A Major Project Report
on

"Epileptic Seizure Prediction Using Machine Learning and Deep Learning "

Title of project

By

1. Khush Dave (A-823)
2. Ajay H. Desai (A-824)
3. Shubham M Hedavkar (A-838)

under the guidance of

Prof. Preeti Satao

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI

Department of Computer Engineering

University of Mumbai

May – 2021

MANJARA CHARITABLE TRUST

RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI

Juhu-Versova Link Road Versova, Andheri(W), Mumbai-53.

Certificate

Department of Computer Engineering

This is to certify that

1. Khush Dave(A-823)
2. Ajay H. Desai (A-824)
3. Shubham Hedavkar(A-838)

Have satisfactorily completed this project entitled

"Epileptic Seizure Prediction using Machine Learning and Deep Learning Technique"

Towards the partial fulfilment of the

BACHELOR OF ENGINEERING
IN
(COMPUTER ENGINEERING)

as laid by University of Mumbai.

Guide	**Head of Department**
Prof. Preeti Satao	**Prof. S. P. Khachane**

Principal

Dr. Sanjay Bokade

Project Report Approval for B. E.

This project report entitled **"Epileptic Seizure Prediction using Machine Learning and Deep Learning Technique"** by ***Khush Dave, Ajay H. Desai, and Shubham Hedavkar*** is approved for the degree of Computer Engineering.

Examiners

1. ———————————

2. ———————————

Date:

Place:

Declaration

We wish to state that the work embodied in this project titled **"Epileptic Seizure Prediction using Machine Learning and Deep Learning Technique"** forms our own contribution to the work carried out under the guidance of **Prof. Preeti Satao** at the Rajiv Gandhi Institute of Technology.

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Khush Dave (A-823)

Ajay H. Desai (A-824)

Shubham M. Hedavkar (A-838)

Abstract

Epileptic seizures occur due to disorders in brain functionality which can affect a patient's health. Prediction of epileptic seizures before the beginning of the onset is quite useful for preventing the seizure by medication. Machine Learning, Deep Learning techniques and computational methods are used for predicting epileptic seizures from Electroencephalograms (EEG) signals. However, preprocessing of EEG signals for noise removal and features extraction are two major issues that have an adverse effect on both anticipation time and true positive prediction rate. Therefore, we propose a model that provides reliable methods of both preprocessing and feature extraction. Our model predicts epileptic seizures' sufficient time before the onset of seizure starts and provides a better true positive rate. We have applied empirical mode decomposition (EMD) for preprocessing and have extracted time and frequency domain features for training a prediction model. The proposed model detects the start of the preictal state, which is the state that starts a few minutes before the onset of the seizure, with a higher true positive rate as compared to traditional methods.

Contents

List of Figures viii

List of Tables ix

List of Algorithms x

List of Symbols xi

1 Introduction 11

1.1 Introduction Description 11

1.2 For whom is this system……11

1.3 Organization of Report 11

2 Literature Review 13

2.1 Survey Existing system 13

2.2 Limitation Existing system or research gap 14

2.3 Problem Statement and objectives 14

2.3.1 Objectives 14

2.4 Scope 14

3 Proposed System 15

3.1 Analysis/Framework/Algorithm 15

3.1.1 Libraries……16

3.1.2 Algorithms……18

3.2 Details of Hardware and Software 26

3.2.1 Hardware Requirements 26

3.2.2 Software Requirements……..26

3.3 Design Details 27

3.3.1 System Flow/System Architecture……27

3.3.2 Basic Seizure Prediction Block Diagram……...29

3.3.3 Data Set Used……30

3.4 Methodology/Procedures……31

4 Results & Discussions 33

4.1 Result And Output……...33

4.2 Discussions……..……45

4.3 Discussion-Comparative study/Analysis……..……46

5 Conclusion 47

References 48

Acknowledgements 49

List of Figures

3.1.2 Recurrent Neural Network Loops…….18

3.1.2 Unrolled Recurrent Neural Network…….18

3.1.2 Problem of Long Term Dependencies……...19

3.1.2 LSTM Networks……....20

3.1.2 Step-by-Step LSTM Walk Through……..22

3.1.2 Variants on LSTM…….24

3.3.1 System Flow(For Machine Learning Model).....25

3.3.1 System Flow(For Deep Learning Model).....26

3.3.2 Basic Seizure Prediction Block Diagram……..27

List of Tables

4.3 Discussion - Comparative Study..……44

List of Algorithms

3.1.2 KNN……17

3.1.2 LSTM……..18

Chapter 1

Introduction

1.1 Introduction Description

To diagnose epilepsy or to find its cause, an Electroencephalogram(EEG) is performed which helps record the electrical activity of the brain. Machine Learning and Deep Learning models are used to predict epileptic seizures. These models include EEG signal acquisition, signal preprocessing, features extraction from the signals, and finally classification between different seizure states. The objective of the prediction model with Machine Learning and Deep Learning was to detect the preictal state's sufficient time before seizure onset starts. We believe our project has the potential to be carried on as a solution that can be easily and readily deployed in tomorrow's market.

1.2 For whom is this system

- Healthcare systems
- Diagnosis of Neurological ailments

1.3 Organization of report

Describe every chapter (what every chapter contain)

- Ch.1 Introduction:

Introduction to Epilepsy Seizure Prediction and feature.

- Ch.2 Literature Review:

Analysis done based on technical papers read and what we have understood and what we would like to implement or enhance.

- Ch.3 Proposed System:

The changes or enhancements to the existing system such that we can improve our project in our future developments

- Ch.4 Results & Discussion:

What we have analyzed or implemented as part of our requirements gathering and analysis phase.

- Ch.5 Conclusion :

We have successfully carried out comparative analysis of Epilepsy Seizure Prediction based on their Machine Learning and Deep Learning Model.

Chapter 2

Literature Review

Research has been done to provide the best possible solution to detect Epilepsy Seizures using EEG signals. Many Machine Learning models have been created to find the best possible solution to this problem. But as EEG signals are quiet, tricky to work with no model till date has proved to be satisfactory.

2.1 Survey existing system

1. **Author** :Falco-Walter, Jessica J., Ingrid E. Scheffer, and Robert S. Fisher.
Paper Title :The new definition and classification of seizures and epilepsy.
Year : 2018
Topic :Epilepsy Research.

2. **Author** :Ullah, Ihsan, Muhammad Hussain, and Hatim Aboalsamh.
Paper Title : An automated system for epilepsy detection using EEG brain signals based on deep learning approach.
Year : 2018
Topic :Expert Systems with Applications.

3. **Author** :Usman, Syed Muhammad, Muhammad Usman, and Simon Fong.
Paper Title :Epileptic seizures prediction using Deep Learning methods.
Year : 2017
Topic :Computational and mathematical methods in medicine.

Falco-Walter et al. discuss the upgraded classifications of epilepsies and seizures. These new classifications include some previously unclassifiable seizure types. In their proposed method they include two main seizure types namely basic and expanded. Doctors and Neurologists can make use of the basic classification whereas the expanded classification will be preferred by researchers and epileptologists.

Ihsan et al. in their research work establish that the existing methods do not perform well when classification of the ternary scenario is concerned. The known methods give a maximum accuracy of 97±1% for this case by the is. They overcame this problem by proposing a system which has deep learning as its foundation. The system consists of an ensemble constructed out of convolutional one dimensional neural networks (P-1D-CNN). The performance of the system is

commendable when used on a benchmark dataset. To clinically validate this model is an area of future work. Incorporation of the proposed system on a wearable device
can also be considered as the future scope.

Hussein et al. identify the challenges encountered by EEG based seizure detection systems in their research work. EEG signals are not stationary and are prone to several noise types. To overcome these challenges they introduce a deep learning based approach that automatically learns the distinct EEG features of epileptic seizures. The EEG data that is in time series format is first transformed into segments of non overlapping epochs.

2.2 Limitation Existing system or research gap

- There is still no physical handy device to detect Epilepsy seizure prediction due to lack of potential Machine Learning or Deep Learning Model. And as for Prediction problems it is always about getting the right data set and constantly improving on accuracy. Through our Comparative Analysis of Machine Learning and Deep Learning Model, search for a more accurate model is supported.

2.3 Problem Statement and objectives

Epilepsy is a group of neurological disorders characterized by recurrent epileptic seizures.This seizures can cause severe shaking attacks therefore in these seizures patients can hurt her/himself very badly. Luckily epilepsy is a curable disease that 70% of cases are controllable with early diagnosis and medication. Therefore in order to make an early diagnosis, many epilepsy diagnostic methods have been developed over time. One of them is diagnosis with the help of EEG signals. However, the visual inspection process for discriminating EEGs is a time consuming and high costly process. To address these challenges, we introduce the use of a comparative model of Machine Learning and Deep Learning-based approach that automatically learns the discriminative EEG features of epileptic seizures.

2.3.1 Objectives

- To make epileptic seizure detection very quickly and accurately by using EEG signals of the brain data with Machine Learning and Deep Learning techniques.

2.4 Scope

Epilepsy Seizure Prediction is aimed to make epileptic seizure detection models using EEG signals with Machine Learning and Deep Learning techniques. This can be employed in healthcare industries and hospitals to detect and treat patients efficiently.

Chapter 3

Proposed System

3.1 Analysis/Framework/Algorithm:

- The preictal state is very useful for seizure prediction, as it starts a few minutes before the seizure.
- The aim of this research is to predict epileptic seizure by detecting the start of preictal state's sufficient time before the ictal state or onset of seizure starts.
- Consequently, we have used a publically free online available dataset of CHB-MIT. The dataset has been acquired by placing 23 electrodes on the scalp of 22 subjects.
- We have performed preprocessing of the data in two stages; in the first stage, 23 channels of EEG signals are converted into a surrogate channel, which is a single signal to improve the SNR. In the second stage of preprocessing, empirical mode decomposition (EMD) has been applied to the surrogate channel for further increasing SNR.

Algorithms:

The classification Models we are considering for our Comparative Analysis of Machine Learning and Deep Learning Models are:

- KNN (K Nearest Neighbor)
- LSTM (Long Short-Term Memory)

We will be performing a comparative analysis of Machine Learning and Deep Learning models to identify the proper working of the algorithm on our data provided.

3.1.1 Libraries

A) **PANDAS** -In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

B) **NUMPY**- NumPy can be abbreviated as Numeric Python, is a Data analysis library for Python that consists of multi-dimensional array-objects as well as a collection of routines to process these arrays. In this tutorial, you will be learning about the various uses of this library concerning data science.

C) **MATPLOTLIB** - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.

D) **SEABORN**- Used for statistical data visualization.
Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in the exploration and understanding of data. One has to be familiar with Numpy and Matplotlib and Pandas to learn about Seaborn. It provides a high-level interface for drawing attractive and informative statistical graphics.

E) **SKLEARN** - Scikit-learn (Sklearn) is the most useful and robust library for Deep Learning in Python. It provides a selection of efficient tools for Deep Learning and statistical modeling including classification, regression, clustering, and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy, and Matplotlib.

F) **Tensorflow** **-** The TensorFlow software handles data sets that are arrayed as computational nodes in graph form. The edges that connect the nodes in a graph can represent multidimensional vectors or matrices, creating what are known as tensors. Because TensorFlow programs use a data flow architecture that works with generalized intermediate results of the computations, they are especially open to very large-scale parallel processing applications, with neural networks being a common.TensorFlow 2 is an end-to-end, open-source machine learning platform. You can think of it as an infrastructure layer for differentiable programming. It combines four key abilities:

- Efficiently executing low-level tensor operations on CPU, GPU, or TPU.
- Computing the gradient of arbitrary differentiable expressions.
- Scaling computation to many devices, such as clusters of hundreds of GPUs.
- Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile and embedded devices.

G) **Keras** **-** Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result as fast as possible is key to doing good research.*

Keras is:

- Simple -- but not simplistic. Keras reduces developer *cognitive load* to free you to focus on the parts of the problem that really matter.
- Flexible -- Keras adopts the principle of *progressive disclosure of complexity*: simple workflows should be quick and easy, while arbitrarily advanced workflows should be *possible* via a clear path that builds upon what you've already learned.
- Powerful -- Keras provides industry-strength performance and scalability: it is used by organizations and companies including NASA, YouTube, or Waymo.

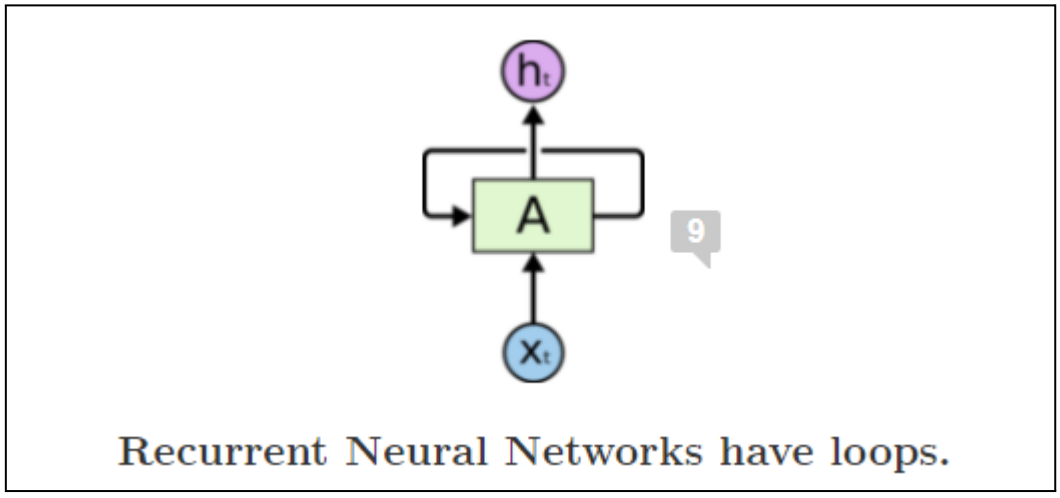
3.1.2 Algorithms:

- ***K Nearest Neighbor (KNN)* :-**
  K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
    - The K-NN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories.
    - K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm.
    - The K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
    - K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
    - It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
    - The KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
    - **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know whether it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

- ***LSTM (Long-Short Term Memory) :-***

Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.
Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

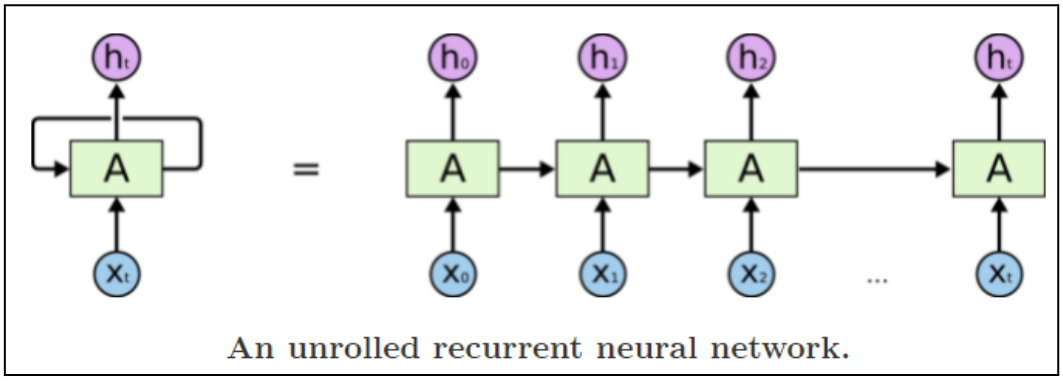
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

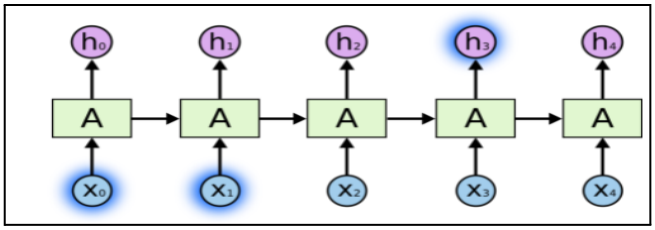


An unrolled recurrent neural network.

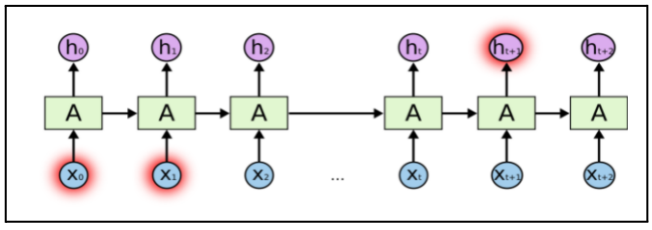
This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.
And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on. I'll leave discussion of the amazing feats one can achieve with RNNs to Andrej Karpathy's excellent blog post, The Unreasonable Effectiveness of Recurrent Neural Networks. But they really are pretty amazing.
Essential to these successes is the use of "LSTMs," a very special kind of recurrent neural network which works, for many tasks, much much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

The Problem of Long-Term Dependencies

One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame. If RNNs could do this, they'd be extremely useful. But can they? It depends.
Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the sky," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France… I speak fluent French." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.
Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

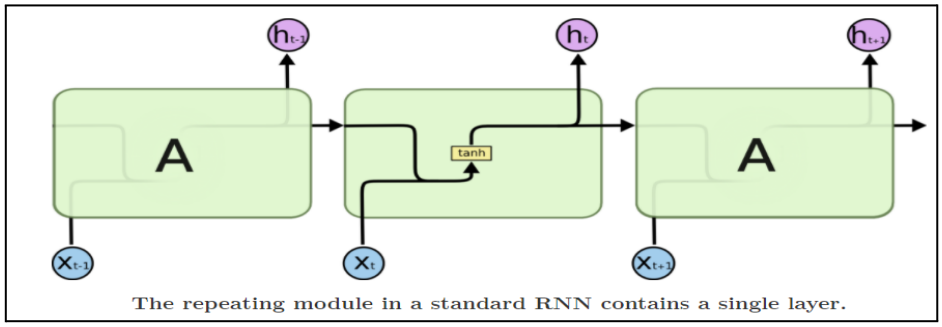


LSTM Networks

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.1 They work tremendously well on a large variety of problems, and are now widely used.

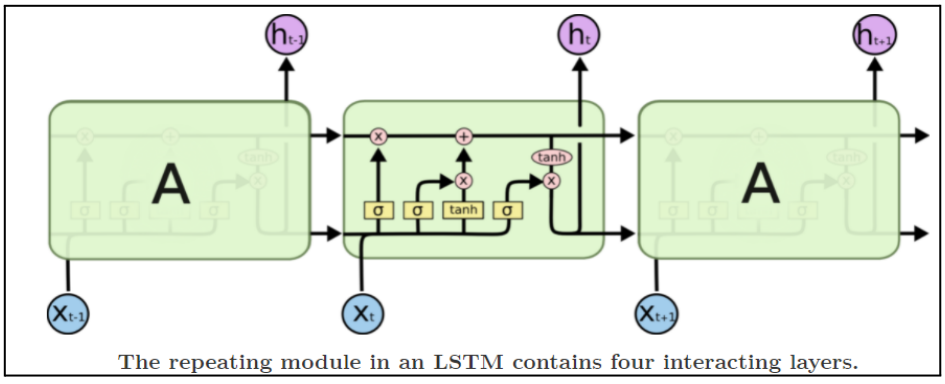
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



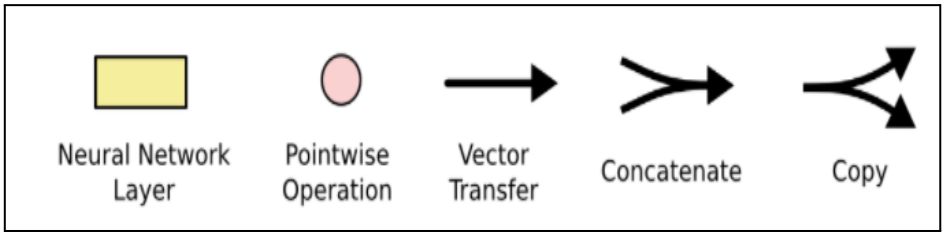
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

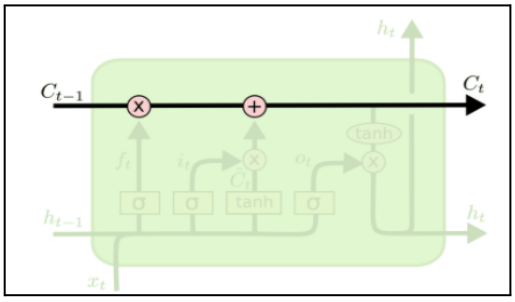
Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.



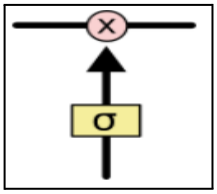
In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



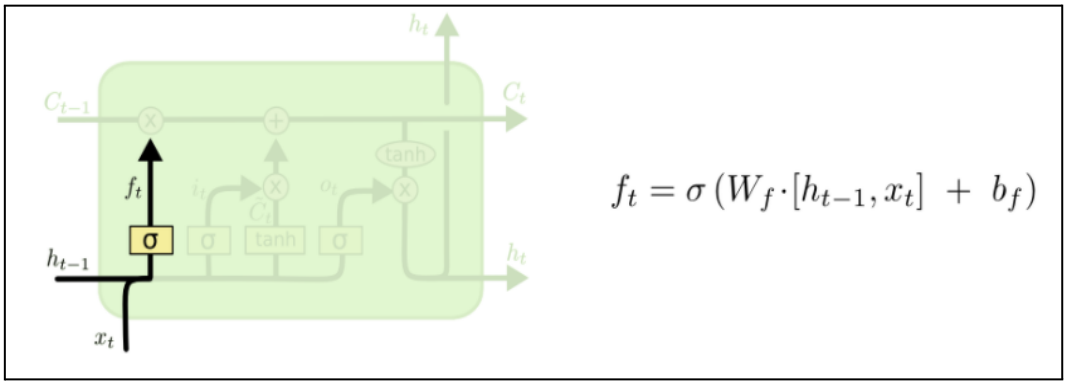
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

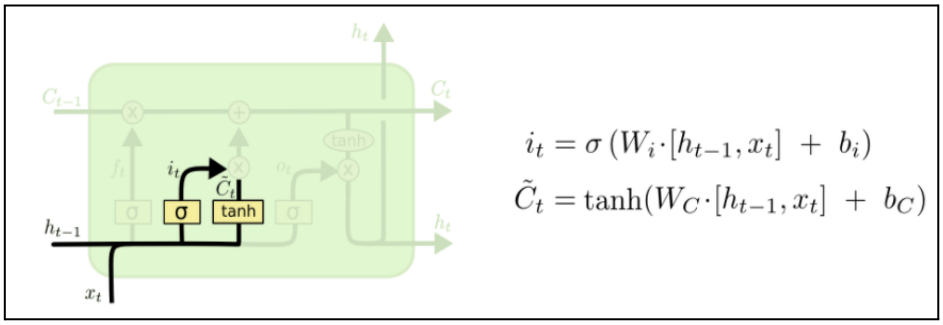
The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at ht−1 and xt, and outputs a number between 0 and 1 for each number in the cell state Ct−1. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."

Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, C~t, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

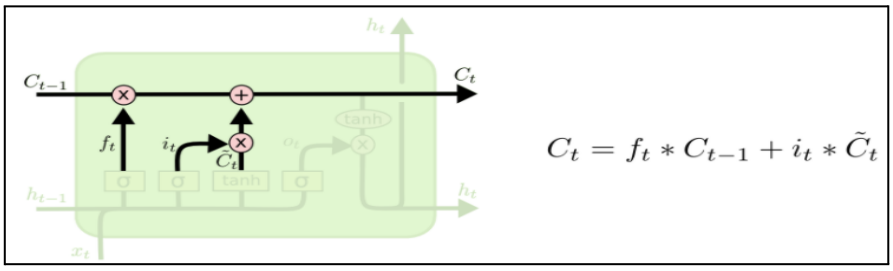
In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



It's now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.

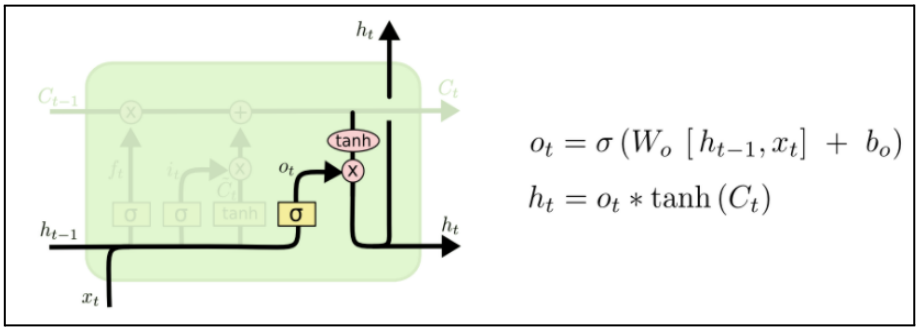
We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add it∗C~t. This is the new candidate values, scaled by how much we decided to update each state value.

In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



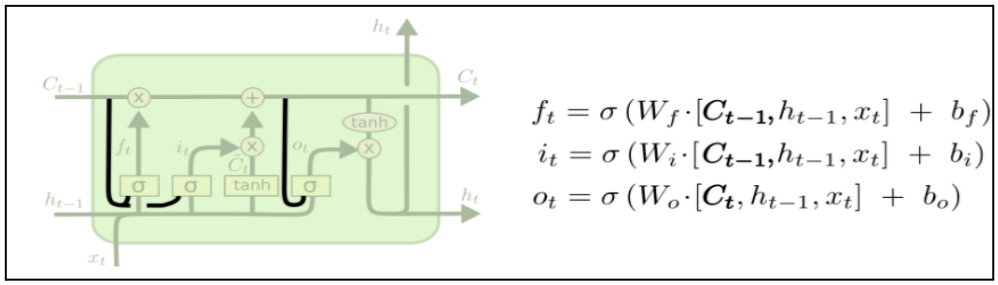
Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



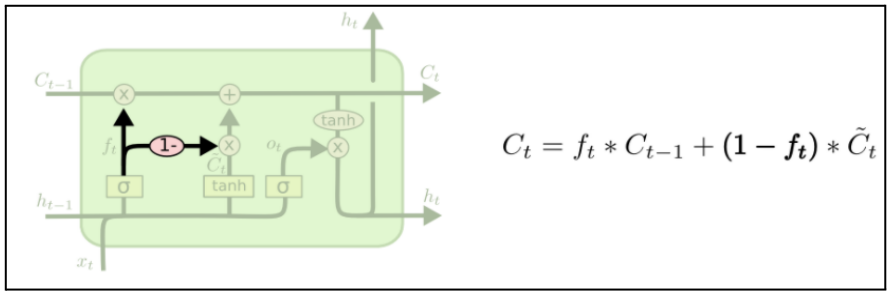
Variants on Long Short Term Memory

Not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them.
One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.

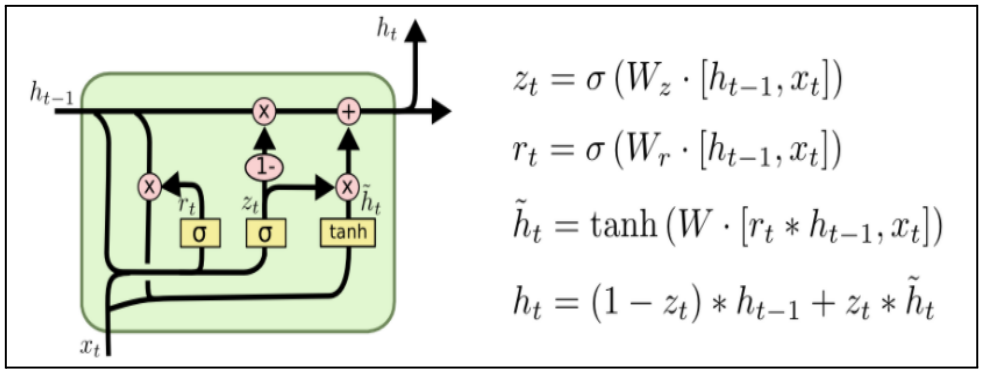


The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



3.2 Details of Hardware and Software

3.2.1 Hardware requirements (minimum)

- 4-bit 2.6 GHz Intel core i5 CPU
- 8 GB RAM
- Windows 10 environment

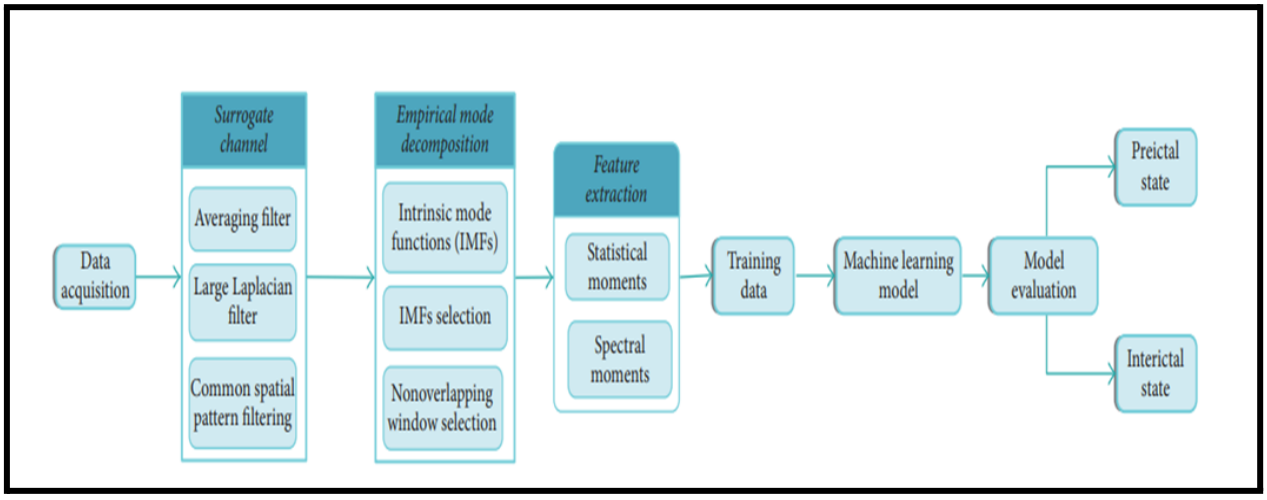
3.2.2 Software requirements

- Jupiter Lab (Python)
- Matlab

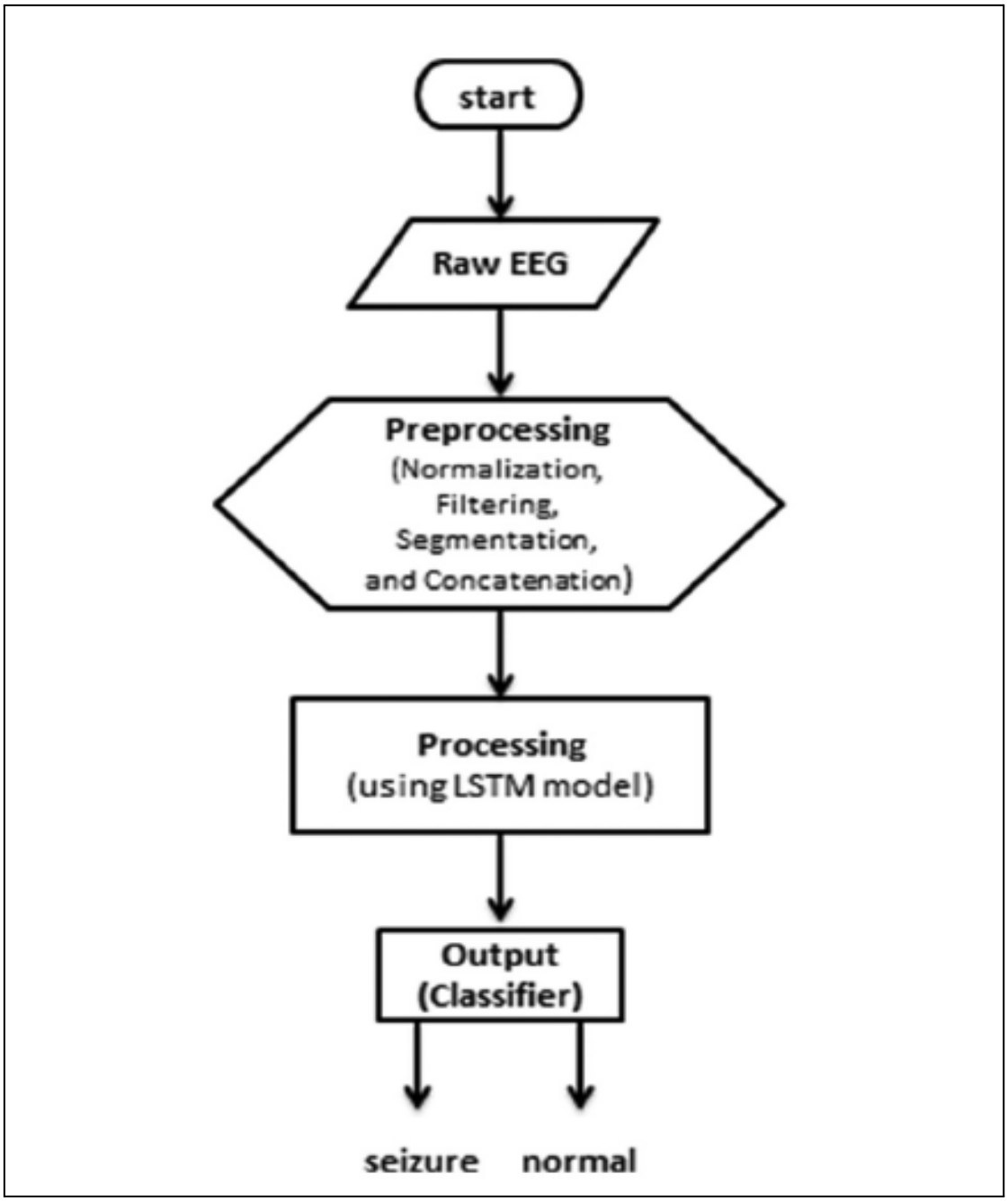
3.3 Design Details

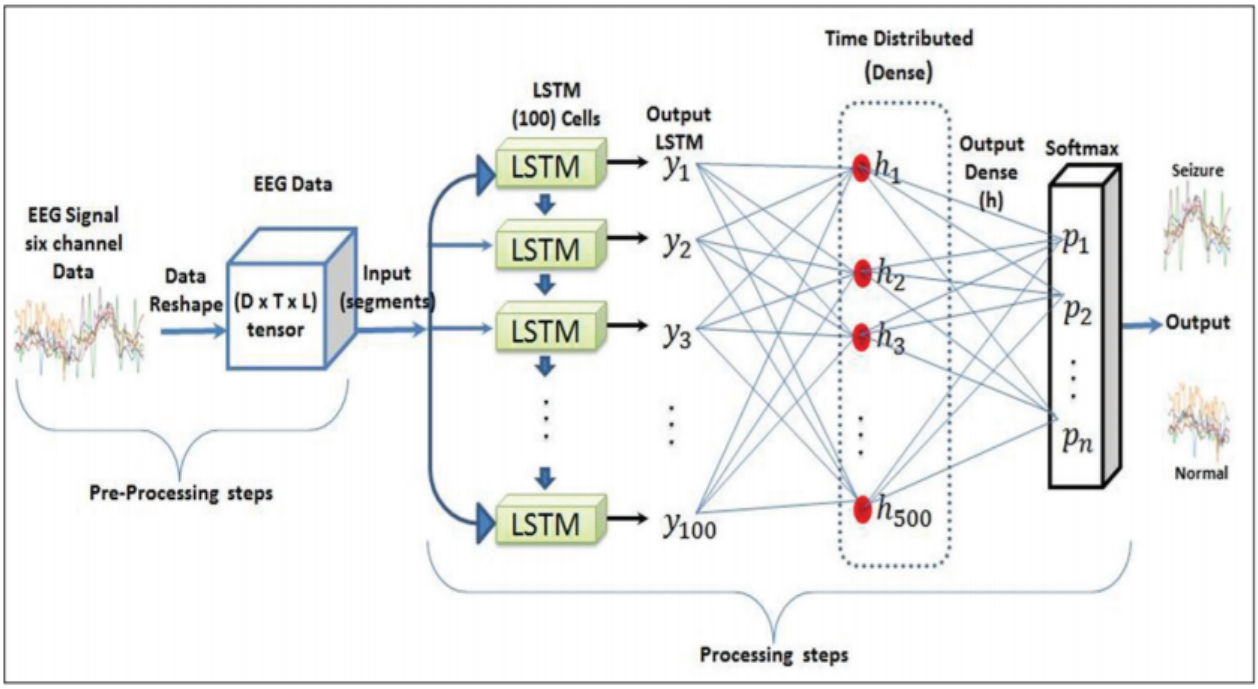
3.3.1 System Flow/ System Architecture

➔ **For Machine Learning based Model :-**

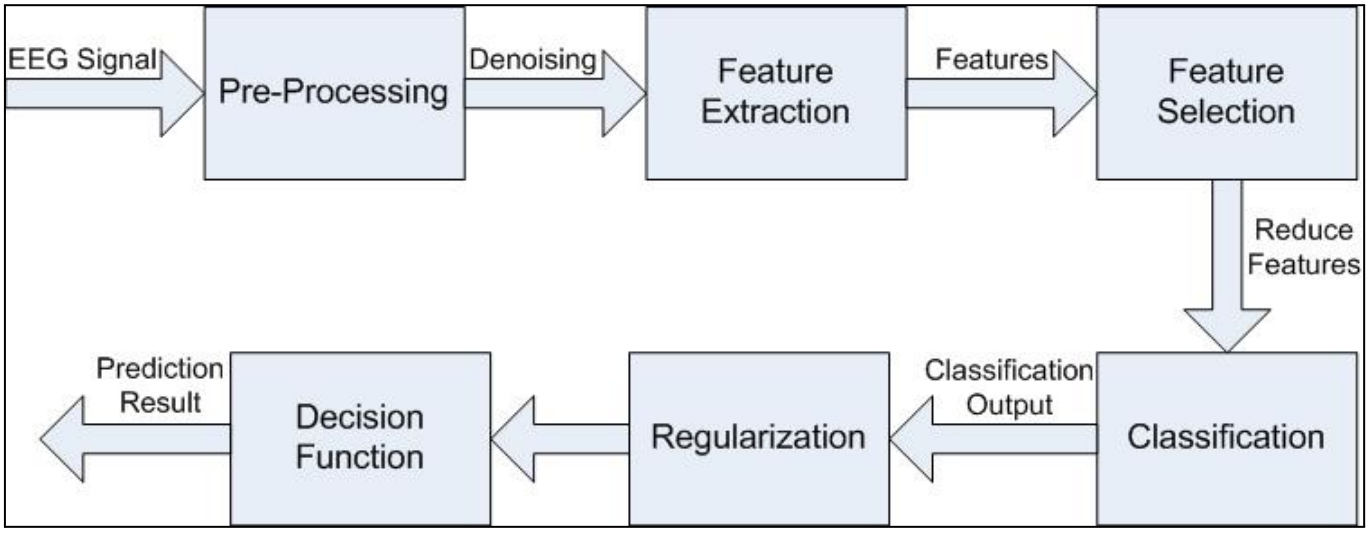


➔ **For Deep Learning based Model :-**





3.3.2 Basic Seizure Prediction Block Diagram



3.3.3 Data Set Used :-

- The dataset includes 4097 electroencephalograms (EEG) readings per patient over 23.5 seconds, with 500 patients in total.
- The 4097 data points were then divided equally into 23 chunks per patient; each chunk is translated into one row in the dataset.
- Each row contains 178 readings that are turned into columns; in other words, there are 178 columns that make up one second of EEG readings.

All in all, there are 11,500 rows and 180 columns with the first being patient ID and the last column containing the status of the patient, whether the patient is having a seizure or not..

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	y
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31	-77	-103	-127	-116	-83	-51	4
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146	152	157	156	154	143	129	1
2	X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	...	57	64	48	19	-12	-30	-35	-35	-36	5
3	X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	...	-82	-81	-80	-77	-85	-77	-72	-69	-65	5
4	X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	...	4	2	-12	-32	-41	-65	-83	-89	-73	5

3.4 Methodology/Procedures *(your approach to solve the problem)*

A. For Machine Learning Based Model Using KNN

Step 1 : We first start by importing the required libraries and also reading the dataset.

Step 2 : Then we check for the null values present in our dataset, and in our case there is no null value present.

Step 3 : Then we check for unique values in our output dependent variable 'y' i.e [1,2,3,4,5] and out of these values only 'y=1' value has epileptic seizure present.

Step 4 : In this step, we drop the unnamed column which consists of patient ID as it is of no use and edit our dependent 'y' to "OUTPUT_LABEL".

Step 5 : Here we visualize our output column and we get to know that almost 20% of people are having epileptic seizures present in their brain.

Step 6 : In this step we find the correlation of the first 16 feature columns and visualize it with the help of a heat map.

Step 7 : In this step, we standardize our feature columns as KNN is a distance based classifier that classifies new observations based on similarity measures with labelled observations of the training set.

Step 8 : Here, we perform the train test split based on 70-30 % proportion.

Step 9 : In this step, we plot the graph of error rate vs k value and we observe that as k value increases error rate also increases. So we choose k=4 for our further proceedings.

Step 10 : Finally our kNeighborsClassifiers model predicts our model's accuracy based on k value.

B. For Deep Learning Based Model Using LSTM

Eplileptical - 1 Not Epileptical - 0

1. Model.ipnyb:

   First i visualised the data, for all the 5 classes.

   (i) Then define a Recurrent Neural Network architecture that has 2 layers of LSTMs.

   (ii)Then defined the optimiser as 'adam' and loss as 'binary_crossentropy' which gave better results than 'categorical_crossentropy'.

   (iii)Then we processed the data by :

   taking 1 in every 4 samples,then normalizing the data and using the previously splitted test data as validation data.

   (iv) Saved the model after training it for 50 epochs.

   (v) Plotted the rate at which the loss and accuracy are changing.

2. Final.ipynb:

   The trained model is loaded then it is used for prediction on the validation set. Now the predictions are changed for the binary class, i.e., if the prediction is 1 then the signal is epileptical otherwise the class is made 0 and the signal is non epileptical. Same procedure is followed for the training set.

Chapter 4

Results & Discussions

4.1 Results and Outputs

In this section we have analysed the Epilepsy Seizures dataset for checking the features and attributes provided and understand which data we have and accordingly develop our model.

A. For Machine Learning Model Using KNN

Importing Required Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```

Performing EDA

```
In [2]: df = pd.read_csv('data.csv') #reading data
```

```
In [3]: df.head()
```

Out[3]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	y
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31	-77	-103	-127	-116	-83	-51	4
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146	152	157	156	154	143	129	1
2	X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	...	57	64	48	19	-12	-30	-35	-35	-36	5
3	X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	...	-82	-81	-80	-77	-85	-77	-72	-69	-65	5
4	X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	...	4	2	-12	-32	-41	-65	-83	-89	-73	5

5 rows × 180 columns

Checking for null values

```
In [4]: df.isnull()
```

Out[4]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	y
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11495	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11496	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11497	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11498	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11499	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

11500 rows × 180 columns

```
In [5]: df.isnull().values.any()
```

```
Out[5]: False
```

Finding the unique values in our dependent variable

In [6]:
```
df.y.unique()
```

Out[6]:
```
array([4, 1, 5, 2, 3], dtype=int64)
```

- All subjects falling in classes 2, 3, 4, and 5 are subjects who did not have epileptic seizure. Only subjects in class 1 have epileptic seizure

- So our aim here will to keep the output value of y=1 as it is and convert all remaining values i.e 2,3,4,5 to 0

In [7]:
```
df["OUTPUT_LABEL"] = df.y == 1
df["OUTPUT_LABEL"] = df["OUTPUT_LABEL"].astype(int)
df.pop('y')
df.drop(df.columns[0], axis=1, inplace=True)
```

In [8]:
```
df.head()
```

Out[8]:

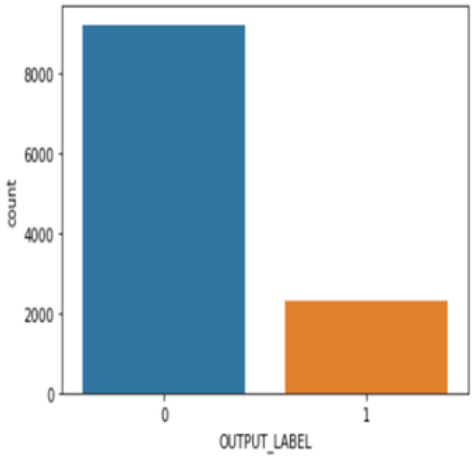
	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	OUTPUT_LABEL
0	135	190	229	223	192	125	55	-9	-33	-38	...	-17	-15	-31	-77	-103	-127	-116	-83	-51	0
1	386	382	356	331	320	315	307	272	244	232	...	164	150	146	152	157	156	154	143	129	1
2	-32	-39	-47	-37	-32	-36	-57	-73	-85	-94	...	57	64	48	19	-12	-30	-35	-35	-36	0
3	-105	-101	-96	-92	-89	-95	-102	-100	-87	-79	...	-82	-81	-80	-77	-85	-77	-72	-69	-65	0
4	-9	-65	-98	-102	-78	-48	-16	0	-21	-59	...	4	2	-12	-32	-41	-65	-83	-89	-73	0

5 rows × 179 columns

Plotting the count of dependent variable

In [9]:
```
sns.countplot(x='OUTPUT_LABEL',data=df)
```

Out[9]:
```
<AxesSubplot:xlabel='OUTPUT_LABEL', ylabel='count'>
```

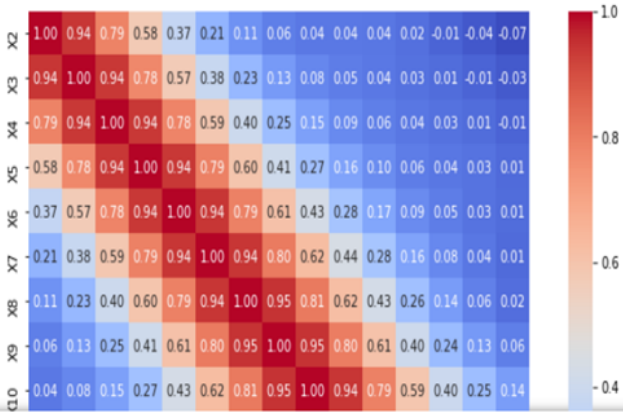


- From the above plot we can say that around 20% people are having a seizure present and rest are seizure free

In [12]:
```
# Considering the first 15 feature cloumn
sel_features= list(df.columns[1:16])
print(sel_features)
corr = df[sel_features].corr()
plt.figure(figsize=(12,8))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt= '.2f',
            xticklabels= sel_features, yticklabels= sel_features,
            cmap= 'coolwarm')
```

```
['X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16']
```

Out[12]: <AxesSubplot:>



Standardizing the Variables

- k-nearest neighbors is a distance based classifier that classifies new observations based on similarity measures (e.g., distance metrics) with labeled observations of the training set. Standardization makes all variables to contribute equally to the similarity measures .

In [13]:
```
from sklearn.preprocessing import StandardScaler
```

In [14]:
```
scaler = StandardScaler()
```

In [19]:
```
scaler.fit(df.drop('OUTPUT_LABEL',axis=1))
```

Out[19]:
```
StandardScaler()
```

In [21]:
```
scaled_features = scaler.transform(df.drop('OUTPUT_LABEL',axis=1))
```

In [22]:
```
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1]) #independent features
df_feat.head()
```

Out[22]:

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X169	X170	X171	X172	
0	0.885051	1.209929	1.462764	1.439539	1.242366	0.818262	0.380910	-0.014267	-0.163196	-0.198415	...	0.105907	-0.041630	-0.020282	-0.107131	-(
1	2.400577	2.366190	2.239441	2.109257	2.037441	1.996033	1.941660	1.719102	1.546403	1.484569	...	1.074264	1.057700	0.972841	0.943023	(
2	-0.123287	-0.169154	-0.225131	-0.172743	-0.149016	-0.179743	-0.312756	-0.409056	-0.484131	-0.547478	...	0.233004	0.407820	0.455214	0.361582	(
3	-0.564057	-0.542530	-0.524794	-0.513803	-0.503073	-0.545472	-0.591462	-0.575607	-0.496475	-0.453979	...	-0.426689	-0.436417	-0.417531	-0.397851	-(
4	0.015586	-0.325731	-0.537025	-0.575814	-0.434746	-0.254129	-0.058825	0.041250	-0.089134	-0.329314	...	0.118012	0.085916	0.082040	0.005598	-(

5 rows × 178 columns

Performing the train test split

In [23]:
```
from sklearn.model_selection import train_test_split
```

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['OUTPUT_LABEL'],test_size=0.30)
```

In [29]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

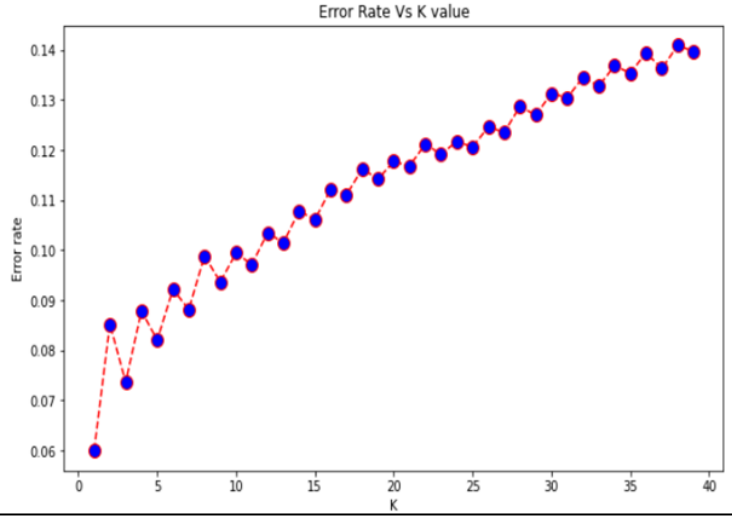
```
(8050, 178)
(3450, 178)
(8050,)
(3450,)
```

In [30]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [31]:
```
# Checking the error rate
error_rate = []

for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [32]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40), error_rate, color='red',linestyle='dashed', marker='o',markerfacecolor='blue',markersize=10)
plt.title('Error Rate Vs K value')
plt.xlabel('K')
plt.ylabel('Error rate')
```

Out[32]: Text(0, 0.5, 'Error rate')



```
In [93]: from sklearn.metrics import confusion_matrix,classification_report
         # Now with k=3
         knn = KNeighborsClassifier(n_neighbors=3)

         knn.fit(X_train,y_train)
         pred = knn.predict(X_test)

         print('With k=3')
         print('\n')
         print(confusion_matrix(y_test,pred))
         print('\n')
         print(classification_report(y_test,pred))
```

```
With k=3


[[2752    4]
 [ 250  444]]


              precision    recall  f1-score   support

           0       0.92      1.00      0.96      2756
           1       0.99      0.64      0.78       694

    accuracy                           0.93      3450
   macro avg       0.95      0.82      0.87      3450
weighted avg       0.93      0.93      0.92      3450
```

```
In [65]: from sklearn import metrics

In [107]: knn = KNeighborsClassifier(n_neighbors=3)

In [108]: knn.fit(X_train,y_train)

Out[108]: KNeighborsClassifier(n_neighbors=3)

In [109]: pred = knn.predict(X_test)

In [110]: print("The accuracy we get after using knn is :",metrics.accuracy_score(y_test, pred))

          The accuracy we get after using knn is : 0.9263768115942029
```

B. For Deep Learning Model Using LSTM

Importing Required Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Performing EDA

In [2]:
```
df = pd.read_csv('data.csv') #reading data
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	y
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31	-77	-103	-127	-116	-83	-51	4
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146	152	157	156	154	143	129	1
2	X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	...	57	64	48	19	-12	-30	-35	-35	-36	5
3	X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	...	-82	-81	-80	-77	-85	-77	-72	-69	-65	5
4	X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	...	4	2	-12	-32	-41	-65	-83	-89	-73	5

5 rows × 180 columns

Checking for null values

In [4]:
```
df.isnull()
```

Out[4]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	y
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11495	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11496	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11497	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11498	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
11499	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

11500 rows × 180 columns

In [5]:
```
df.isnull().values.any()
```

Out[5]:
```
False
```

Finding the unique values in our dependent variable

In [6]:
```
df.y.unique()
```

Out[6]:
```
array([4, 1, 5, 2, 3], dtype=int64)
```

- All subjects falling in classes 2, 3, 4, and 5 are subjects who did not have epileptic seizure. Only subjects in class 1 have epileptic seizure

- So our aim here will to keep the output value of y=1 as it is and convert all remaining values i.e 2,3,4,5 to 0

In [7]:
```
df["OUTPUT_LABEL"] = df.y == 1
df["OUTPUT_LABEL"] = df["OUTPUT_LABEL"].astype(int)
df.pop('y')
df.drop(df.columns[0], axis=1, inplace=True)
```

In [8]:
```
df.head()
```

Out[8]:

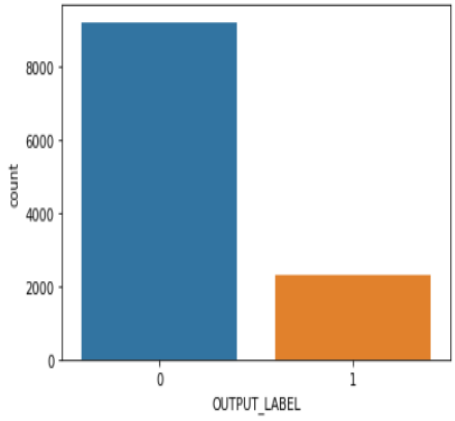
	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	OUTPUT_LABEL
0	135	190	229	223	192	125	55	-9	-33	-38	...	-17	-15	-31	-77	-103	-127	-116	-83	-51	0
1	386	382	356	331	320	315	307	272	244	232	...	164	150	146	152	157	156	154	143	129	1
2	-32	-39	-47	-37	-32	-36	-57	-73	-85	-94	...	57	64	48	19	-12	-30	-35	-35	-36	0
3	-105	-101	-96	-92	-89	-95	-102	-100	-87	-79	...	-82	-81	-80	-77	-85	-77	-72	-69	-65	0
4	-9	-65	-98	-102	-78	-48	-16	0	-21	-59	...	4	2	-12	-32	-41	-65	-83	-89	-73	0

5 rows × 179 columns

Plotting the count of dependent variable

In [9]:
```
sns.countplot(x='OUTPUT_LABEL',data=df)
```

Out[9]:
```
<AxesSubplot:xlabel='OUTPUT_LABEL', ylabel='count'>
```

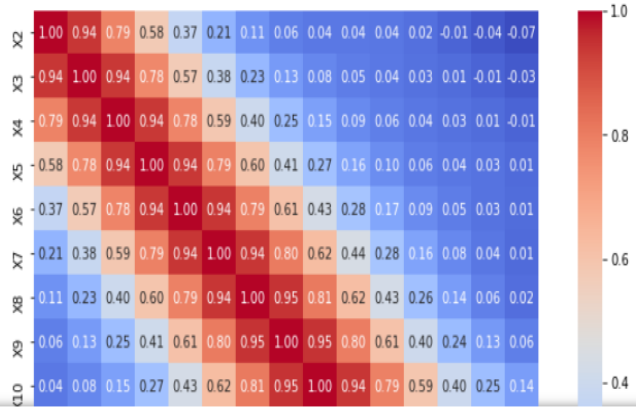


- From the above plot we can say that around 20% people are having a seizure present and rest are seizure free

In [12]:
```
# Considering the first 15 feature cloumn
sel_features= list(df.columns[1:16])
print(sel_features)
corr = df[sel_features].corr()
plt.figure(figsize=(12,8))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt= '.2f',
            xticklabels= sel_features, yticklabels= sel_features,
            cmap= 'coolwarm')
```

```
['X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'X12', 'X13', 'X14', 'X15', 'X16']
```

Out[12]: <AxesSubplot:>



Classification of Epileptical Seizures from EEG signals using Deep Learning

In [84]:
```
import tensorflow as tf
import numpy as np
from keras.models import Sequential
from keras.utils import np_utils
from keras.layers import Dense, Activation , LSTM , Dropout , AveragePooling3D
import pandas as pd

import matplotlib.pyplot as plt
```

Data Preprocessing

In [85]:
```
df=pd.read_csv('data.csv')
df.head()
```

Out[85]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	y
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31	-77	-103	-127	-116	-83	-51	4
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146	152	157	156	154	143	129	1
2	X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	...	57	64	48	19	-12	-30	-35	-35	-36	5
3	X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	...	-82	-81	-80	-77	-85	-77	-72	-69	-65	5
4	X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	...	4	2	-12	-32	-41	-65	-83	-89	-73	5

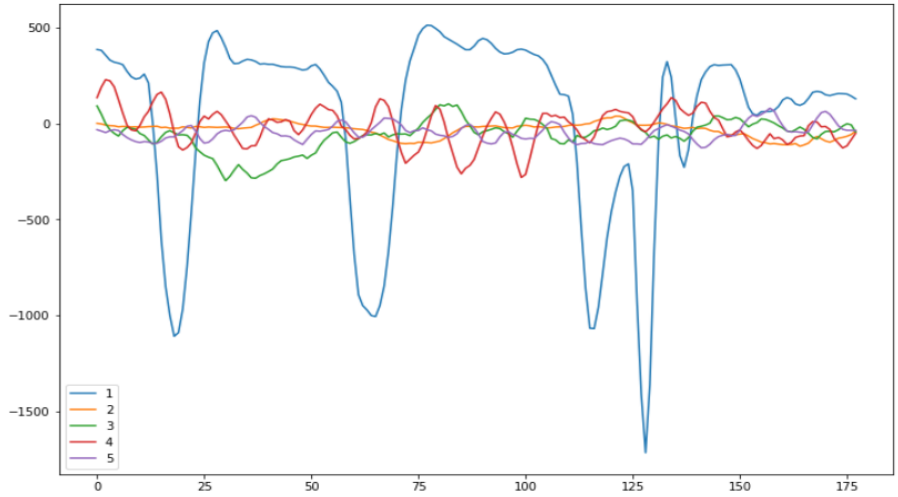
5 rows × 180 columns

In [86]:
```
X=df.values
```

In [87]:
```
X=X[:,1:-1]
```

Intuition of the Epileptical Signal In the Dataset

In [90]:
```
plt.figure(figsize=(12,8))
plt.plot(X[1,:],label='1')
plt.plot(X[7,:],label='2')
plt.plot(X[12,:],label='3')
plt.plot(X[0,:],label='4')
plt.plot(X[2,:],label='5')
plt.legend()
plt.show()
```



Creating LSTM Neural Network For the Classification

In [91]:
```
model = Sequential()
model.add(LSTM(56, input_shape=(45,1), return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(56))
model.add(Dropout(0.3))
model.add(Dense(20))
model.add(Activation('tanh'))
model.add(Dense(5))
model.add(Activation('softmax'))

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_8 (LSTM)                (None, 45, 56)            12992
_________________________________________________________________
dropout_8 (Dropout)          (None, 45, 56)            0
_________________________________________________________________
lstm_9 (LSTM)                (None, 56)                25312
_________________________________________________________________
dropout_9 (Dropout)          (None, 56)                0
_________________________________________________________________
dense_8 (Dense)              (None, 20)                1140
_________________________________________________________________
activation_8 (Activation)    (None, 20)                0
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 105
_________________________________________________________________
activation_9 (Activation)    (None, 5)                 0
=================================================================
Total params: 39,549
Trainable params: 39,549
Non-trainable params: 0
_________________________________________________________________
```

In [92]:
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [93]:
```
hist = model.fit(((X_train[:,::4]-X_train.mean())/X_train.std()),Y_train[:,1:],validation_data=((X_test[:,::4]-X_test.mean())/X_t
                 ,epochs = 50, batch_size=15,shuffle=False
                 )
```

```
Epoch 1/50
614/614 [==============================] - 43s 61ms/step - loss: 0.4273 - accuracy: 0.3686 - val_loss: 0.3855 - val_accuracy:
0.4565
Epoch 2/50
614/614 [==============================] - 33s 53ms/step - loss: 0.3806 - accuracy: 0.4461 - val_loss: 0.3756 - val_accuracy:
0.4717
Epoch 3/50
614/614 [==============================] - 33s 54ms/step - loss: 0.3719 - accuracy: 0.4657 - val_loss: 0.3715 - val_accuracy:
0.4843
Epoch 4/50
614/614 [==============================] - 34s 55ms/step - loss: 0.3354 - accuracy: 0.5357 - val_loss: 0.2785 - val_accuracy:
0.6452
Epoch 5/50
614/614 [==============================] - 34s 55ms/step - loss: 0.2792 - accuracy: 0.6293 - val_loss: 0.2561 - val_accuracy:
0.6717
Epoch 6/50
614/614 [==============================] - 34s 55ms/step - loss: 0.2572 - accuracy: 0.6601 - val_loss: 0.2496 - val_accuracy:
0.6722
Epoch 7/50
```

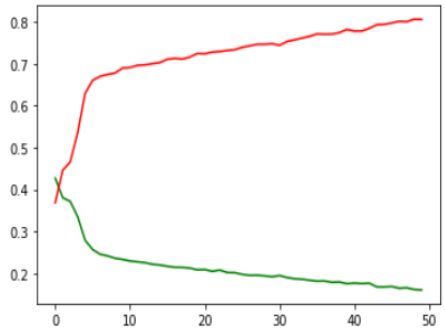
Saving the trained model

In [95]:
```
model.save('Epilepsy.h5')
```

Visualization

- Graph Between Training Loss and Training Accuracy

In [97]:
```
plt.figure(0)
plt.plot(hist.history['loss'],'g')
plt.plot(hist.history['accuracy'],'r')
plt.show()
```



Loading The Trained Model

In [2]:
```
model = load_model('Epilepsy.h5')
```

Data handling

In [3]:
```
df=pd.read_csv('data.csv')
df.head()
```

Out[3]:

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X170	X171	X172	X173	X174	X175	X176	X177	X178	y
0	X21.V1.791	135	190	229	223	192	125	55	-9	-33	...	-17	-15	-31	-77	-103	-127	-116	-83	-51	4
1	X15.V1.924	386	382	356	331	320	315	307	272	244	...	164	150	146	152	157	156	154	143	129	1
2	X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	...	57	64	48	19	-12	-30	-35	-35	-36	5
3	X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	...	-82	-81	-80	-77	-85	-77	-72	-69	-65	5
4	X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	...	4	2	-12	-32	-41	-65	-83	-89	-73	5

5 rows × 180 columns

In [5]:
```
X=df.values
```

In [6]:
```
X=X[:,1:-1]
```

In [7]:
```
from sklearn.model_selection import train_test_split


y=np.array(df['y'])
Y=np_utils.to_categorical(y)
Y.shape
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=1)
```

Accuracy on Validation Dataset

In [15]:
```
ypred=model.predict((X_test[:,::4]-X_test.mean())/X_test.std())
ypred.shape
```

Out[15]: (2300, 5)

In [16]:
```
yp=np.zeros((Y_test.shape[0]))
yo=np.ones((Y_test.shape[0]))
```

In [17]:
```
for i in range(Y_test.shape[0]):
    yp[i]=np.argmax(ypred[i])+1
    yo[i]=np.argmax(Y_test[i])
```

In [18]:
```
yp.shape
```

Out[18]: (2300,)

In [19]:
```
yo.shape
```

Out[19]: (2300,)

In [20]:
```
np.unique(yo)
```

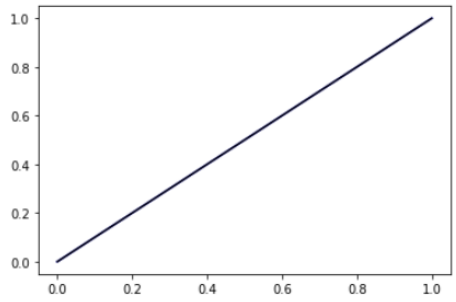
Out[20]: array([1., 2., 3., 4., 5.])

In [21]:
```
np.unique(yp)
```

Out[21]: array([1., 2., 3., 4., 5.])

In [22]:
```
#conversion of classes
for i in range(Y_test.shape[0]):
    if yo[i]!=1:
        yo[i]=0
    if yp[i]!=1:
        yp[i]=0
```

```
In [23]: np.unique(yo)
Out[23]: array([0., 1.])

In [24]: np.unique(yp)
Out[24]: array([0., 1.])

In [25]: from sklearn.metrics import accuracy_score

In [26]: accuracy_score(yo,yp)
Out[26]: 0.9908695652173913

In [27]: plt.plot(np.unique(yo),'b')
         plt.plot(np.unique(yp), 'black')


         plt.show()
```



Accuracy on Training Dataset

```
In [28]: ypred1=model.predict((X_train[:,::4]-X_train.mean())/X_train.std())
         ypred1.shape
Out[28]: (9200, 5)

In [29]: yp1=np.zeros((Y_train.shape[0]))
         yo1=np.ones((Y_train.shape[0]))

In [30]: for i in range(Y_train.shape[0]):
             yp1[i]=np.argmax(ypred1[i])+1
             yo1[i]=np.argmax(Y_train[i])

In [31]: yp1.shape
Out[31]: (9200,)

In [32]: yo1.shape
Out[32]: (9200,)

In [33]: np.unique(yo1)
Out[33]: array([1., 2., 3., 4., 5.])

In [34]: np.unique(yp1)
Out[34]: array([1., 2., 3., 4., 5.])

In [35]: #conversion of classes
         for i in range(Y_train.shape[0]):
             if yo1[i]!=1:
                 yo1[i]=0
             if yp1[i]!=1:
                 yp1[i]=0

In [36]: np.unique(yo1)
```

```
In [36]: np.unique(yo1)
Out[36]: array([0., 1.])

In [37]: np.unique(yp1)
Out[37]: array([0., 1.])

In [38]: accuracy_score(yo1,yp1)
Out[38]: 0.9989130434782608

In [ ]:
```

4.2 Discussions:

A. Machine Learning Based Model

Classification Models we were considering for Machine Learning based Approach:

- K Nearest Neighbor
- Logistic Regression
- Support Vector Machine
- Random Forest
- Extreme Gradient Boosting(Xgboost)

But from all the above mentioned algorithms we prefer using KNN because of its accuracy and easy implementation.

Why use KNN?

- Quick Calculation time
- Simple algorithm to implement
- Versatile - Useful for classification as well as regression
- High accuracy - don't need to compare with other Supervised Learning Algorithms
- No assumption about data - no need to tune several parameters, or build several models. This makes it crucial in a non-linear dataset.

B. Deep Learning Based Model

Classification Models we were considering for Deep Learning based Approach:

- LSTM (Long Short Term Memory)
- CNN (Convolutional Neural Network)
- ANN (Artificial Neural Network)

But from all the above mentioned algorithms we prefer using LSTM because of its accuracy and easy implementation.

Why use LSTM ?

- Quick Calculation time
- Simple algorithm to implement
- More parameters and a system of gating units that control the flow of information
- Higher Accuracy in prediction

4.3 Discussion-Comparative study/Analysis

MODEL	ACCURACY	PRECISION	RECALL
KNN	0.9263	0.92	0.64
LSTM	0.99	0.97	0.97

Chapter 5

Conclusion

We have conducted a comparative study of different machine learning techniques and deep learning techniques to identify the best machine learning-deep learning algorithm identifying the best performing epileptic seizures prediction method.
From our analysis of 2 algorithms - KNN (Machine Learning Model) and LSTM (Deep Learning Model), we had concluded that LSTM is the best deep neural network among Artificial Neural Network, Convolutional Neural Network, and Autoencoders. LSTM has a memory cell, and it avoids long-term dependency problems. LSTM is the best model for seizure prediction of big volume time-series big data. It can achieve high sensitivity and specificity when compared with other models. However, there is still an opportunity to get better in numerous perspectives. In the future, we can incorporate ensemble learning in seizure prediction by combining LSTM with other efficient models. Other than that future research can be done to reduce the number of parameters earned in Deep Learning Models.

References

[1] Falco-Walter, Jessica J., Ingrid E. Scheffer, and Robert S. Fisher. "The new definition and classification of seizures and epilepsy."

[2]Ullah, Ihsan, Muhammad Hussain, and Hatim Aboalsamh. "An automated system for epilepsy detection using EEG brain signals based on a deep learning approach."

[3]Usman, Syed Muhammad, Muhammad Usman, and Simon Fong.

[4] "Epileptic seizures prediction using Deep Learning methods." Zhou, Mengni, Cheng Tian, Rui Cao, Bin Wang, Yan Niu, Ting Hu, Hao Guo, and Jie Xiang. "Epileptic Seizure Detection Based on EEG Signals and CNN." Frontiers in neuroinformatics 12 (2018): 95.

[5] Cao, Yuzhen, Yixiang Guo, Hui Yu, and Xuyao Yu. "Epileptic seizure auto-detection using a deep learning method." In 2017 4th

Acknowledgement

We wish to express our sincere gratitude to **Dr. Sanjay U. Bokade, Principal** and **Prof. S. P. Khachane , H.O.D.** of Department Computer Engineering of Rajiv Gandhi Institute of Technology for providing us an opportunity to do our project work on **"Epilepsy Seizure Prediction "**.

This project bears the imprint of many peoples. We sincerely thank our project guide Prof**. Preeti Satao** for his/her guidance and encouragement in carrying out this synopsis work.

Finally, we would like to thank our colleagues and friends who helped us in completing project work successfully

1. Khush Dave (A-824)
2. Ajay H. Desai (A-824)
3. Shubham M. Hedavkar (A-838)