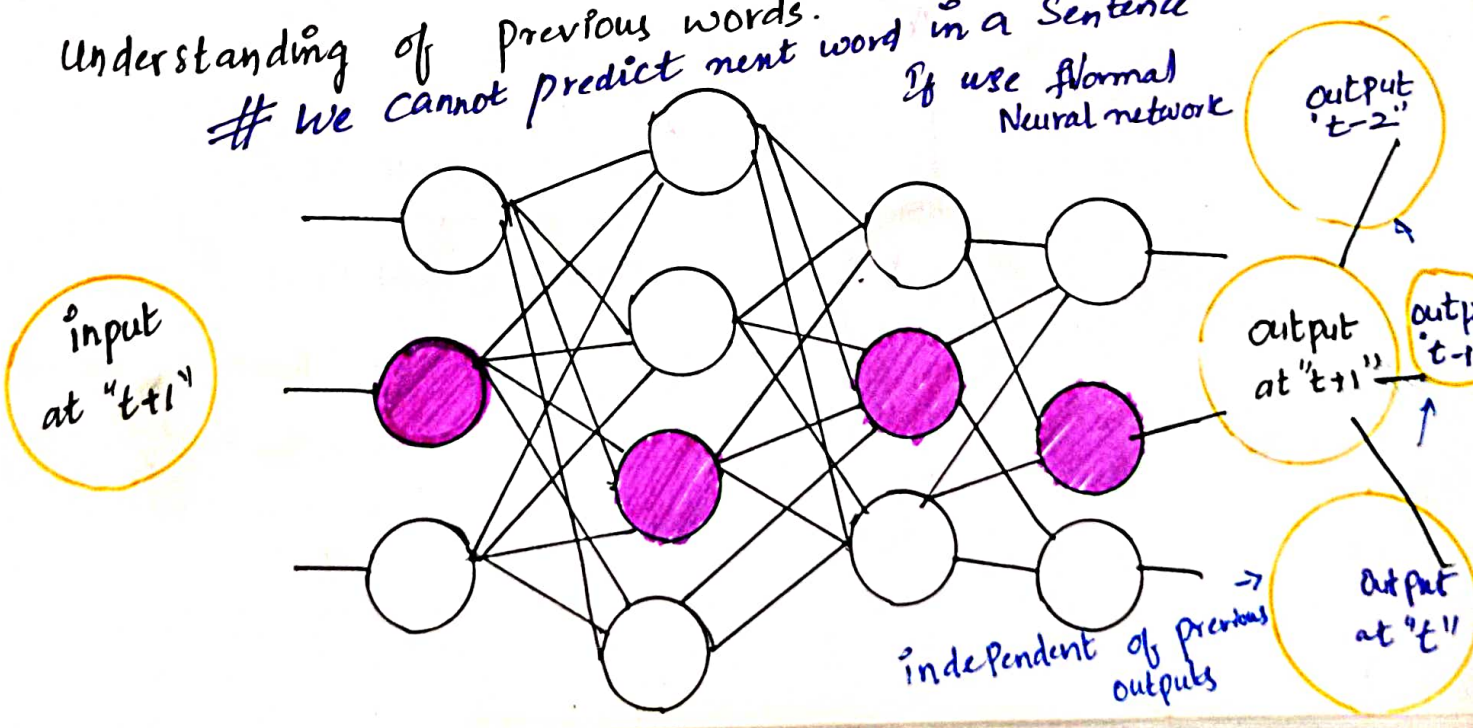
Page has some information of Page: 9

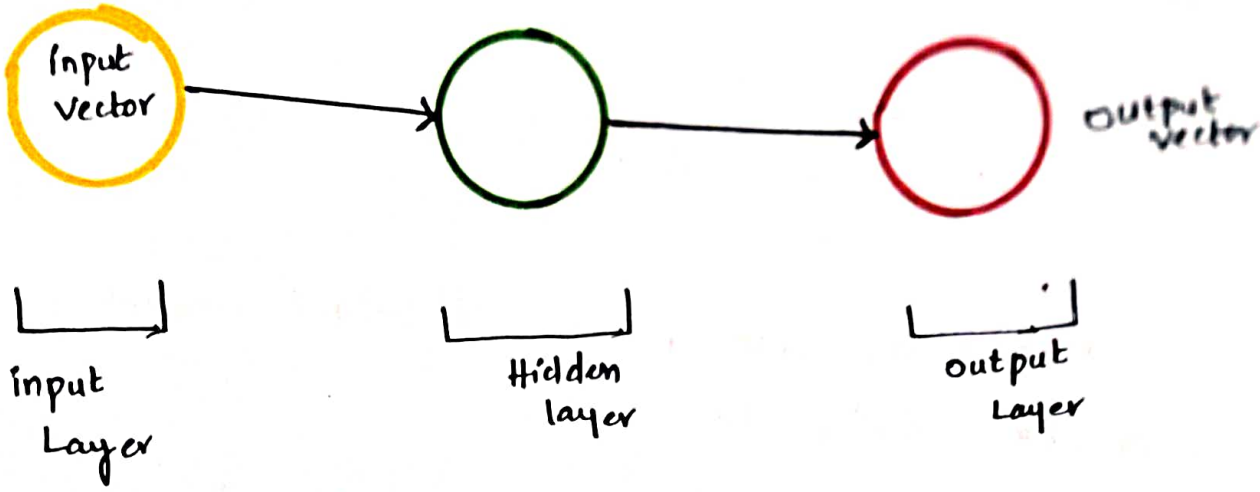
Sequential order

* When we are reading 2nd page. If we have any doubt we check 1st page not 10th page.

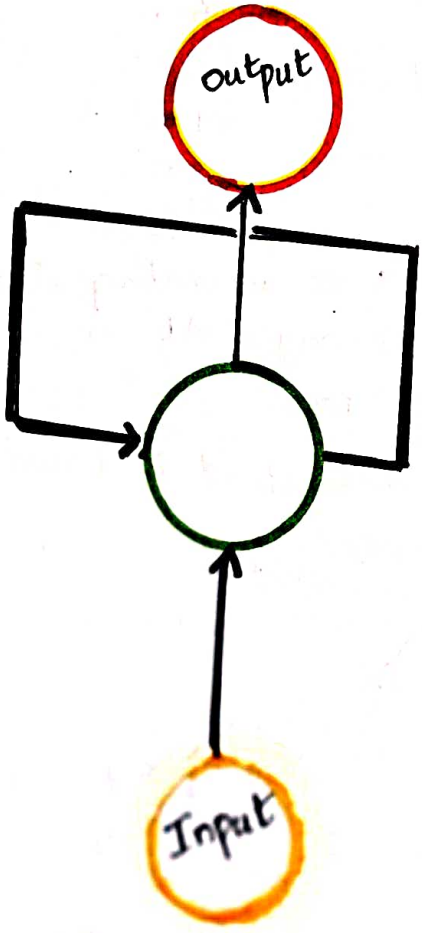
* When You read a book, You understand it based on Your understanding of previous words.

~~We cannot predict next word in a Sentence~~ If use Normal Neural network





R.N.N



connections.

∴ Works on principle of

"Recurrent Neural Network"

↓

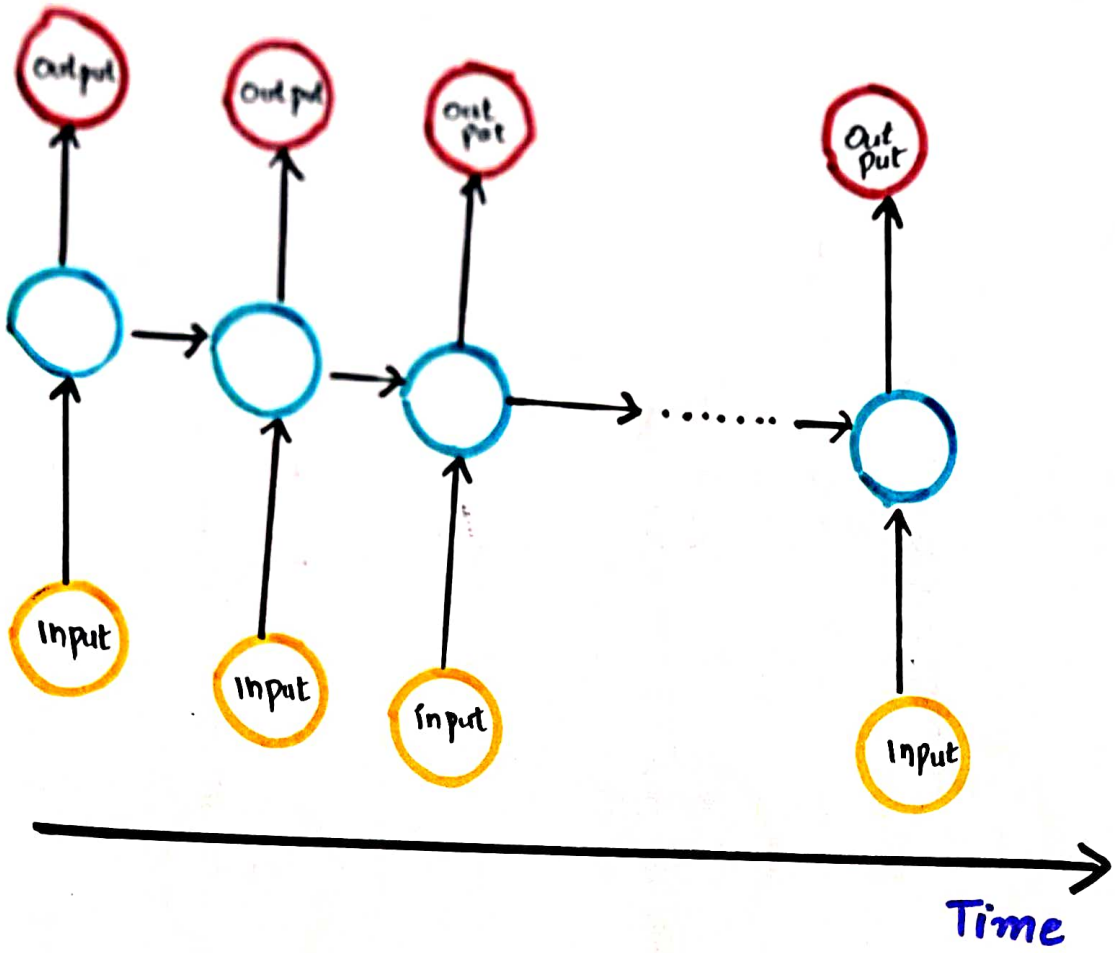
"Recursive"

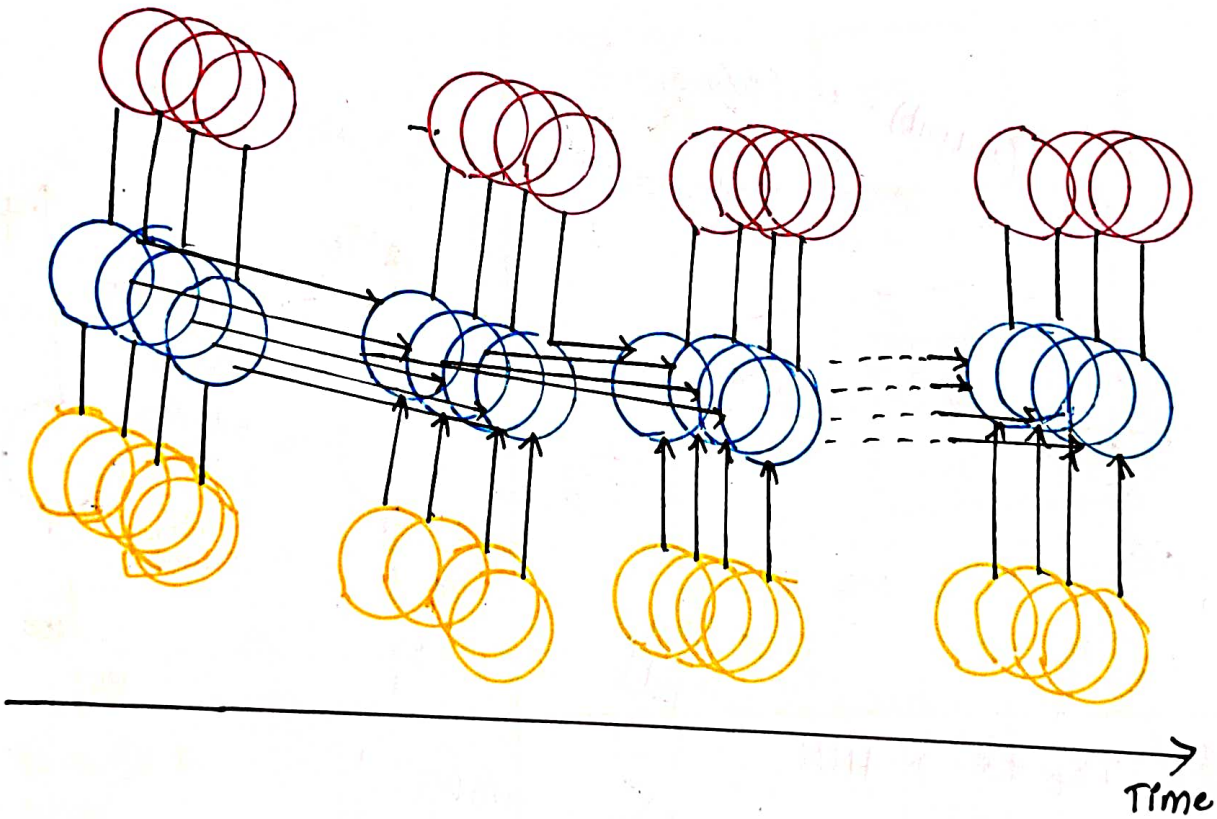
↓

factorial function

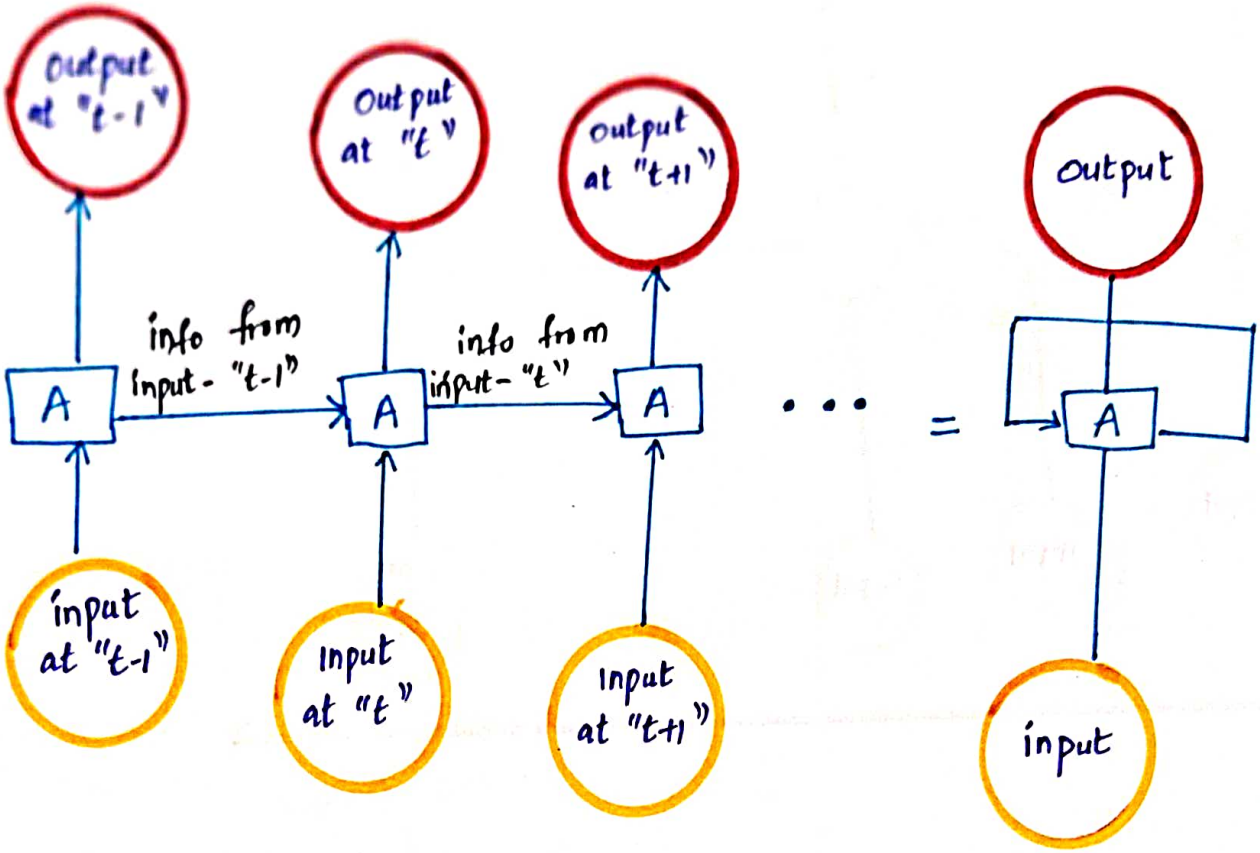
(* within the function, we use "same function")

Recurrent Neural Networks :-

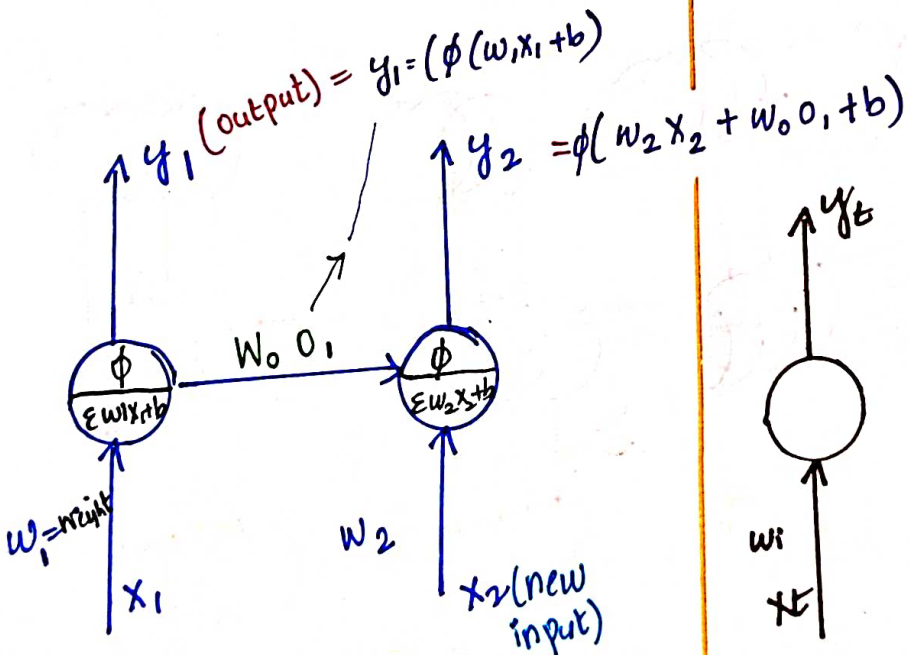


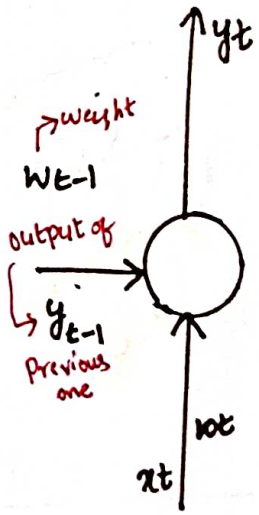


How RNN Works ?



EX:-





* $y = \phi(w_i x_t + b)$ → ANN

* $y = \phi(w_t x_t + w_{(t-1)} y_{(t-1)} + b)$

↓ RNN

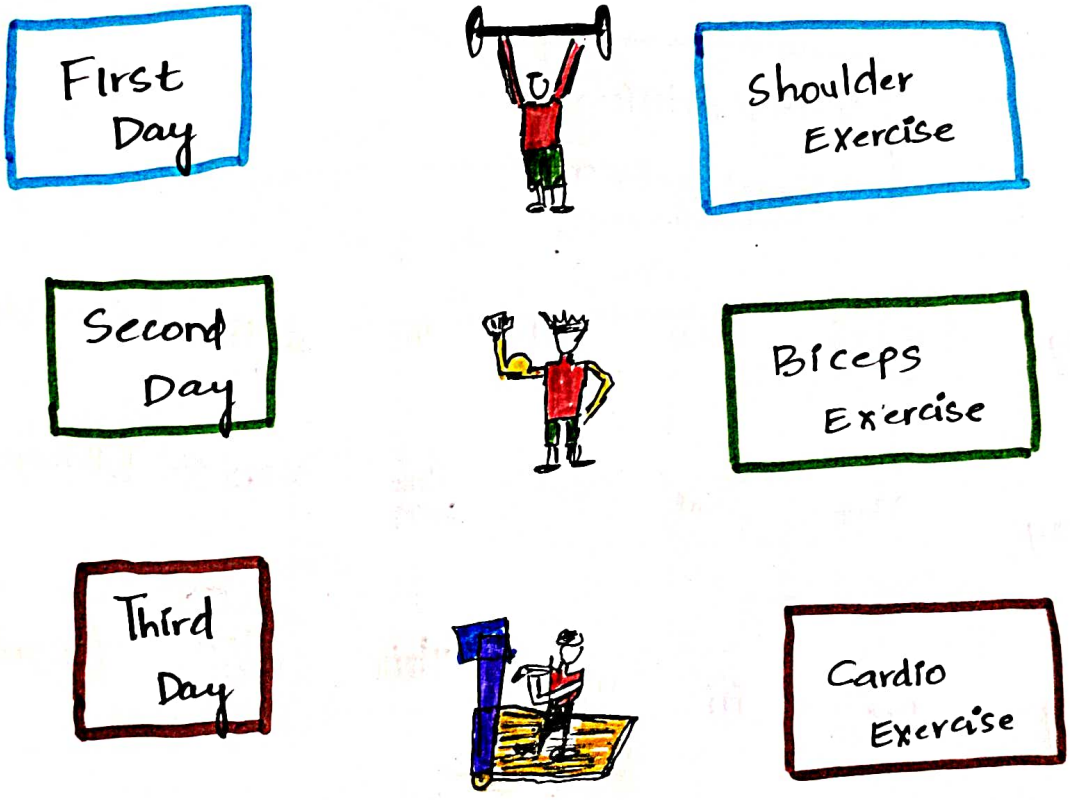
* Artificial neural network

* Recurrent neural network

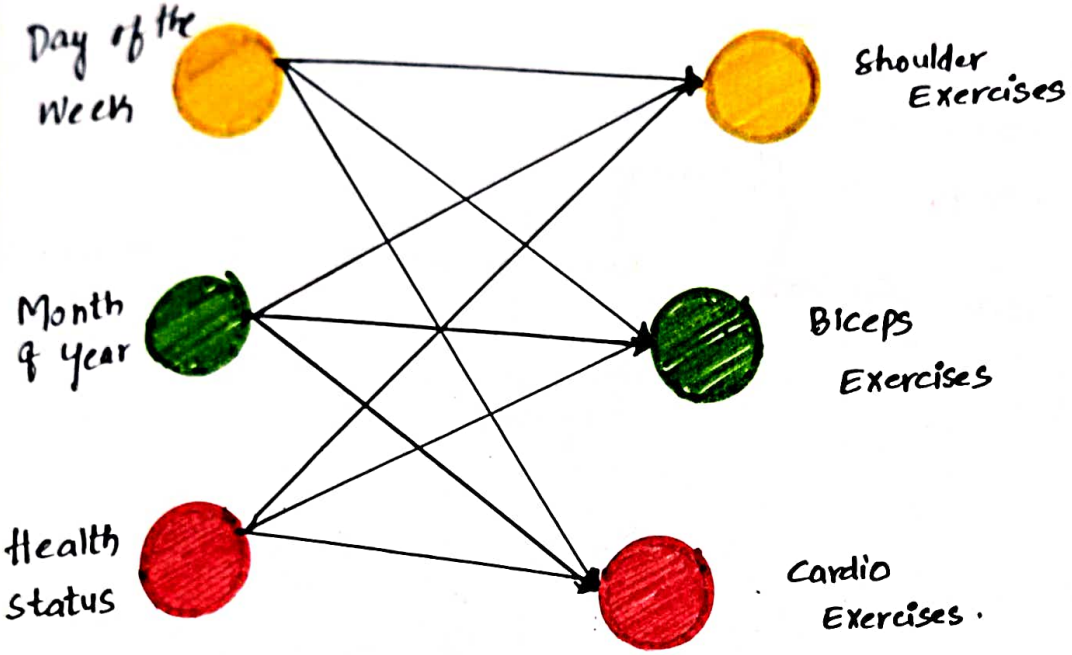
- What is RNN ?

- Recurrent Neural Network designed to recognize patterns in "Sequence of data", such as Text, genomes, handwriting the spoken word, or Numerical times series data Emanting From Sensors, stock markets and govt agen

Example :



Predicting type of Exercise



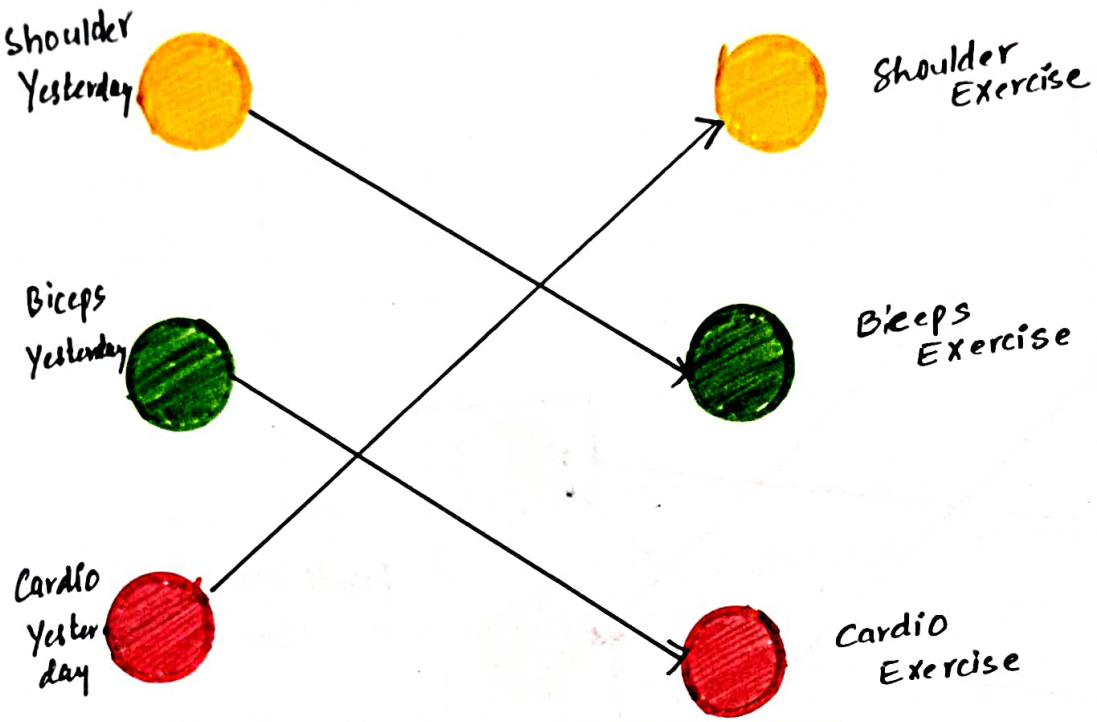
Using Artificial Neural network

First day	**Sun**	Wen	Sat	Tue	Fri	[Shoulder]
Second Day	Mon	Thur	**Sun**	wen	Sat	[Biceps]
Third Day	Tue	Fri	Mon	Thur	**Sun**	[cardio]

↓

"Artificial Neural network"

Predicting the type of exercise.



Using Recurrent Neural network

∴ Yesterday what I have done, today is dependent on Yesterday. : Sequential data.

Ex:-

Mon Boosting (Ad, G)

Tue → Leave

Wen XG [sequence]

As Per Schedule

M - Ad, G

Tue - XG

Wen - SVM

As per Schedule

- Mon : Shoulder
- Tue : Biceps
- Wen : cardio
- Thur : shoulder
- Fri : Biceps
- Sat : cardio
- Sun : Shoulder

" Recurrent Neural network"

In Gym

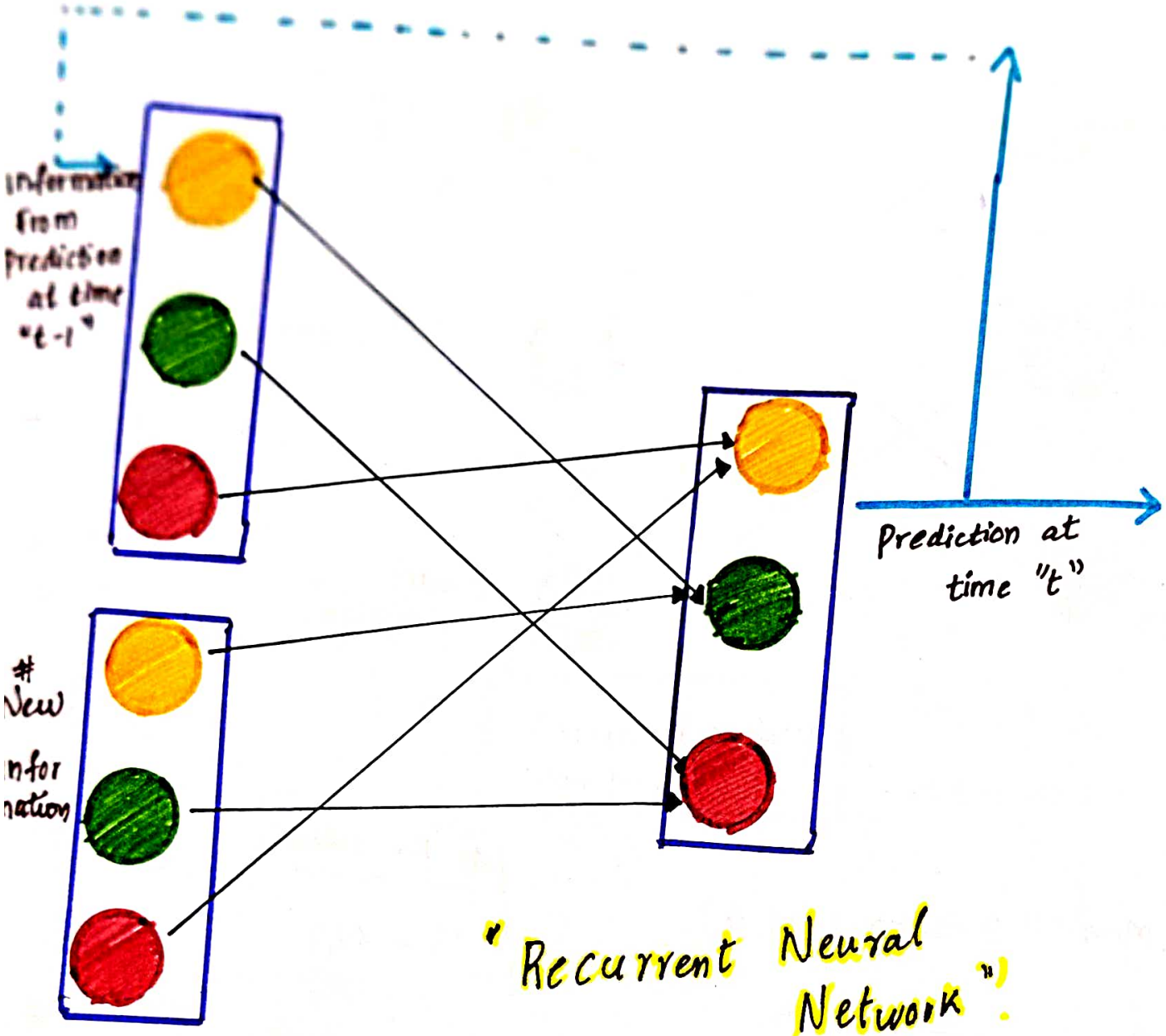
- M : shoulder
- Tue : Bieceps
- Wen : Leave
- Thur : cardio

But as per Schedule

Thursday : Shoulder

We complete pending work first.

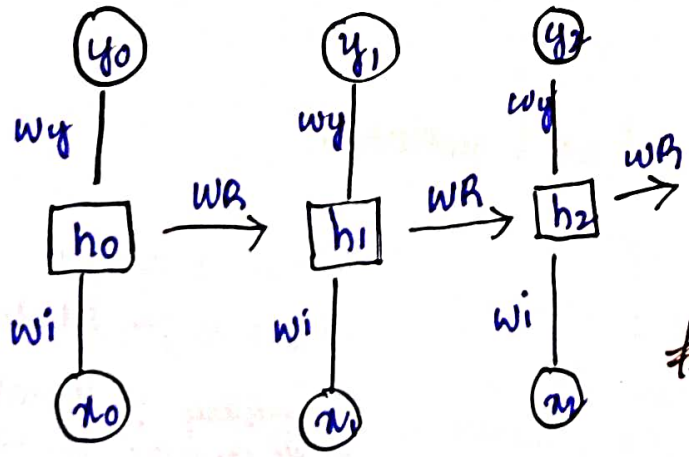
Predicting type of Example



"Recurrent Neural Network"

new information

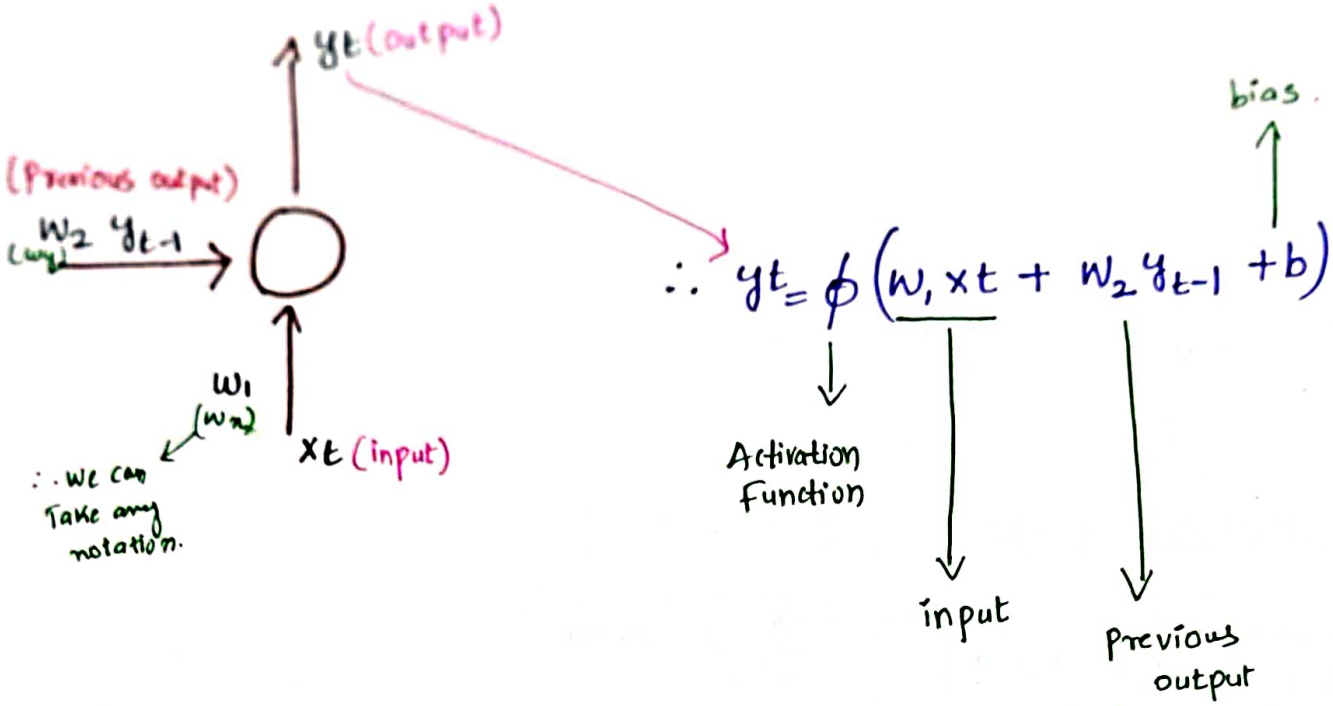
EX:-



$$h(t) = g_h\left(w_i x^{(t)} + w_R h^{(t-1)} + b_h\right)$$

$$y(t) = g_y\left(w_y h^{(t)} + b_y\right)$$

only notation changes. Concept remains Same.



Training A Recurrent Neural Network.

Recurrent Neural Nets uses back propogation algorithm, But it is applied For Every time stamp. it is Commonly known as **Back propogation through time (BTT)**

Issues with back propogation.

* **Vanishing**
* **Exploding Gradient**

* RNN's back propogation has problem with the "Vanishing" and "Exploding Gradient".

1. Vanishing

Ex:

$$2x + 4 = 8$$

$\therefore$ initalizing $x = 10$ $\quad \hat{y}$ $\quad$ error

$\Delta x = 1$ (change in "x" value)

$x = 9$	$\hat{y}$	error
$x = 8$	$\hat{y}$	error
$x = 7$	$\hat{y} = 2(7)+4 = 18$	Error 16 → high.
$x = 6$	$\hat{y}$	Error
$x = 5$	$\hat{y}$	Error
$x = 4, 3$	$\hat{y}$	Error
$x = 2$	$\hat{y} \Rightarrow 2(2)+4 = 8$	Error 0

Error is minimum.

$$2x + 4 = 8$$

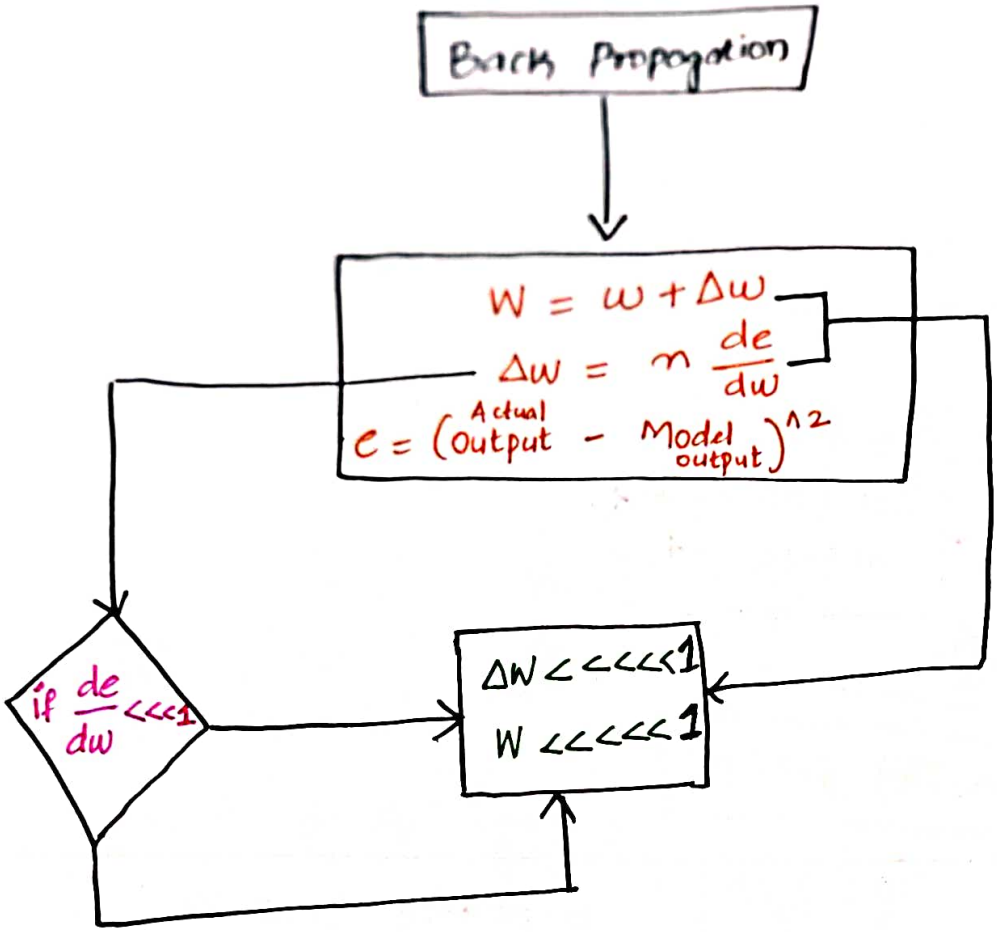
$\Delta x = 0.01$ (changes in "x" value.)

$x = 10$	$\hat{y}$	Error
$x = 9.99$	"	"
$x = 9.98$	"	"
x	"	"
⋮	"	"
⋮	"	"
⋮	"	"
$x = 2$	"	"

900 iteration

When Δw is Very low is called "Vanishing"

Δw → change in weight



2. Exploding Gradient

when Δw is Very high is called "Exploding Gradient"

Δw → change in weight

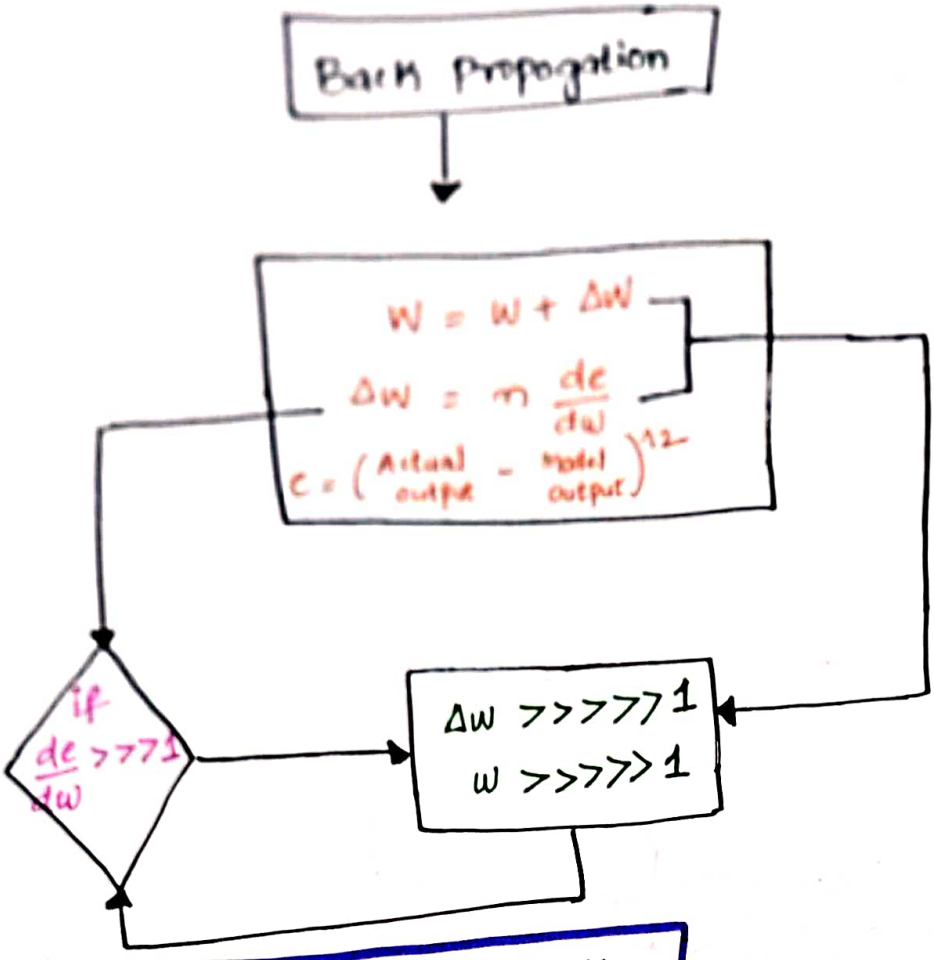
Ex: $2x + 4 = 8$

∴ initializing $x = 10$ → y → error

$x = 5$ → y → "

$x = 0$ → " → "

$\Delta x = 5$ → change in x value



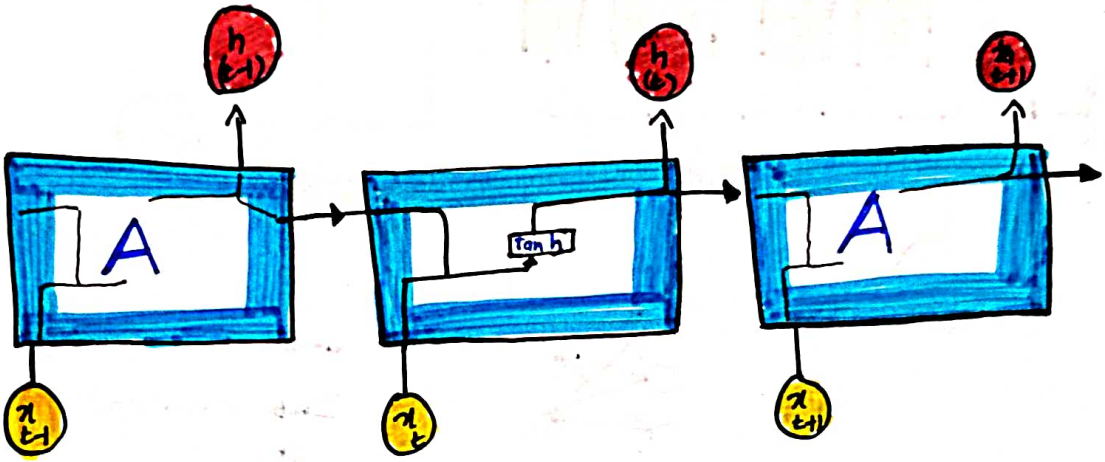
* How to overcome this problem.

Vanishing	Exploding gradients
• ReLu activation function we can use activation like Relu, which gives output one while calculating gradient • RMs prop clip the gradient when it goes higher than a threshold. • LSTM, GRUs different network	• Truncated BTT Instead of starting back propogation at last time stamp, we can choose smaller time stamp like 10 (we will lose the temporal context after 10 times stamps) • clip gradients at threshold clip the gradient when it goes higher than a threshold • RMs prop To adjust learning rate.

2. LSTM

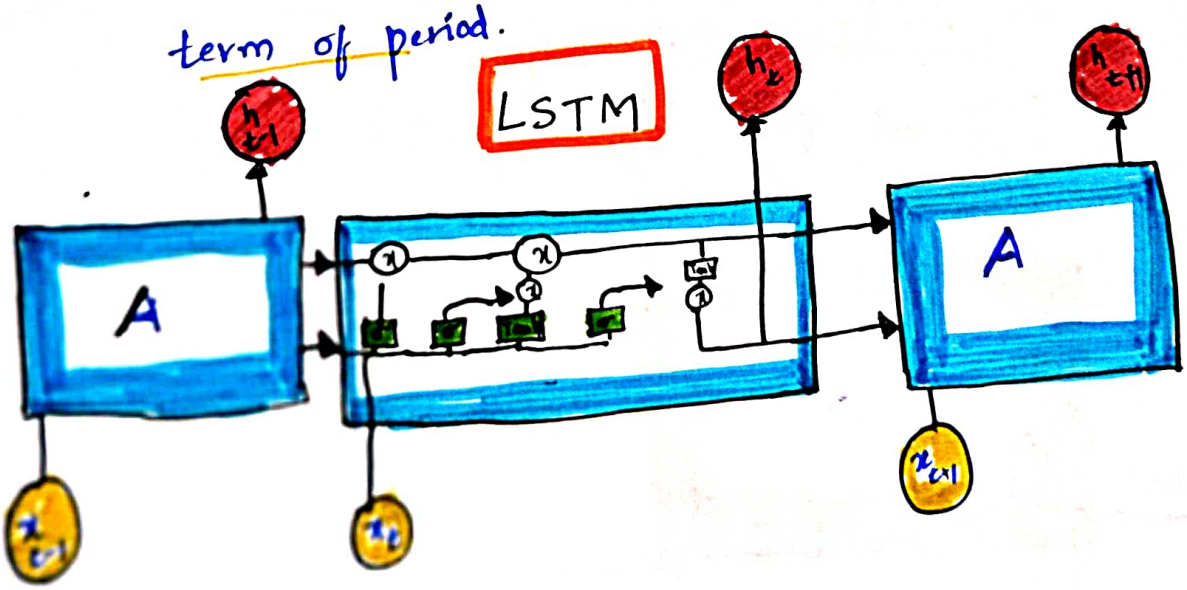
∴ RNN → Short term memory (Ex:- only Yesterday)
↳ Previous output

LSTM → Long short term memory (Ex:- Incidents in 10th class)

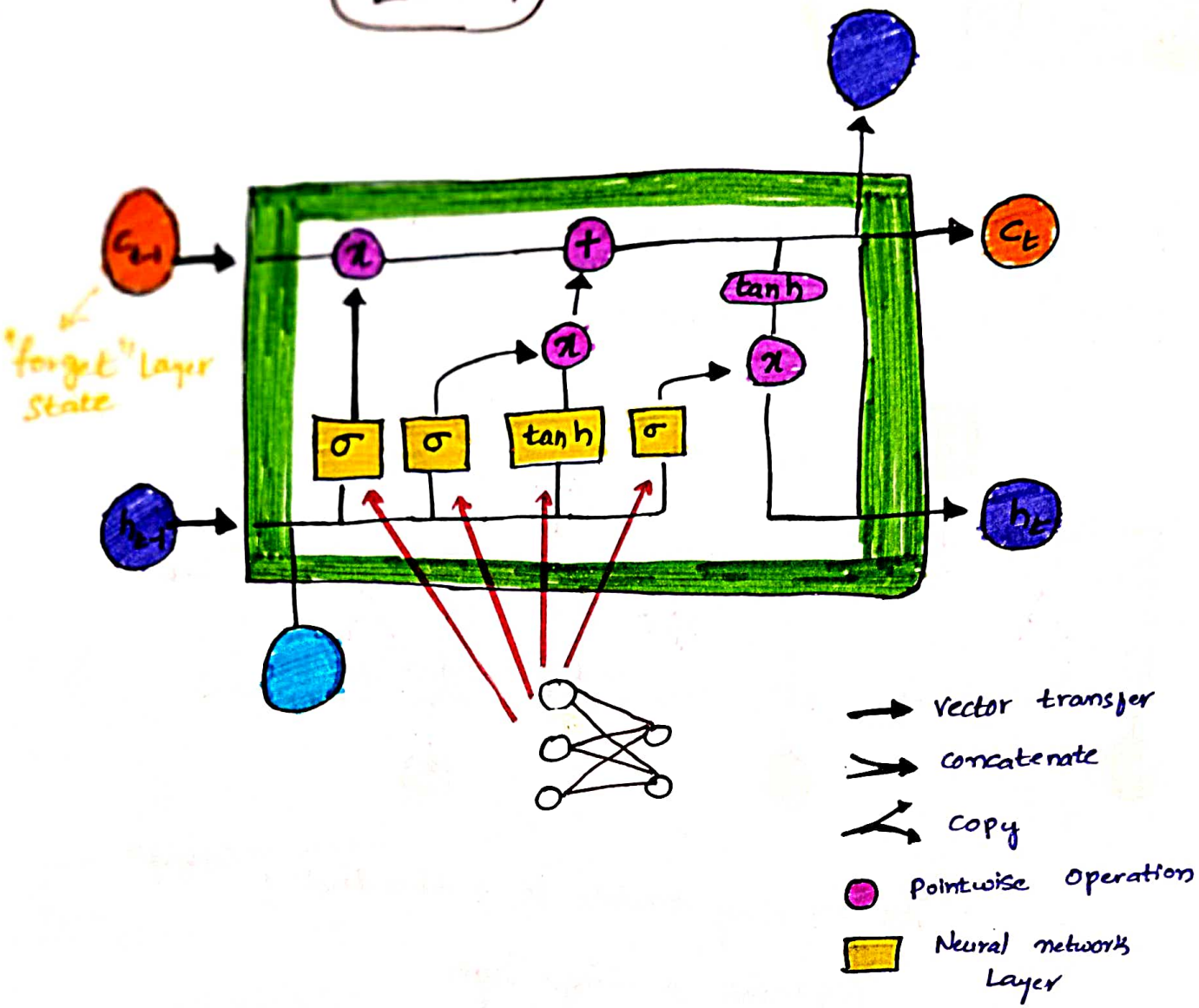


The repeating module in a standardard "RNN"
Contains a single layer

- Lstm are capable of Learning long term dependencies.
- it stores "short term memories" for the long term of period.

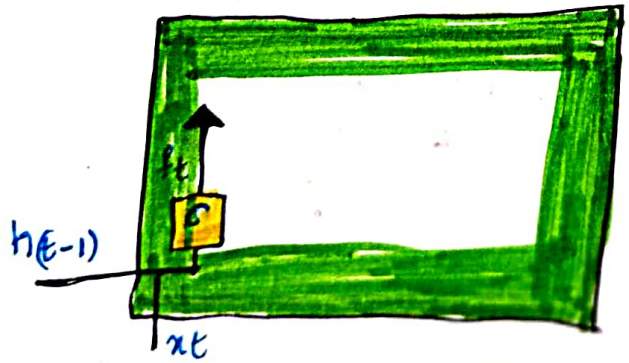


LSTM



STep : 1

The First step in The LSTM is to identify those information at are not required and will be thrown away From the ell state. this decision is made by a Sigmoid layer called as "forget gate layer"



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

$\therefore w_f$ = weight

h_{t-1} = Output from previous time stamp

x_t = new input

b_f = bias

Ex: Shirt warranted on last friday (01) Jan 04.

We cant remember because it routine for us.

it is not important to rember

if you watch a movie you will remember. not as daily watching shows. Why ? Because we watch movie, with curiosity.

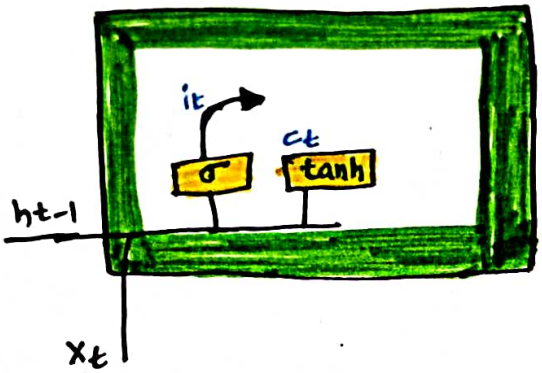
when you watch with interset, it going to rember in the brain.

Step : The replication of human intelligence artificially is known "artificial intellegence".

by taking "sigmoid activation" function (0 (or) 1). we are going to give input & we take "sigmoid". if it gives output as "0" (not important), 1 (important). it thrown into "forget state".

Step: 2 # In this part analysis takes place.

The next step is to decide, what new information were going to score in the cell state. This whole process comprises of following steps. A "Sigmoid Layer" called "Input gate layer" decides which values will be updated, next, a "Tanh Layer" creates a vector of new candidate values, that could be added to the state.



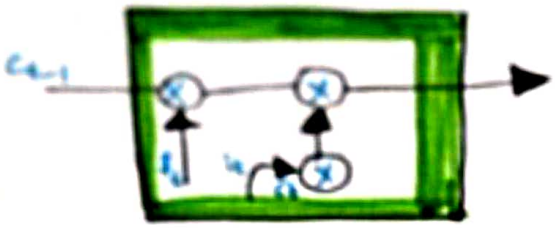
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

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

In the next step, we'll combine These two to update the state

Step: 3

Now, we will update the old cell state, C_{t-1}, into e new cell state C_t. First, we multiply the old state C_{t-1}) by f_t, forgetting the things we decided to forget earlier. Then, we add it $* \tilde{C}_t$. this is the new candidate values, scaled by how much we decided to update each state value.



$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$

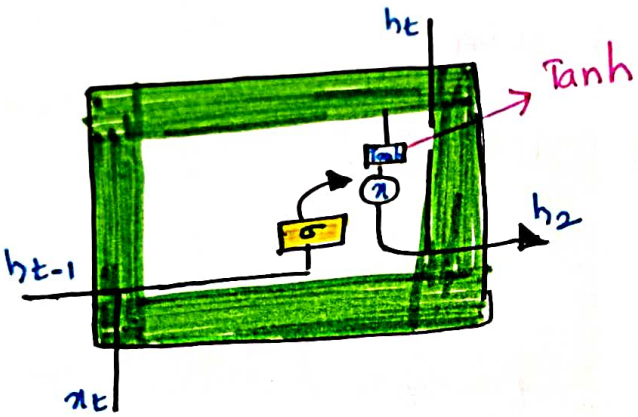
Ex:- You got Fight with brother. And Your not talking. Now, after certain time we talk. Again once we fight each other. again, we Talk. But Your not going to remember first fight that You fighted. But Your only rember the Lastest Fight with each other.
(reason)

Here, it going to update the old information. and replace with New information. based on that it transforms the data.

it is updated as in the artificial neural network in the "forgetting layer".

Step: 4

We will return a **Sigmoid Layer** which decides what parts of the cell state were going to **output**. Then, we put The cell state through **Tanh** (push The values between **-1 to +1**) and **multiply** it by the output of sigmoid gate, so that We only output The parts we decided to.



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

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

Ex:- processing will be done, transfer the output.

at last after processing, what information is not required Previous one, we don't required. Simply we are going to remove That one.

4 gates :- 1. Forget gate
2. Crossing gate
3. replacing gate
4. output gate.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

import training set

```
# df_train = pd.read_excel("Sbin.xlsx", sheet_name = 0)
```

when Your Excel You having multiple sheet, multiple data

we use this sheet : 1

```
# df_train.head()
```

Out:

Symbol	Series	Date	Prev close	Open price	high price	low price	Last Price	close price	Avg price	Total Traded	Turnover
SBIN	EQ	20-9-04	213.15	210.0	212.30	205.9	207.50	206.60	208.61	5443581	1.13560 e+10
SBIN	EQ	20-9-07	206.60	207.5	209.65	205.4	208.25	207.90	207.62	3887677	7.40944
SBIN	EQ	20-9-08	207.90	207.9	208.50	202.9	203.25	204.05	206.58	3141478	7.05294
SBIN	EQ	20-9-09	204.05	201.1	201.40	192.5	195.70	194.85	195.68	7271696	1.42298
SBIN	EQ	20-9-10	194.85	197.7	201.40	195.2	198.35	198.15	198.48	6779024	1.34522

```
# training_set = df_train.iloc[:, 4:5].values
```

[:, 4:5]: all columns; 4:5 selects the 5th column; .values: convert to array (if don't write it, data Frame)

```
# training_set
```

It:

```
array([[210.0],
       [207.5],
       [207.9],
       [201.1],
       ...
```

in deep learning, we takes values as "arrays".

Feature Scaling

→ convert to "0 to 1" because process will be fast.

```
from sklearn.preprocessing import MinMaxScaler
# sc = MinMaxScaler()
# training_set_scaled = sc.fit_transform(training_set)

# training_set.shape
```

Out: (255, 1)

Creating a data structure with 60 timesteps and 1 output

→ initial. from that. output, I have got, I am going to give to 61 record, output 61. give to 62.

```
# x_train = []
# y_train = []
for i in range(60,225):
    x_train.append(training_set[i-60:i,0])
    y_train.append(training_set[i,0])

# x_train, y_train = np.array(x_train), np.array(y_train)

x_train.shape, y_train.shape
```

]: ((165, 60), (165,))

# total = 255
removed = 60 records

Reshaping

```
# X_train = np.reshape(X_train, (X_train.shape[0],
                                 X_train.shape[1],1))
```

Part-2 Building and Training RNN

```
# importing Keras Libraries and packages
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

Initalising the RNN

```
# regressor = Sequential()
```

1. Adding The first LSTM layer and Some Dropout regularisation

1. Hyperparameter tuning
2. Regularization
In order to reduce overfitting we use
3. drop some neurons

↑ Instead of Dense, we take LSTM

```
# regressor.add(LSTM(units = 50, return_sequences = True,
                     input_shape = (X_train.shape[1],1)))
# regressor.add(Dropout(0.2))
```

↳ we have drop only 20 %. data (neurons) not more than that.

Adding second LSTM layer

```
# regressor.add(LSTM(units = 50, return_sequences = True))
# regressor.add(Dropout(0.2))
```

Adding a third LSTM Layer

```
# regressor.add(LSTM(units = 50, return_sequences = True))
# regressor.add(Dropout(0.2))
```

Adding fourth LSTM Layer

```
# regressor.add(LSTM(units = 50))
# regressor.add(Dropout(0.2))
```

Adding output Layer

```
# regressor.add(Dense(units = 1))
```

Compiling the RNN

```
# regressor.compile(optimizer = "adam", loss = "mean_squared_error")
```

Fitting The RNN to training Set

```
# regressor.fit(x_train, y_train, epochs = 100, batch_size = 32)
```

```
Epoch 1/100 [==========] - 1s 107ms/step loss:- 12400.3415
```

```
# pred = regressor.predict(X_train)

# pred
```

Out: array([[38.817867],
[38.817867],
[38.817867],
⋮

in LSTm, we are not going to give any ∅ (activation function) itis fixed.

Evaluating the RNN

```
import math
from sklearn.metrics import mean_squared_error

# rmse = math.sqrt(mean_squared_error(y_train, pred))

# rmse
```

Out: 329.6378 ⟹ Here RMSE High, we use drop function.

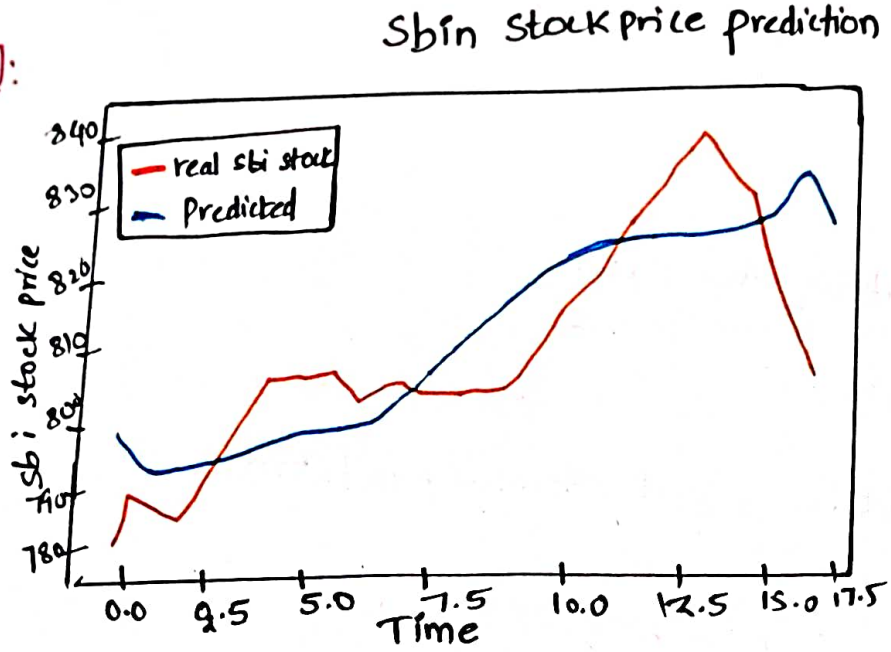
0.028692

Visualising The results

```
# plt.plot(real_stock_price, color = "red", label = "sbi stocks")

# plt.plot(predicted_stock_price, color = "blue", label = "predic

# plt.title("sbi_stock_price")

plt.xlabel("Time")
```

```
# plt.ylabel ("sbi stock price")
# plt.legend()
# plt.show().
```

Out:



16/05/22
10:30 pm.