# Report

# Epileptic Seizure Dataset

Submitted by Group 16

For

CSN-382 Machine Learning

Instructor

Dr. Partha Pratim Roy


Members

| | |
|---|---|
| Ankur Parihar | 16114015 |
| Parvat Yadar | 16114043 |
| Rishikesh Chaudhary | 16114054 |
| Sagar Dhurwe | 16114059 |

# Objective

Classification of [epileptic seizure dataset]

# Data Overview

The original dataset from the reference consists of 5 different folders, each with 100 files, with each file representing a single subject/person. Each file is a recording of brain activity for 23.6 seconds. The corresponding time-series is sampled into 4097 data points. Each data point is the value of the EEG recording at a different point in time. So we have total 500 individuals with each has 4097 data points for 23.5 seconds.

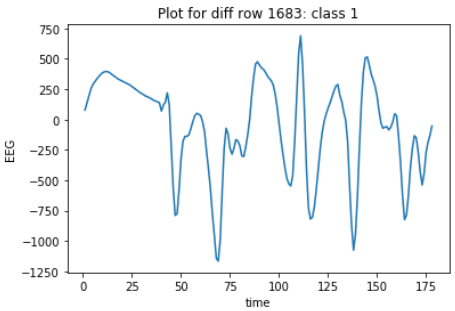
The data is divided and shuffled every 4097 data points into 23 chunks, each chunk contains 178 data points for 1 second, and each data point is the value of the EEG recording at a different point in time. So now we have 23 x 500 = 11500 pieces of information (row), each information contains 178 data points for 1 second (column), the last column represents the label y $\{1, 2, 3, 4, 5\}$.

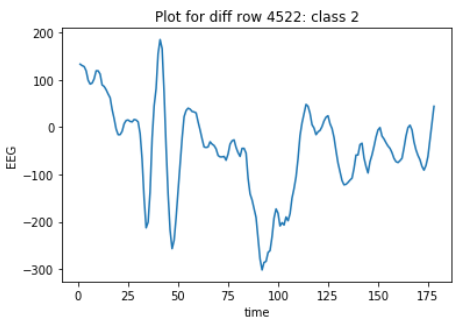
The response variable is y in column 179, the Explanatory variables $X_1, X_2, \ldots, X_{178}$. $y$ contains the category of the 178-dimensional input vector. Specifically y in $\{1, 2, 3, 4, 5\}$:

1. Recording of seizure activity
2. They recorder the EEG from the area where the tumour was located
3. Yes they identify where the region of the tumour was in the brain and recording the EEG activity from the healthy brain area
4. eyes closed, means when they were recording the EEG signal the patient had their eyes closed
5. eyes open, means when they were recording the EEG signal of the brain the patient had their eyes open

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. The motivation for creating this version of the data was to simplify access to the data via the creation of a .csv version of it. Although there are 5 classes most authors have done binary classification, namely class 1 (Epileptic seizure) against the rest.

For other details check out
https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition



# Our Approach

We solve the above problem in two steps. First steps involves transforming the data to a more suitable form and step 2 operates upon the transformed data.

## Stage 1: pulseNet

We developed a technique specially designed for pulse detection in waves. We call it **pulseNet**. pulseNet is an extension and can be used to transform data.

We start parsing the wave data point by point. We define an upper limit and a lower limit. The average of two data points is denoted and used to determine if the data is over limit or under limit. And we define categories the data can lie. In our case it is 4 categories. The categories are as follows:

$$avg = \frac{(x_i + x_{i+1})}{2}$$

$$if\ (avg > limit_{upper}\ or\ avg < limit_{lower}\ )\ limit = under\ else\ limit = over$$

0 – Increment but under limit

1 – Decrement but under limit

2 – Increment but over limit

3 – Decrement but over limit

Now analysing two data points gives us corresponding category. Then we move one step forward and do the same. Repeat until the whole wave is parsed.

```
        Y = []
        rows = len(X)
        cols = len(X[0])
        for i in range(rows):
            small_Y = []
            for j in range(cols-1):
                avg = (X[i][j] + X[i][j+1])/2
                if (avg > self.pos_lim) or (avg < self.neg_lim):
                    over = 1
                else:
                    over = 0
                if(X[i][j+1] >= X[i][j]):
                    small_Y.append(over*2 + 0)
                else:
                    small_Y.append(over*2 + 1)
            Y.append(small_Y)
        self.Y = Y
```

Now we have a wave containing only 4 values and length is one less than original.

In the next step we define a length of portion of wave to analyse. Based on the length we create an empty square matrix initialized with 0's. Side length of matrix is calculated as follows:

$$side = (n_{cat})^{pulselen/2}$$

Where:

$pulselen$ = Length of portion

$n_{cat}$ = no. of categories

Side must be an integer and since there is a square root operation $\boldsymbol{n_{cat}}$ **must be a whole squared integer**.

After initializing the matrix we start to parse the transformed wave. We select data points equal to size of $pulselen$ and plot it on matrix. The selected data is called the **address** and is used to determine location on matrix grid. The plotting function is core of this method

The matrix addresses are denoted in a block and level scheme. Largest or upper level blocks are the most significant digit of the address and lowest or smallest level blocks denote the least significant digit of the address.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 002 | 003 | 012 | 013 | 102 | 103 | 112 | 113 |
| 020 | 021 | 030 | 031 | 120 | 121 | 130 | 131 |
| 022 | 023 | 032 | 033 | 122 | 123 | 132 | 133 |
| 200 | 201 | 210 | 211 | 300 | 301 | 310 | 311 |
| 202 | 203 | 212 | 213 | 302 | 303 | 312 | 313 |
| 220 | 221 | 230 | 231 | 320 | 321 | 330 | 331 |
| 222 | 223 | 232 | 233 | 322 | 323 | 332 | 333 |

3-address matrix

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| 002 | 003 | 012 | 013 | 102 | 103 | 112 | 113 |
| 020 | 021 | 030 | 031 | 120 | 121 | 130 | 131 |
| 022 | 023 | 032 | 033 | 122 | 123 | 132 | 133 |
| 200 | 201 | 210 | 211 | 300 | 301 | 310 | 311 |
| 202 | 203 | 212 | 213 | 302 | 303 | 312 | 313 |
| 220 | 221 | 230 | 231 | 320 | 321 | 330 | 331 |
| 222 | 223 | 232 | 233 | 322 | 323 | 332 | 333 |

address of pulse 301

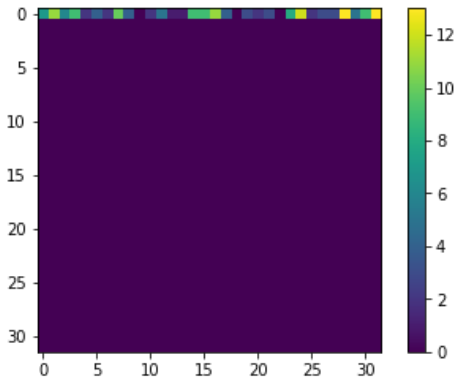
The plotting function starts by analysing the first member of address and jumps to specific location or block. For example the currently the complete matrix is on focus. Suppose the data is 2 (out of 4 categories) than we jump to the block denoted by 2. Not this block is in focus and the scope is reduced. Repeat this until the pulse is parsed. After parsing the complete address we have the location corresponding the address. We then increase the value by unit of that location. Now we move one step ahead and repeat the same process until the whole wave is exhausted.

```
def pulses_to_matrix(self):
        """
        Return list of numpy 2d matrix with pulse plots
        """
        Y = self.Y
        self.final_mat = []
        divisor = math.sqrt(self.categories)
        for i in range(len(Y)):
            self.init_mat()
            for j in range(len(Y[0])-self.pulse_len+1):
                side = self.side / 2
                row = 0
                col = 0
                for k in range(self.pulse_len):
                    # start coord of ith row is j+k
                    row += (Y[i][j+k] //2) * side
                    col += (Y[i][j+k] % 2) * side
                    side = side // divisor
                self.matrix[int(row)][int(col)] += 1
            self.final_mat.append(self.matrix)
        return self.final_mat
```

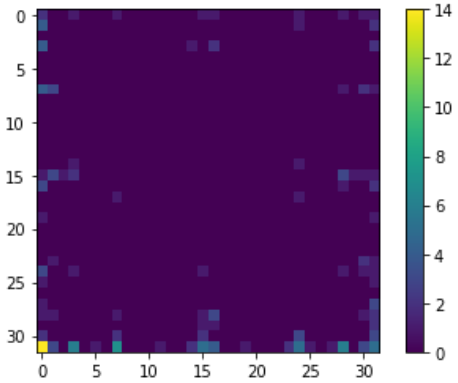
So in this way we apply fractal approach to prepare the final matrix. This is done to all the transformed waves and finally a list of wave is returned.

Now we have a list of 2D matrices made from wave data. We can export them to images or use directly in the next stage.

Some example of 2D image graphs generated by pulseNet ($pulselen = 5$ ,$n_{cat} = 4$)




Class: False

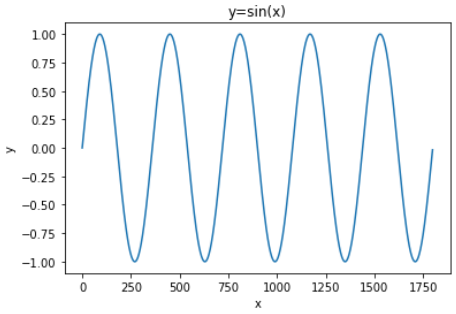


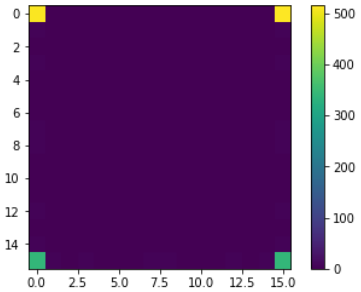

Class: True

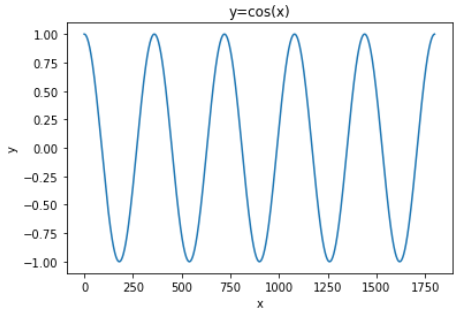
## Benefits from pulseNet:

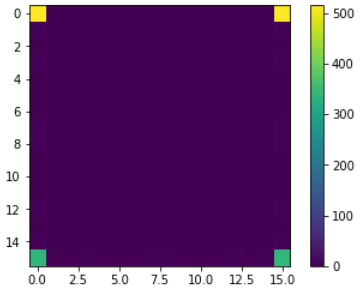
1. Convert large wave into smaller data (image or matrix) without significant reduction in important data.
   No matter how long the pulse is we are only looking for slopes or similar properties in it and store in definite matrix which can be feed to deep learning networks.

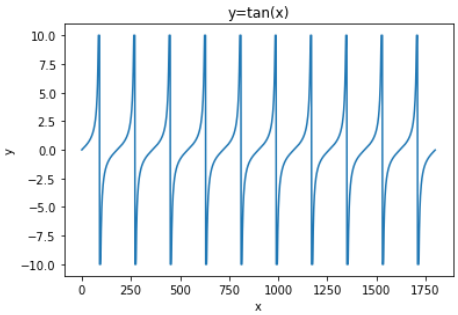
2. Behaves same with shifted waves. For example, it does not differentiate between sin wave and cos wave but effectively differentiate between sin wave and tan wave.

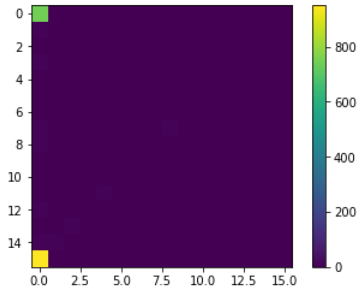




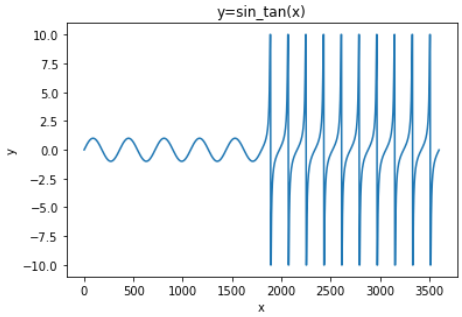



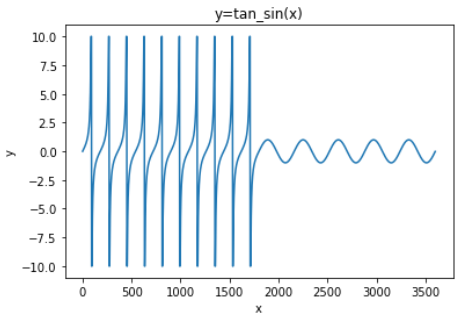


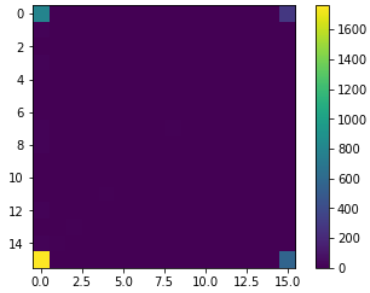



3. Stores every sub-wave of length *pulselen* effectively and efficiently.
Each sub wave of pulse have some unique signature or address it they differ and similar pulses have same signature. To store each pulse of a type we need only one unit space. A specific unit of matrix contains the number of pulses with that signature.

4. Pulse order in wave does not matter. But pulse count does matter. Good for independent pulses.

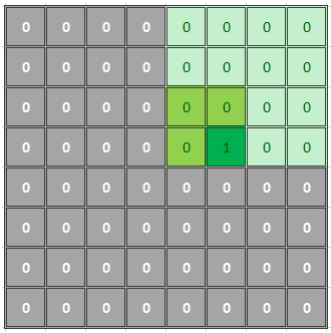


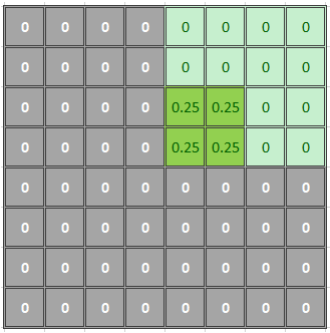


Same plot for both pulses

5. Focusing ability – We can choose to increment one pixel or a set of pixel depending on level of detail. For example while parsing pulses we decide to ignore the effect of last data then we can increase all 4 final blocks ($n_{cat} = 4$) by 0.25 or more general way we can increase all $n_{cat}$ final points by $1/n_{cat}$



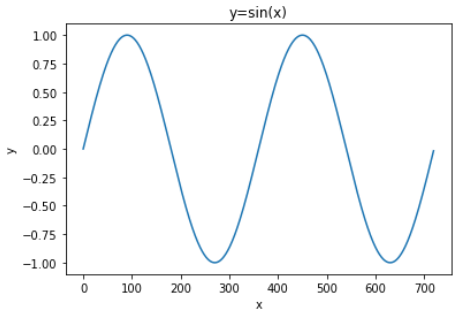
Increase 1 cell (sharp)

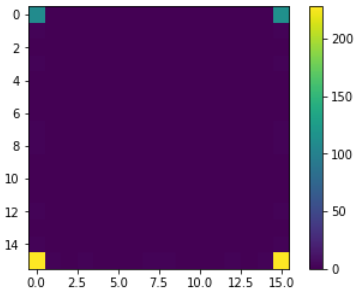


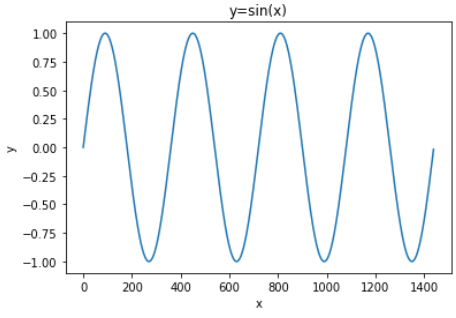
Increase 4 cells (blur)

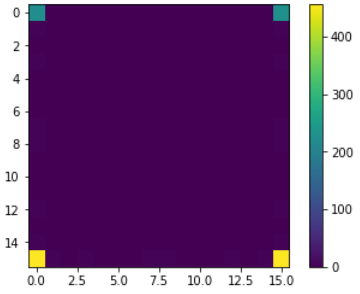
6. Supports functional limits. Instead of providing fixed limits, we can also use functions to determine limiting situations.

7. A repeated shape can be seen as a layer. When the shape repeats the layer simple doubles.





## Stage 2: CNN classification

In this step we feed the exported images from previous step into a tensor-flow CNN image classifier and get the accuracy of model.

## Configuration and Parameters

```
# Convolutional Layer 1.
filter_size1 = 3
num_filters1 = 32

# Convolutional Layer 2.
filter_size2 = 3
num_filters2 = 32

# Convolutional Layer 3.
filter_size3 = 3
num_filters3 = 64

# Fully-connected layer.
fc_size = 128             # Number of neurons in fully-connected layer.

# Number of color channels for the images: 1 channel for gray-scale.
num_channels = 3

# validation split
validation_size = .20
```

Convolution layer 1

```
<tf.Tensor 'Relu:0' shape=(?, 8, 8, 32) dtype=float32>
```

Convolution layer 2

```
<tf.Tensor 'Relu_1:0' shape=(?, 4, 4, 32) dtype=float32>
```

Convolution layer 3

```
<tf.Tensor 'Relu_2:0' shape=(?, 2, 2, 64) dtype=float32>
```

Flatten layer

```
<tf.Tensor 'Reshape_1:0' shape=(?, 256) dtype=float32>
```

Num features = 256

Fully connected layer 1

```
<tf.Tensor 'Relu_3:0' shape=(?, 128) dtype=float32>
```

Fully connected layer 2

```
<tf.Tensor 'add_4:0' shape=(?, 2) dtype=float32>
```

## Pooling

Image resolution was down-sampled using 2x2 max-pooling followed by ReLU operation.

## Predicted class

The predicted class function uses softmax to normalize and argmax for estimation.

## Cost function

Using TensorFlow's in-built cross-entropy function. It internally calculates the softmax. In order to use the cross entropy to guide the optimization of the model's variables we need a single scaler value, so we simply take the average of the cross entropy for all the image classifications using `reduce_mean`.

## Optimization method

We use TensorFlow training library's `AdamOptimizer`, which is an advanced form of Gradient Descent.

## Training

The model was trained with $pulselen = 4, 5, 6$. We trained the model for multiple classification as well as binary classification for class 1 against all other.

The testing accuracy we get for all cases after 10,000 iterations is listed below:

| Class \ $pulselen$ | 4 | 5 | 6 |
|---|---|---|---|
| Binary | 95.0% | 96.0% | 95.8% |
| Multiclass | 66.4% | 65.0% | 68.8% |