Prediction and detection of epileptic seizures

Aprendizagem Computacional / Machine Learning (MEI)

Computação Neuronal e Sistemas Difusos /
Neural Computation and Fuzzy Systems (MIEB)

André Correia (MEI) – 2020158188

Ennio Malvati (MEI) – 2020187992

Departamento de Engenharia Informática

Faculdade de Ciências e Tecnologia

Universidade de Coimbra

November 2020

Contents

Introduction

It is proposed, with this practical assignment, to develop neuronal networks, CNN and LSTM in order to predict and detect epileptic seizures.

The possibility to predict or to detect a seizure, based on the information from brain signals, principally the EEG (Electroencephalogram), is an actual and challenging research problem. Seizure prediction would allow the patients to act for their own safety.

An electroencephalogram (EEG), which involves applying electrodes to the patient's head. Doctors use this in the presence or suspicion of neurological diseases, capable of altering the conduction of electrical signals within the brain. Data collected by EEG so is a set of electrical signals, in scale of microvolts.

It is supposed that many neurologic information is embedded in the EEG signals. To predict and detect a seizure, one needs to develop signal analysis methods able to detect patterns in the EEG typical of a preictal or of an ictal state, respectively. Where preictal state are the moments before the epileptic seizure and ictal state during seizure.

1. Dataset

In the present work the set of features used exploit the different frequency bands of the electrical signals, and have been extracted by Doctor Mojtaba Bandarabadi, during his PhD studies in our Department. They refer to one EEG invasive channel placed in the epileptic focus (the place in the brain where the seizure starts). The features have been extracted using 2 seconds EEG segments with 50% superposition between segments (i.e., consecutive time-windows share half of the information). In practical terms every second a vector of 29 features is obtained. These 29 features are characteristic of the frequency spectrum of the EEG channel (see at the end of this document). For the complete register of a patient, a high number of vectors is obtained, and the objective of the present work is to classify them among the defined three classes, principally to the pre-ictal and ictal classes.
For the prediction and detection of seizures, the brain state will be classified into three states. A three output NN is needed, one output for each class.
1- Class **interictal**, normal brain state.
2- Class **Preictal**, a seizure is coming.
3- Class **Ictal,** a seizure is happening.
Where the postictal phase is considered part of the interictal.

In this assignment we had to use datasets, with the structure describe before, from two different patients. The data from patient `54802` contains 31 seizures, while data from patient `112502` contains 14 seizures.

The supplied datasets have the following variables inside:

- FeatVectSel, is the features matrix where each column represents an extracted feature, and each row a vector of features. So, this matrix must be transposed to make the P matrix defined in class (one column is a vector of features).
- Trg, a column matrix (it is a vector) that identifies the seizures. It is made with 0 and 1. The value *0* means non-ictal class (*so interictal, preictal, or postictal*), and the value 1 ictal. The points of the preictal are easily defined: 900 points before the first 1 for each seizure, corresponding to 900 seconds=15 minutes. Similarly, the postictal points are those 300 after the last 1 of each seizure (5 minutes).

2. Pre-Processing

For pre-processing of dataset, we made a MATLAB function: `loadDataset.m`. The purpose of this function is to define matrices *P* and *T*, from the given dataset, to be used in the training and the test of the various neural networks.

Matrix P is made up from the transposed variable `FeatVectSel`, while the definition of the T matrix is more complex.

As described in the previous paragraph `Trg`, is a column matrix that identifies the seizures. It is made with 0 and 1. The value *0* means non-ictal class (*so interictal, preictal, or postictal*), and the value 1 ictal. So, from this variable we create T matrix, with the following pattern:

- First row indicates if a point is interictal,
- Second row indicates if a point is preictal,
- Third row indicates if a point is ictal.

All postictal points were considered as interictal points, so all *300* points after each *1* value, which indicates an epileptic seizure in Trg, were regarded as interictal.

In the present work, a seizure may last in average 2 minutes, while the average registered time per patient is 148 hours. It means that the number of data points for the ictal class is much lower than the other classes, specially the interictal class that contains more than 90% of the points. Without some precaution, we will have catastrophic result from the point of view of the aim of the problem, because NN classifies well all the interictal points and classifies wrongly most of the ictal (or preictal) points. So, we decided to implement the possibility of using class balancing approach, where in this case the number of interictal class instances is equal to the sum of all other classes. Note that this approach is used just for *training and validation set and not in test set*.
So, for matrix P and T we decided to use for test set 25% of the number of samples in the dataset and for training 75%.

Moreover we tried to give more importance to the ictal (for detection) or preictal (for prediction) instants using different weights (penalization) of the error of the NN in the different instants. We did it for multilayer (with delays and without delays) networks as follow:

```
weight_penalty = ones(1, length(t_matrix));

interictal_idxs = find(t_matrix(1,:) == 1);
interictal_total = length(interictal_idxs);
preictal_idxs = find(t_matrix(2,:) == 1);
preictal_total = length(preictal_idxs);
ictal_idxs = find(t_matrix(3,:) == 1);
ictal_total = length(ictal_idxs);

if strcmp(type, 'prediction')
    error = (interictal_total / preictal_total) * 2;
    weight_penalty(preictal_idxs) = error;
elseif strcmp (type, 'detection')
    error = (interictal_total / ictal_total) * 18;
    weight_penalty(ictal_idxs) = error;
end
```

This part of code is implemented in `train_NN.m`, where the weight penalty to the net before training:

```
net = train(net, p_matrix, t_matrix, [], [], weight_penalty);
```

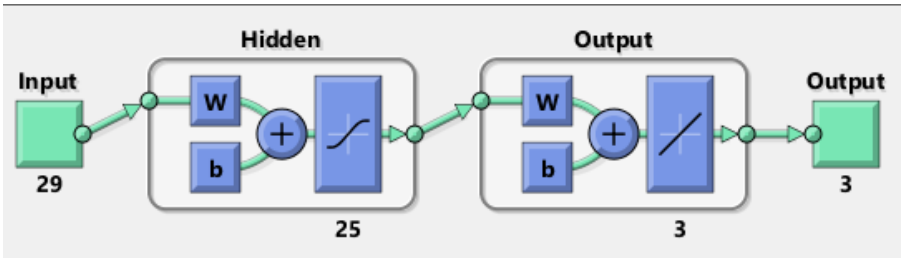
3. Network architecture

Five models were created in order to fulfil the proposed objective. After some testing with different training algorithms, the final chosen architectures were the following:

- For **multilayer network** we used a **feedforward** network with *25 neurons* in the hidden layer, with *logsig* as activation functions in the first layer and *purelin* as activation function in the second layer, Levenberg- Marquardt as the training algorithm and gradient descent as the learning algorithm, this learning function is default for feedforward net, and the default training method (Batch). For training, only training (80%) and validation (20%) sets were used:

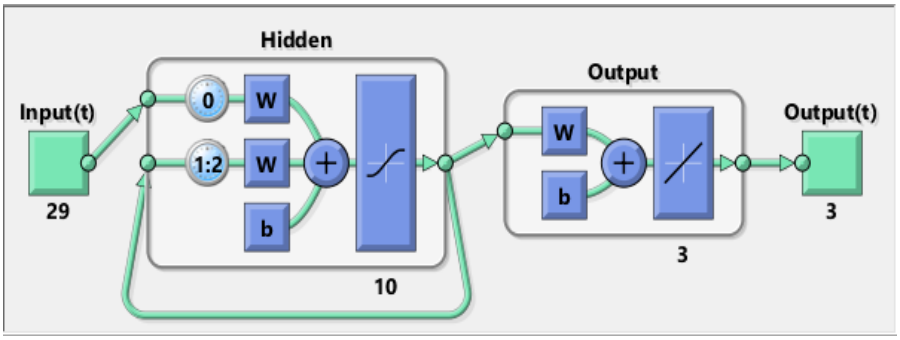
```
net = feedforwardnet(25);

net.divideFcn = 'divideblock';
net.trainFcn = 'trainlm';
net.trainParam.epochs = 1000;
net.trainParam.max_fail = 5000;

% Train division
net.divideParam.trainRatio = 0.80;
net.divideParam.valRatio = 0.20;
net.divideParam.testRatio = 0.0;
```

Remark: feedforward network, in MATLAB, has two default layers.



- For **multilayer network with delays** we have decided to use **layer recurrent network** with *two delays*, 1 and 2, with *10 neurons in hidden layer*. For the first layer and second layer we used the default activation functions, in the first layer *logsig* and *purelin* for the second layer. As training algorithm, we used Levenberg- Marquardt (batch).

```
net = layrecnet(1:2,10);
net.divideFcn = 'divideind';
net.trainFcn = 'trainlm';
net.divideParam.trainInd,~,~] =
divideind(length(p_matrix),1:length(p_matrix),[],[]
);
net.trainParam.epochs = 1000;
```



- For the **Convolutional Neural Network**, CNN, we used the following layers:

```
layers = [
            imageInputLayer([29 29 1])
            convolution2dLayer(5,20)
            reluLayer()
            maxPooling2dLayer(2,'Stride',2)
            fullyConnectedLayer(3)
            softmaxLayer
            classificationLayer
        ];
```

-**imageInputLayer** is a layer that takes *29 x 29* images with a single channel as input and applies data normalization.

-The second layer applies sliding convolutional filters to the input. We decided to use *20* filters with *5x5* dimension.

-The third layer is a **Rectified Linear Unit layer** (reLU) performs a threshold operation to each element of the input, the convoluted output from the previous layer, where any value less than zero is set to zero. This operation is equivalent to:

$$\underset{\text{ReLU}}{\sigma(x)} = \begin{cases} x, x \geq 0 \\ 0, x \prec 0 \end{cases}$$

It's quite useful in this case because the input is a set of image in grey scale for this reason the data are the pixels' intensity in [0 1] and it does not make much sense that the feature maps have negative values.

- Fourth layer is a **max pooling layer** which down-samples the output of the previous layer dividing it in rectangular pooling region, and compute the maximum of each region, with the following dimension 2x2 using a [2,2] stride.

-Fifth layer is a **fully connected layer**, where the output size is equal to the number of classes of the dataset, in this case we use 3 as the number of classes (interictal, preictal and ictal). It has purlin activation function, a set of weight and bias.

-The sixth layer is a **softmax layer**, which applies a softmax function to the input.

-Last layer is a classification layer, which takes as inputs the outputs of the softmax function, of the previous layer, and assigns each input to one of the *3* mutually exclusive classes minimizing the default cross entropy function.

The chosen training option we used is the following:

```
options = trainingOptions('sgdm', ...
        'MaxEpochs',1000, ...
        'InitialLearnRate',1e-4, ...
        'Shuffle','every-epoch', ...
        'Plots','training-progress', ...
        'Verbose',false);
```

So, we decided to use as *learning rate 1e-4* and as training algorithm *gradient descent learning algorithm.*
For the CNN network the inputs were made by creating a 4D array out of our features array. That 4D array (29x29x1xN where N = number of pictures) would represent 2D images with a single channel to be fed to the CNN network

- For the **Long Short-Term Memory** network, LSTM, we used the following layers:

```
numFeatures = 29;
numHiddenUnits = 75;
numClasses = 3;

layers = [
sequenceInputLayer(numFeatures)

lstmLayer(numHiddenUnits,'OutputMode','last')
fullyConnectedLayer(numClasses)
softmaxLayer
classificationLayer];
```

-The first layer, `sequenceInputLayer`, creates a sequence input layer and sets as input size 29, because the number of features we have.

-Second layer is **LSTM** and we used 75 as number of hidden units.

-Third layer is a **fully connected layer,** we use it with 3 classes (interictal, preictal and ictal).

-**softmaxLayer** and **classificationLayer** have the same structure of corresponding CNN's layers.

The chosen training option we used is the following:

```
options = trainingOptions('sgdm', …
        'ExecutionEnvironment','auto', ...
        'GradientThreshold',1, ...
        'MaxEpochs',1000, ...
        'MiniBatchSize',29, ...
        'SequenceLength','longest', ...
        'Shuffle','never', ...
        'Verbose',0, ...
        'Plots','training-progress');
```

We choose to use `sgdm`, Stochastic gradient descent with momentum, to train the network.

For the LSTM network the input were made in this way:

```
p_matrix = num2cell(p_matrix,1);
T_train= zeros(1, length(t_matrix));
T_train(find(t_matrix(1,:) == 1)) = 1;
T_train(find(t_matrix(2,:) == 1)) = 2;
T_train(find(t_matrix(3,:) == 1)) = 3;
T_train = categorical(T_train)';
```

- For the autoencoders classification, the following was performed:

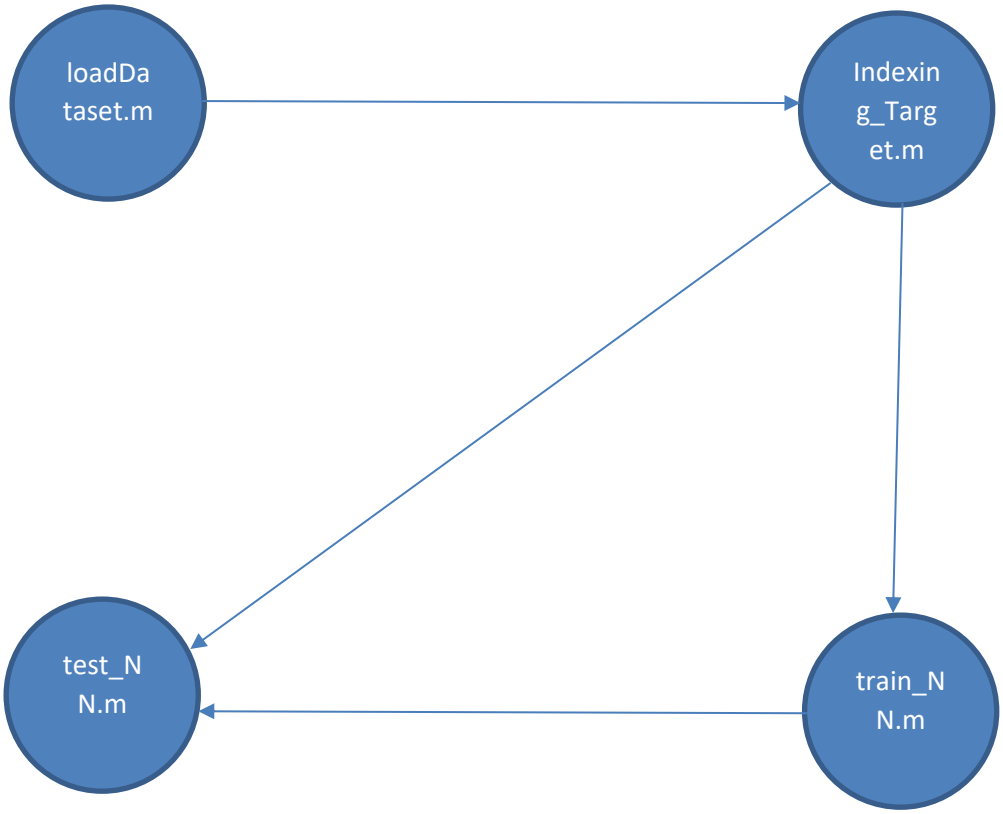
  Two different autoencoders were trained and features were extracted with those autoencoders (the second autoencoder has the extracted features of the first autoencoder as input).
  A softmax layer was trained with the output of the second autoencoder extraction as input.
  All the layers were stacked up and the result was used as the network to train.
  Apart from the normal result demonstration, the autoencoders will create a confusion matrix to show the results

```
autoenc1 = trainAutoencoder(p_matrix, hiddensize1, ...
    'L2weightRegularization', 0.001, ...
    'SparsityRegularization', 4, ...
    'SparsityProportion', 0.05, ...
    'DecoderTransferFunction', 'purelin');

features1 = encode(autoenc1, p_matrix);

autoenc2 = trainAutoencoder(features1, hiddensize2, ...
    'L2weightRegularization', 0.001, ...
    'SparsityRegularization', 4, ...
    'SparsityProportion', 0.05, ...
    'DecoderTransferFunction', 'purelin', ...
    'ScaleData', false);

features2 = encode(autoenc2, features1);

softnet = trainSoftmaxLayer(features2, t_matrix, 'LossFunction', 'crossentropy');

net = stack(autoenc1, autoenc2, softnet);

net = train(net, p_matrix, t_matrix);
```

5.Software Architecture

The architecture of our software is developed in the following way:

- `train_NN.m`: This function will create the networks discussed in this project and will train them with the mentioned parameters.
- `loadDataset.m`: The aim of this function is to pre-process the given dataset.
- `indexing_target.m`: creates T matrix.
- `test_NN.m`: test the result of the network's result, with the computed sensitivity and specificity.

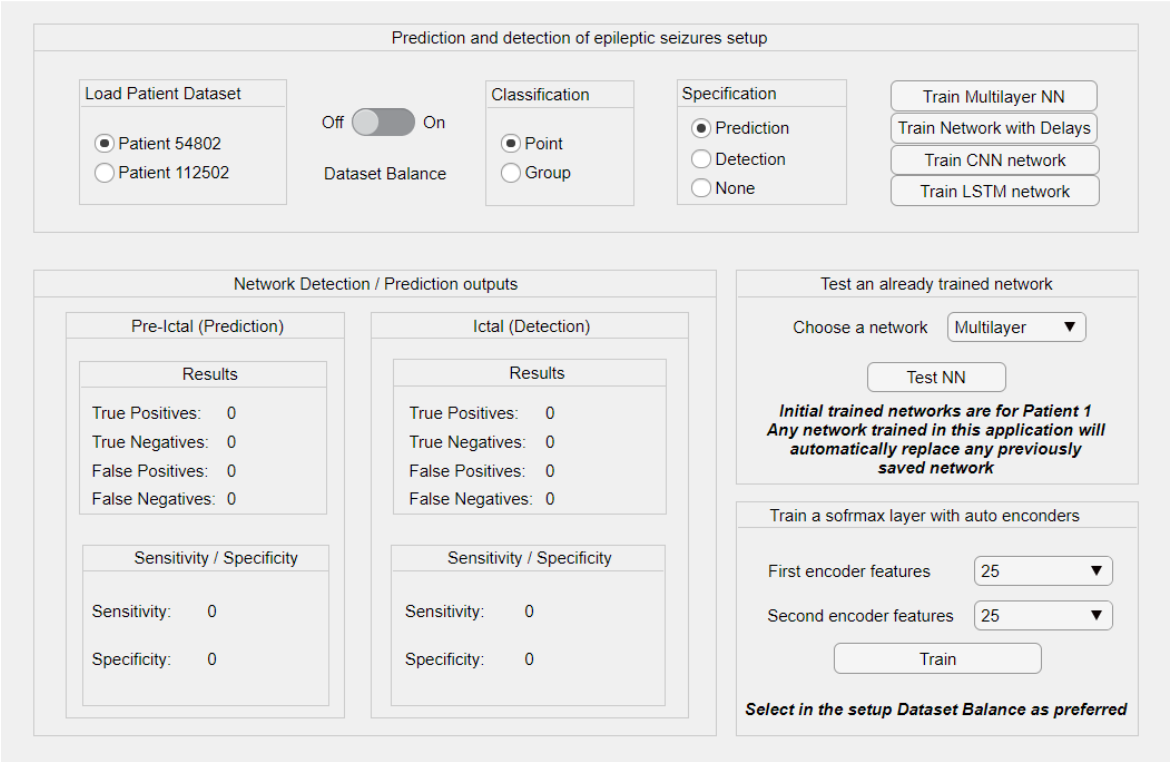


6.GUI

An interactive GUI was developed in order to allow the user to test and use the application.

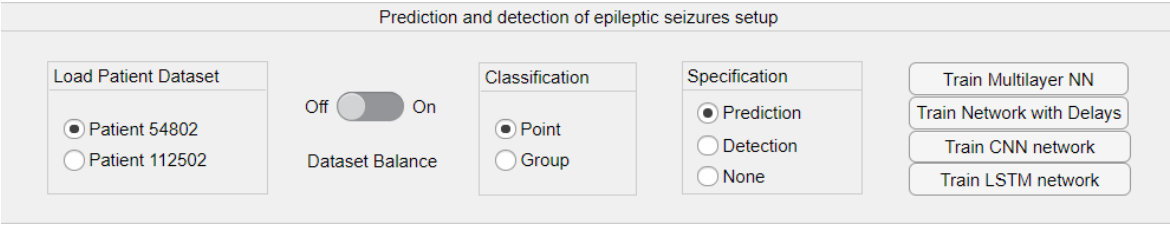
In order to run the application, simply unpack the compressed file, switch your matlab workspace to the folder in which you saved the files and run the file epileptic.mlapp by typing ‘epileptic’ in the matlab command line.

In case you need to manually add the patient’s dataset (54802.mat and 112502.mat), just add them to the same directory / path of the project files.

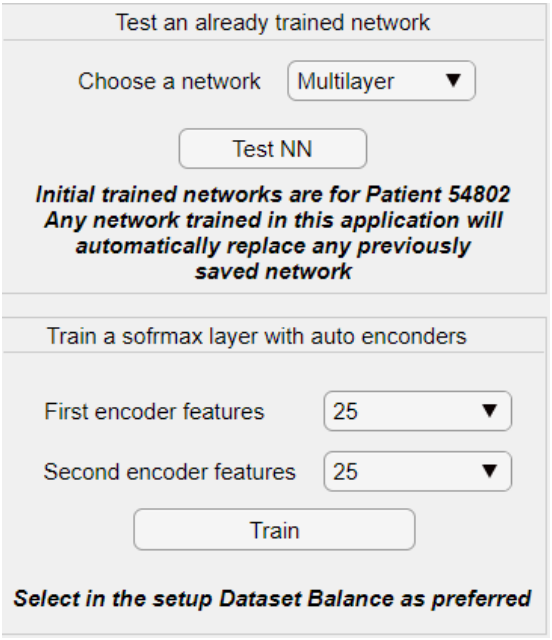


In this app, the user can choose either to train and test a network, or to only test a network. Bear in mind that if the user trains a network, it will replace any other saved network of the same type, so our suggestion is to first test the initial networks in order to save time training a new one.

There are already trained networks for every network in the project scope. In order to save time, we recommend you to first test the saved networks prior to train a new one (training a new network will override the previously saved network). Guidelines to use a saved network will be presented below

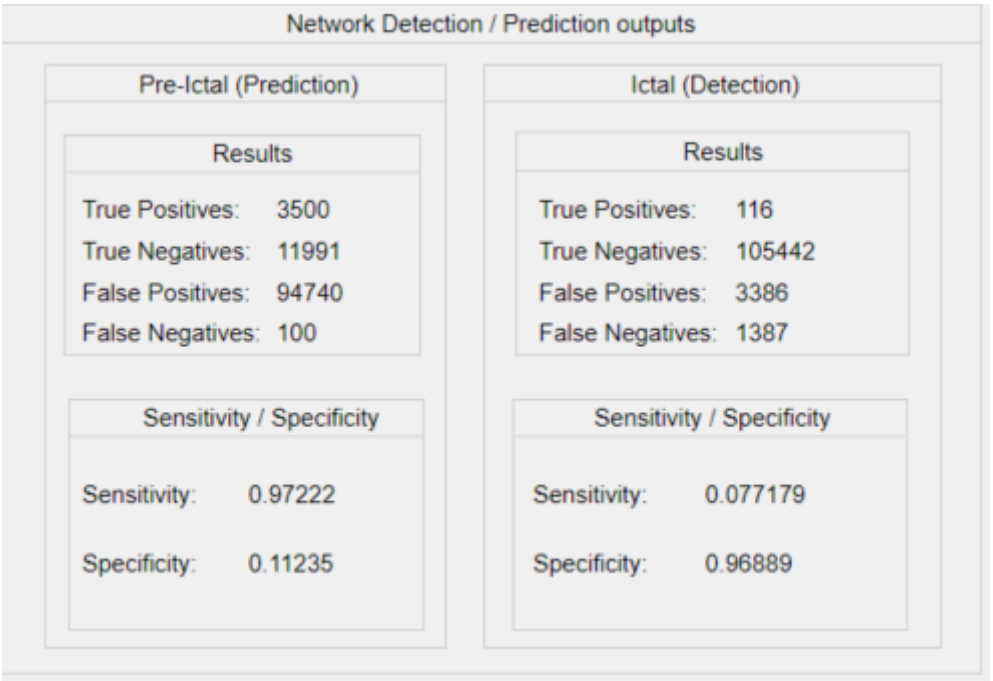


In the setup, the user can choose a Patient, choose if he wants class balancing, the type of classification, the specification, and the type of network to be trained (after training the network, a classification will be automatically made). CNN and LSTM networks will ignore the specification field, since it is not using custom error weights



The user can also test an already trained network by choosing a network from the dropdown box. Keep in mind that a patient, dataset balance, classification and specification (if applicable) shall be chosen from the setup menu.

Additionally, the user can as well train and test 2 autoencoders stacked in a softmax layer. The user can also choose how many features he wants in each autoencoder.



Results will then appear in a box, with the computed Sensitivity and Specificity.

7.Conclusion

From this assignment, we could have some insights regarding deep and shallow networks.

We could see that the LSTM is the most indicated network for this type of problem (out of the tested networks), followed by the multilayer network with delays. This network gave us the most stable Sensitivity and Specificity results amongst all.

The dataset from patient number 2 only had around 1 per cent of ictal points, making almost every network (except LSTM and Multilayer with delays) struggle to correctly detected a seizure. The LSTM, despite these conditions, excelled and made this with quite good results.

As a learning / developing point, we saw that the data pre-processing is crucial in this type of problems, and that there is room for improvement in the way that we balance the dataset and assign error weights. This is a subject that we will look later on to improve.

The networks that disappointed the most were the autoencoders stacked with a softmax layer and the CNN network (the CNN network for sure has a lot of room for improvement, in the way that we format the input data and in the way that we stack up layers. There was still a lot to explore).

In the other hand, when we tried to train the CNN and LSTM networks without Dataset Balancing, the graphics during training were promising and we believe that it could improve results especially in CNN network. That will be something to explore afterwards since the training of the networks without dataset balancing will take a very long time (especially the LSTM)

On an ending note, this project helped to understand how deep and shallow networks work, helped us to understand their usage, and to see their pros and cons compared to each other.