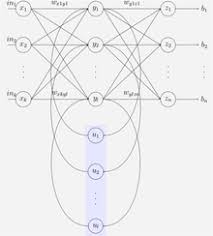
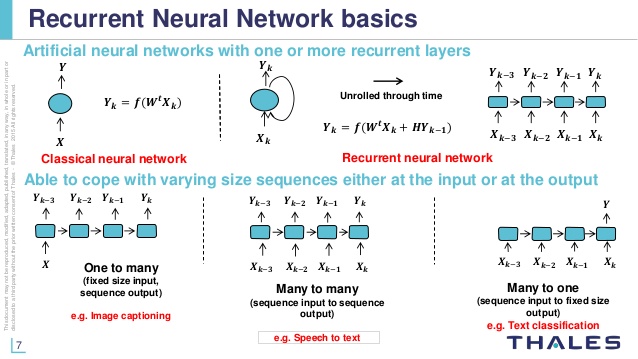
**Reccurent Neural Network**

**Short note about Reccurent Neural Network**:

A **Recurrent Neural Network** (RNN) is a class of artificial **neural network** where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence.

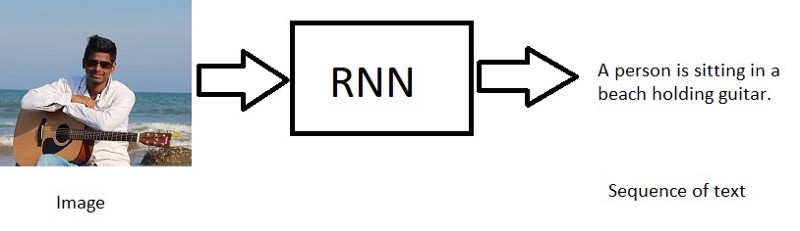




RNN has the following models

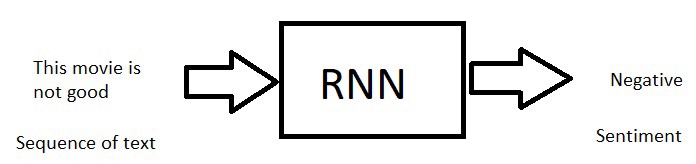
1. One to Many

RNN takes one input lets say an image and generates a sequence of words.



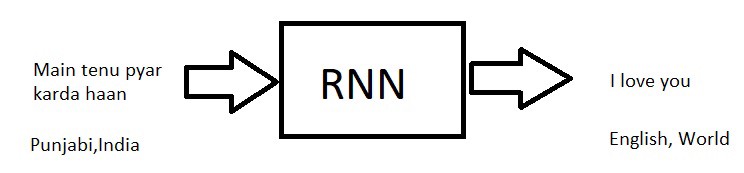
2.Many to One

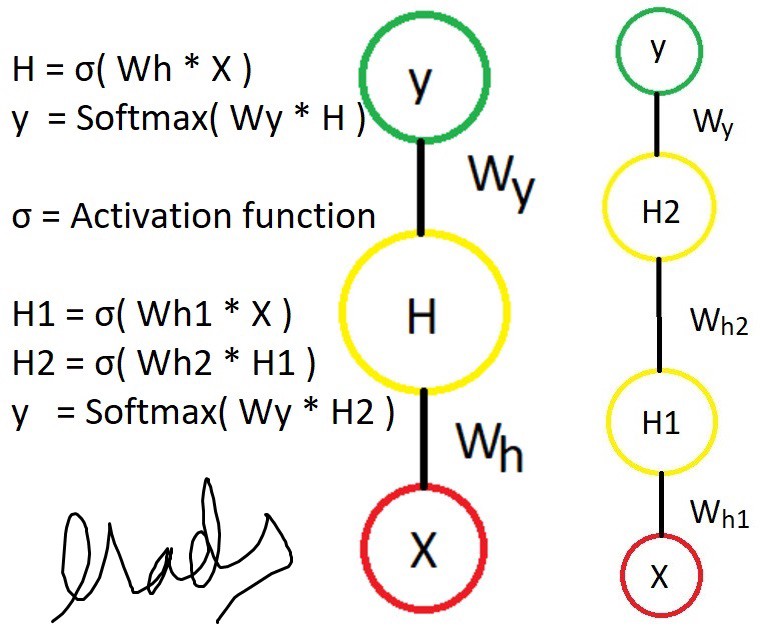
RNN takes sequence of words as input and generates one output.



3.Many to Many

RNN takes sequence of words as input and generates sequence of words as output. (lets say language translations).





The RNN cell contains a set of feed forward neural networks cause we have time steps.

The RNN has: sequential input, sequential output, multiple timesteps, and multiple hidden layers.

Unlike FFNN , here we calculate hidden layer values not only from input values but also previous time step values and Weights ( W ) at hidden layers are same for time steps.

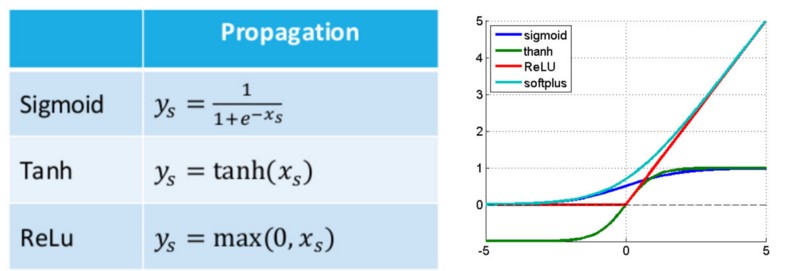
In the picture we are calculating the Hidden layer time step (t) values so

Ht = Activatefunction(input \* Hweights + W \* Ht-1)

yt = softmax(Hweight\* Ht)

Ht-1 is the previous time step and as i said W’s are same for all timesteps.

The activation function can be Tanh, Relu, Sigmoid, etc..



Above we calculated only for Ht similarly we can calculate for all other timesteps.

Steps:

1. Calculate **Ht-1** from **U** and **X**
2. Calculate **yt-1** from **Ht-1** and **V**
3. Calculate **Ht** from **U,X,W** and **Ht-1**
4. Calculate **yt** from **V** and **Ht** and so on…

Note :

1.**U** and **V** are weight vectors, different for every time step.

2.We can even calculate hidden layer( all time steps ) first then calculate y values.

3. Weight vectors are random initially.

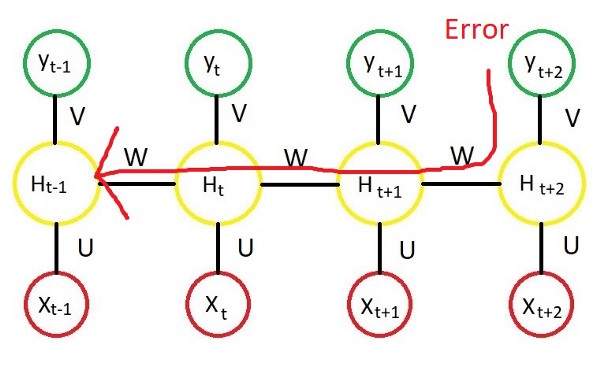
Once Feed forwarding is done then we need to calculate the error and backpropagate the error using back propagation.

we use Cross entropy as cost function ( assume you know so not going into details)

We need to calculate the below terms

1. how much does the ***total error*** change with respect to the ***output (hidden and output units)***? (or how much is a change in ***output***)
2. how much does the ***output*** change with respect to ***weights (U,V,W)***? (or how much is a change in ***weights***)

Since W’s are same for all time steps we need to go all the way back to make an update.



Remember the BP for RNN is as same as neural networks BP

but here Current time step is calculated based on the previous time step so we have to traverse all the way back.

if we apply chain rule which looks like this

Above we calculated only for Ht similarly we can calculate for all other timesteps.

Steps:

1. Calculate **Ht-1** from **U** and **X**
2. Calculate **yt-1** from **Ht-1** and **V**
3. Calculate **Ht** from **U,X,W** and **Ht-1**
4. Calculate **yt** from **V** and **Ht** and so on…

Note :

1.**U** and **V** are weight vectors, different for every time step.

2.We can even calculate hidden layer( all time steps ) first then calculate y values.

3. Weight vectors are random initially.

Once Feed forwarding is done then we need to calculate the error and backpropagate the error using back propagation.

we use Cross entropy as cost function ( assume you know so not going into details)

**BPTT ( Back propagation through time )**

if you know how Normal neural network works , the rest is pretty easy , if you don’t know, here is my article that talks about [**Artificial Neural Networks.**](https://medium.com/deep-math-machine-learning-ai/chapter-7-artificial-neural-networks-with-math-bb711169481b)

We need to calculate the below terms

1. how much does the **total error** change with respect to the **output (hidden and output units)**? (or how much is a change in **output**)
2. how much does the **output** change with respect to **weights (U,V,W)**? (or how much is a change in **weights**)

Since W’s are same for all time steps we need to go all the way back to make an update.

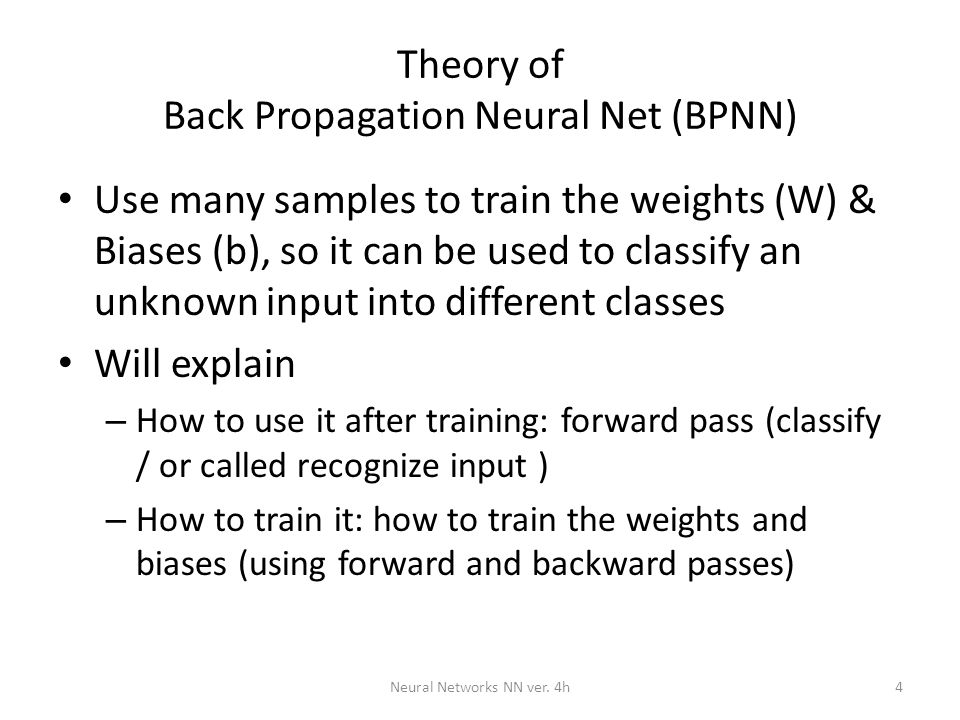
Remember the BP for RNN is as same as neural networks BP

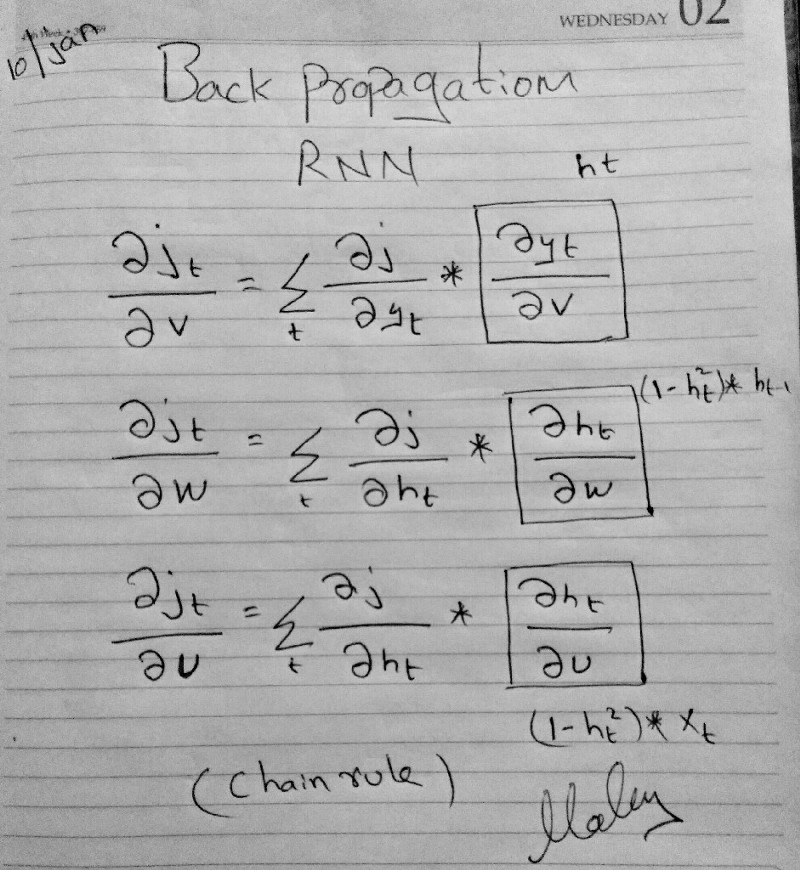
but here Current time step is calculated based on the previous time step so we have to traverse all the way back.

if we apply chain rule which looks like this

Backpropagation Algorithm

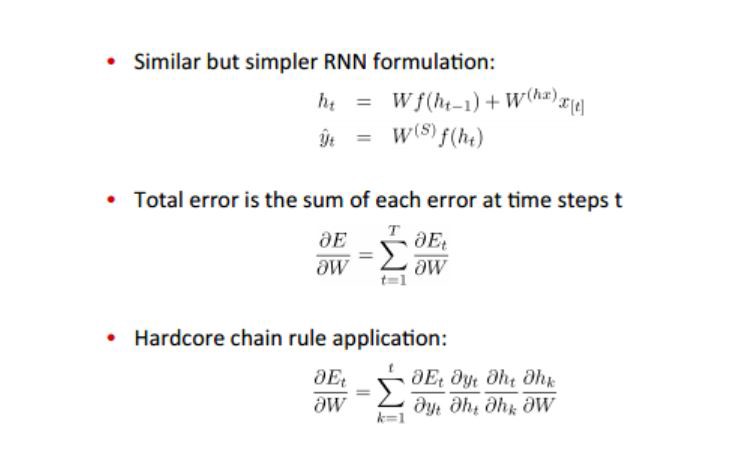
The network must recognize whether a new input vector is similar to learned patterns and produce a similar output. The backpropagation algorithm is used to find a local minimum of the error function. ... The gradient of the error function is computed and used to correct the initial weights.





W’s are same for all the time steps so the chain rule expands more and more

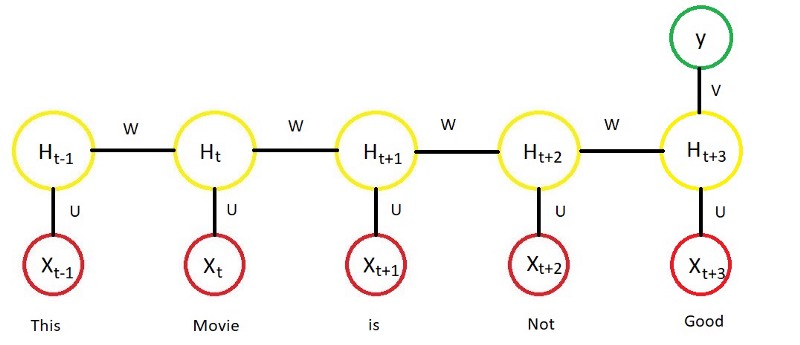
A similar but a different way of working out the equations can be seen in Richard Sochers’s [Recurrent Neural Network](http://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf) lecture slide.



So here **Et** is same as our **J( θ)**

U, V and W should get updated using any optimization algorithms like gradient descent ( Take a look at my story here [**GD**](https://medium.com/deep-math-machine-learning-ai/chapter-1-2-gradient-descent-with-math-d4f2871af402)).

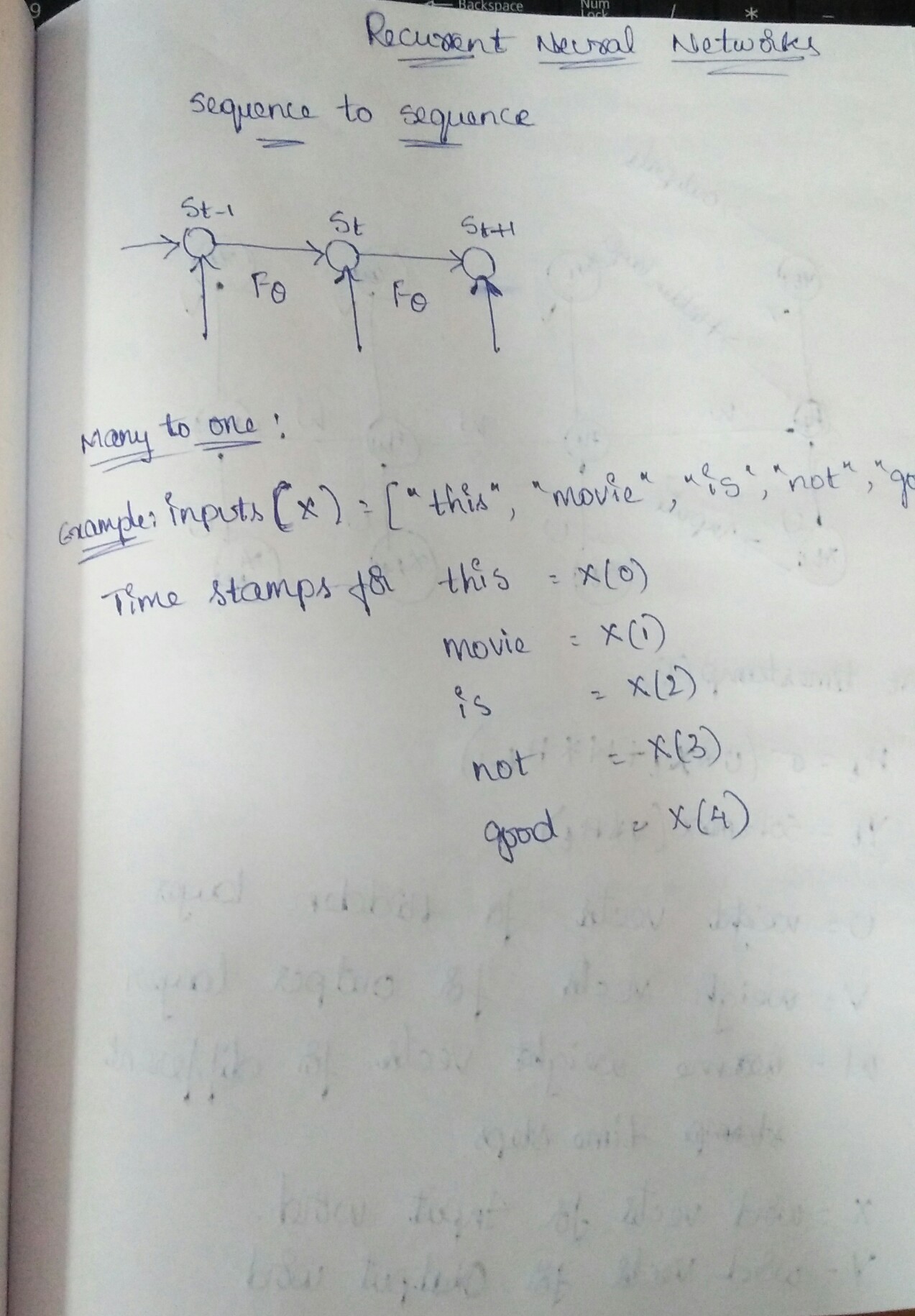
Now if we go back and talk about our sentiment problem here is the RNN for that

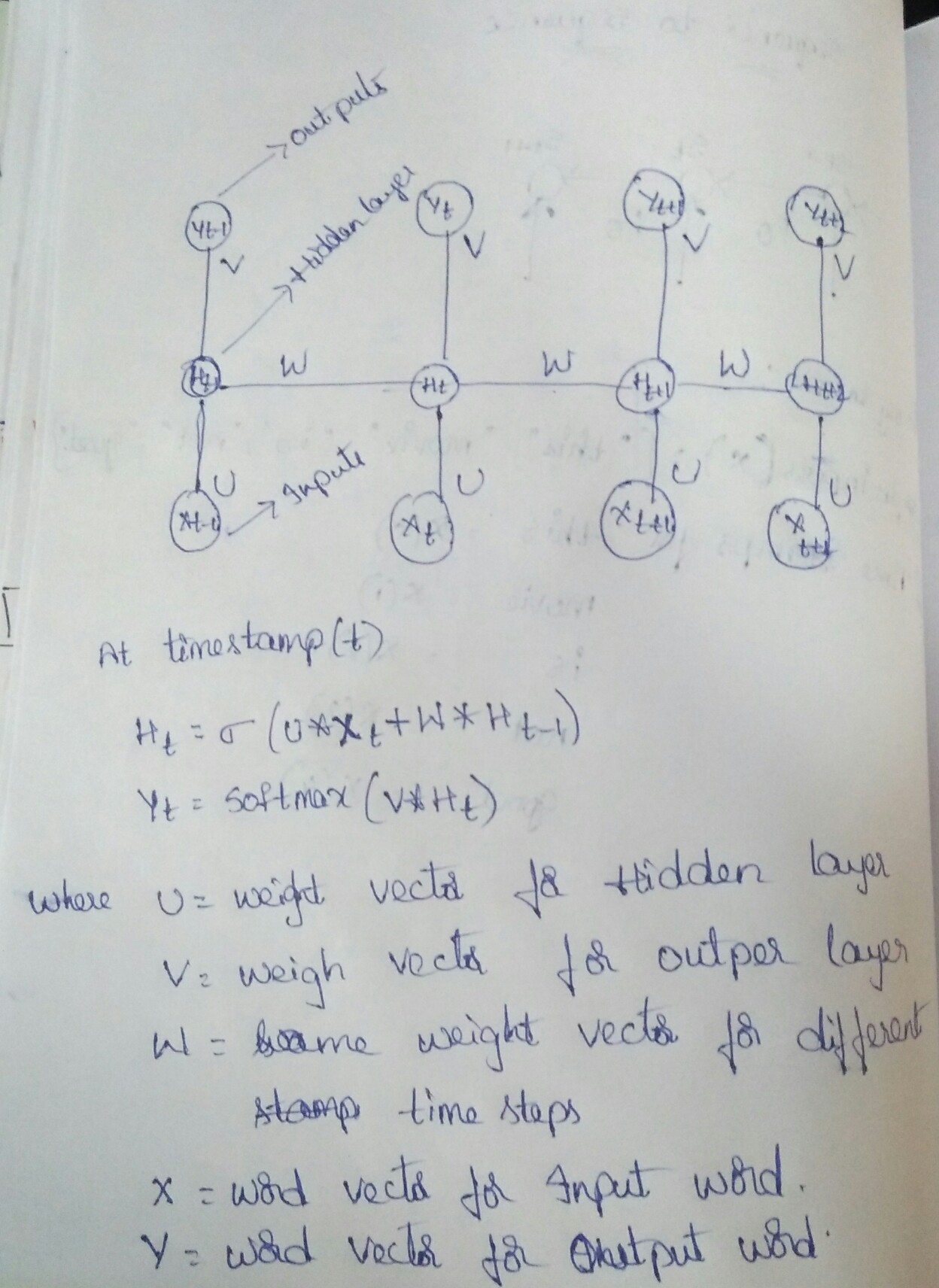


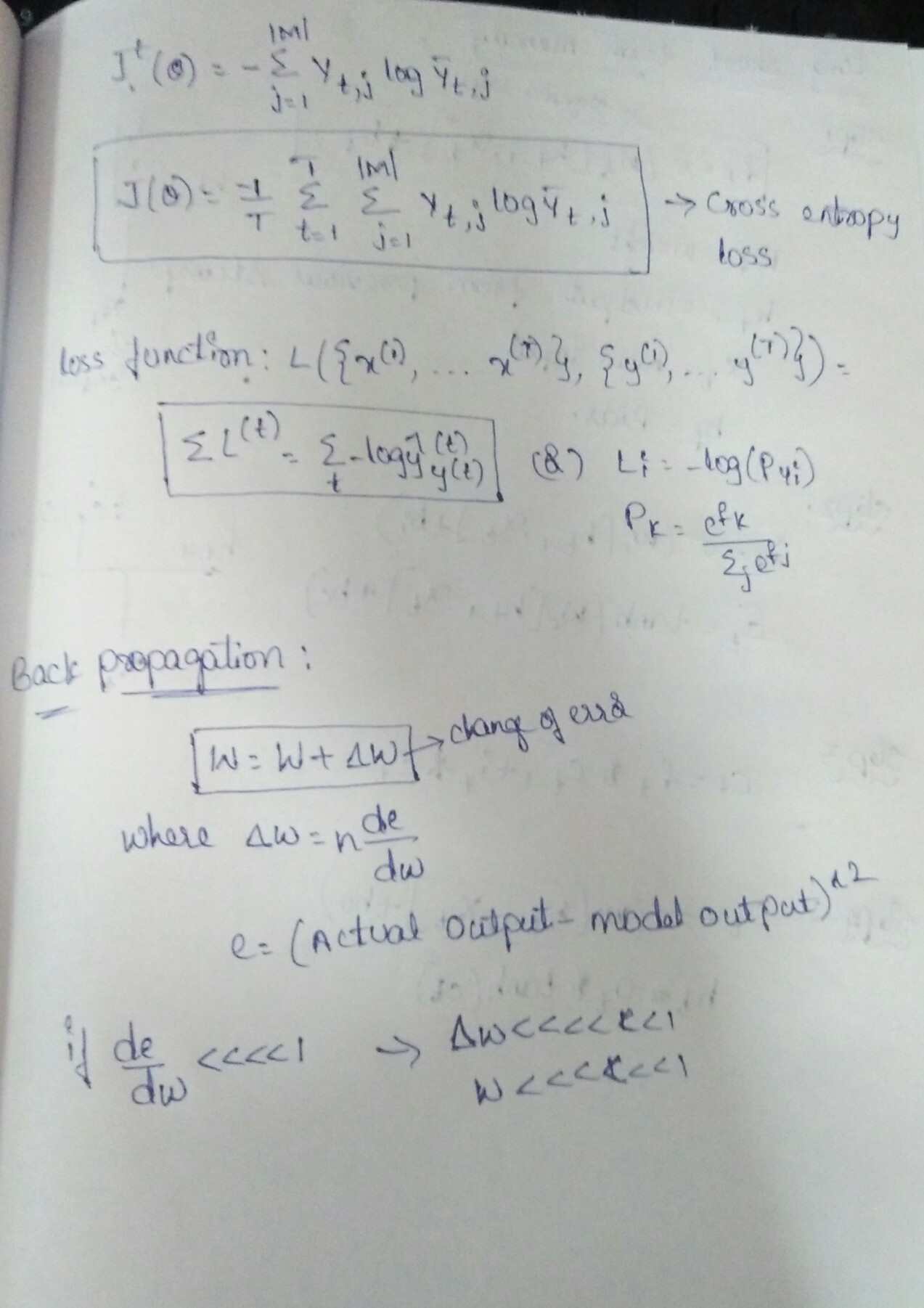
We give word vectors or one hot encoding vectors for every word as input and we do feed forward and BPTT ,Once the training is done, we can give new text for prediction.

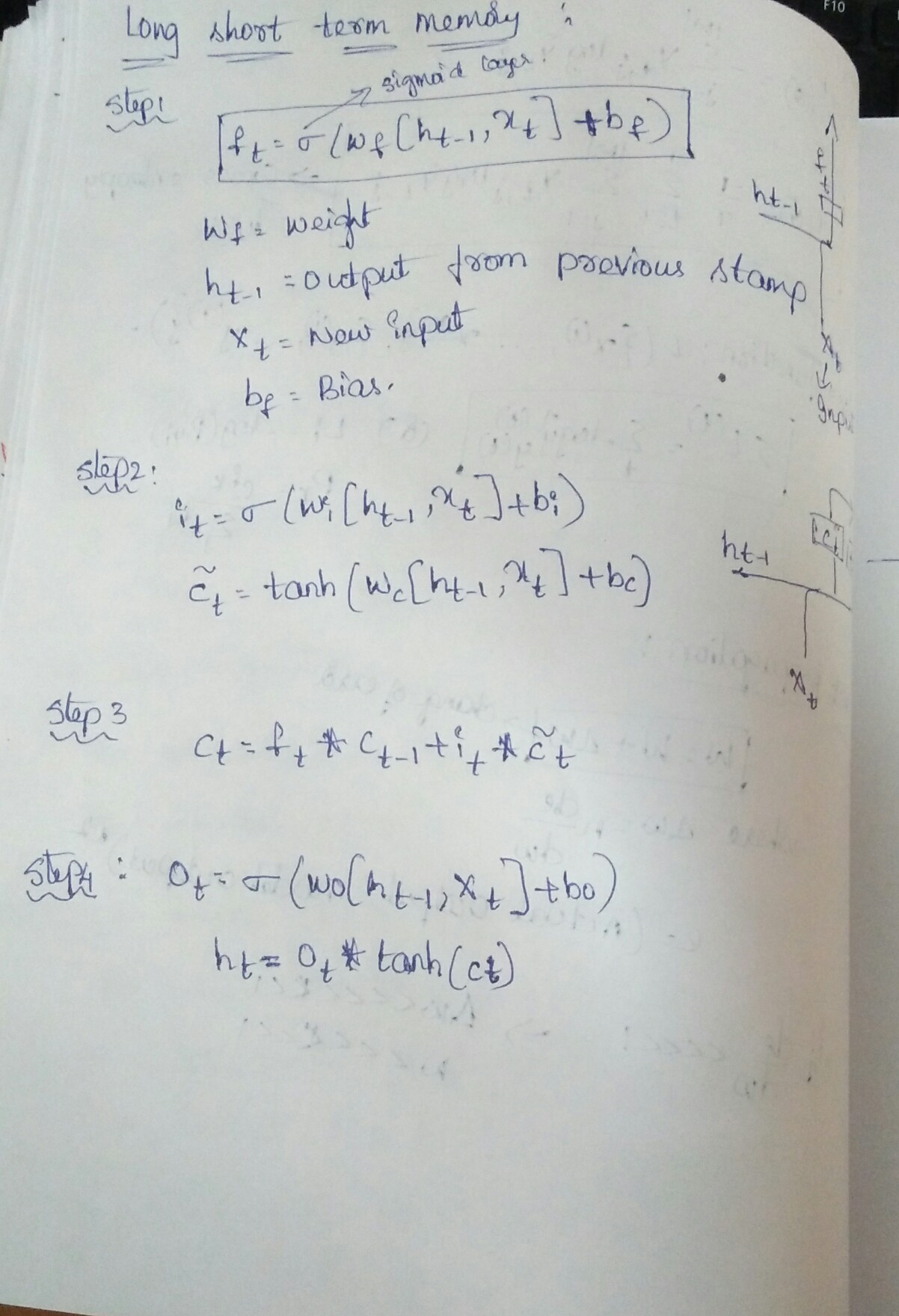
It learns something like w

hereever “not” + positive word = negative.



****





**Use Cases-**

* Weather forecasting
* Text generation
* Sequence generation
* Voice recognition
* Stock prices
* Traffic volume
* Music,vedio,audio
* Temperature and location
* Height and weight
* Car speed and brand
* Automation driving
* Mobile
* cameras.
* To find the dimensions of the room.
* To create the objects.
* To move the objects.
* Finding the changes in Pollution
* To check the existence of the customer in the bank.
* To check the market strategy.
* To find the price fluctuation of Amazon products.

Python-

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the training set

training\_set = pd.read\_csv('C:\\Users\\Rama\\Desktop\\abc.csv')

training\_set = training\_set.iloc[:,0:1].values

# Feature Scaling

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler()

training\_set = sc.fit\_transform(training\_set)

# Getting the inputs and Outputs

X\_train = training\_set[0:5498]

y\_train = training\_set[1:5499]

# Reshaping(5498 represents no.of sequence and 1 row(height) and 1 column(width))

X\_train = np.reshape(X\_train, (5498, 1, 1))

# Part 2 - Building the RNN

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

# Initialising the RNN

regressor = Sequential()

# Adding the first LSTM layer and some Dropout regularisation

regressor.add(LSTM(units = 4, activation = 'sigmoid', input\_shape = (None, 1)))

#

# Adding a second LSTM layer and some Dropout regularisation(1 output)

regressor.add(Dense(units = 1))

#

# Adding a third LSTM layer and some Dropout regularisation

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

# Fitting the RNN to the Training set

regressor.fit(X\_train, y\_train, epochs = 200, batch\_size = 32)

#

# Getting the predicted stock price of 2017

inputs = (582.25)

inputs = sc.transform(inputs)

inputs = np.reshape(inputs, (1, 1, 1))

predicted\_stock\_price = regressor.predict(inputs)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)