UNIVERSIDADE D
COIMBRA

# Aprendizagem Computacional / Machine Learning

## Assignment TP2b: Prediction and detection of epileptic seizures

**Authors**: Hugo Bronze Canelas de Brito Prata - 2014198526

Luigi Masi - 2021195907

# Index



# 1. Introduction

Epilepsy is a neurological disease known for the alterations in the brain activity that leads the person to having convulsions. This kind of crisis can lead to very dangerous situation for the patient's health, so it could be useful to predict these crises and mitigated and, even so, the wellbeing of the patient is not guaranteed.

The main goal of this work is to predict and detect epilepsy crisis by using these deep neural network architectures:

- Feed Forward Neural Network
- Recurrent Neural Network
- Convolutional Neural Network
- LSTM Neural Network

# 2. Dataset

Two datasets have been used: one with the EEG records of 44202 patient and 63502 patient. By looking at these EEG records it's possible to indicate the mental state of the person through signal frequencies. 29 features have been used and they correspond to the 29 different frequency bands (0.5Hz to 512Hz). Patient 44202 has 22 recorded seizures and patient 63502 has 19 recorded seizures.

We can divide the mental state of a person in 4 mainly states by referring to the seizure (although, only the first three of them have been used as class):

- Interictal (1 0 0): indicates that the brain is in a normal state.
- Preictal (0 1 0): indicates that a seizure is imminent.
- Ictal (0 0 1): indicates that a seizure is happening
- Postictal: indicates that a seizure has just finished recently

## 2.1 Target

The target provided in both datasets has been split in two classes (binary categories): Ictal (1) and Non-Ictal (0).
To get the three mental states mentioned before, the dataset has been treat and altered: it is considered that the first 15 minutes (900 examples/seconds) before the seizure starts (before the next 1 in the target vector) belong to the Preictal class (2) while the following 1's in the target are changed to the Ictal Class (3); all the zeros left are changed to the class Interictal (1) until the next start of a seizure and the cycle repeats. To do this kind of transformation, the function *T_convert_to_output_format* has been used: this new vector is converted to a matrix where, instead of having one column with the number of the class, there are four columns that represent their respective class (e.g. Ictal class goes from 3 to [0 0 1]).

## 2.2 Dataset division

The dataset was divided into a training dataset and a testing dataset. The training dataset is comprised of roughly 70% of the records while the testing dataset is comprised of the complementary 30%.

By referring to the patients:

- Patient 44202: the training dataset has 15 seizures and the testing dataset contains 7 seizures;
- Patient 63502: the training dataset has 13 seizures and the testing dataset contains 6 seizures.

## 2.3 Error weights

Since the dataset has records of over 100 hours, and seizure lasts in average 1 minute and 30 seconds, the datasets are majorly comprised of Interictal states. To deal with this, the training dataset has been balanced to obtain a less impartial dataset.

The error weights were defined by the function *penalize_classification* and *penalize_detection* so that the weights were set with the purpose of favouring the non-common instances such as Ictal and Preictal and, therefore, these had a higher value than instances of Interictal. By this, less important classes, such like Interictal, have been penalized.

## 2.4 Class balancing

In the dataset was also applied class balancing and it was selected just a number of Interictal instances per seizure where the sum of these were equal or less the to the sum of Ictal and Preictal instances. This was only applied to the training set and it was done by *balance_training_dataset*.

## 2.5 Parallelism and Graphical Processing Unit

By using the Matlab's Parallel Computing Toolbox and by specifying the relevant arguments to their respective training methods, it was used the backpropagation algorithms in parallel and the graphical processing units (GPU). This was eventually not used because it caused an incompatibility error.

# 3. Architecture

We can distinguish two parts:

- Shallow Neural Networks: it was used *Feedforward Neural Networks* and *Layer Recurrent Neural Networks*
- Deep Neural Networks: it was used *Convolutional Neural Networks* and *Long Short-Term memory Neural Networks*

## 3.1 Feedforward

In the feedforward networks, the information goes through an input layer and “proceeds” through the following hidden layers until reaching the output layer that gives the network’s output.

To achieve this, it was used the Matlab’s function *feedforwardnet()*:

```
% Feedforward NN
function [trained_net, tr] = net_FeedForward(P, T, activation_1, activation_2, training, err_weights)
    trained_net = feedforwardnet([100 100]);
    trained_net.trainFcn = training;
    trained_net.layers{1}.transferFcn = activation_1;
    trained_net.layers{2}.transferFcn = activation_2;

    % window
    trained_net.trainParam.showWindow = 1;
    trained_net.divideFcn = '';

    % Parameters
    trained_net.performParam.lr = 0.01;
    trained_net.trainParam.epochs = 250;
    trained_net.trainParam.show = 35;
    trained_net.trainParam.max_fail=500;
    trained_net.performParam.regularization = tanh(1);
    trained_net.trainParam.goal = 1e-6;
    trained_net.performFcn = 'sse';

    % train net gpu
    [trained_net, tr] = train(trained_net, P, T, [], [], err_weights, 'UseParallel', 'yes', 'UseGPU', 'no');

end
```

## 3.2 Layer Recurrent

In the layer recurrent networks, we have a recurrent connection with one or more delays in all layers except for the output layer. By this, the network has an infinite dynamic response to time series input. To achieve this, it was used the Matlab’s function *layrecnet():*

```
% Recurrent NN
function [trained_net,tr] = net_Recurrent(P, T, activation_1, activation_2, training, err_weights)
    trained_net = layrecnet(1:2,[100 100]);
    trained_net.trainFcn = training;
    trained_net.layers{1}.transferFcn = activation_1;
    trained_net.layers{2}.transferFcn = activation_2;

    % window
    trained_net.trainParam.showWindow = 1;
    trained_net.divideFcn = '';

    % Parameters
    trained_net.performParam.lr = 0.01;
    trained_net.trainParam.epochs = 250;
    trained_net.trainParam.show = 35;
    trained_net.trainParam.max_fail=500;
    trained_net.performParam.regularization = tanh(1);
    trained_net.trainParam.goal = 1e-6;
    trained_net.performFcn = 'sse';

    % train net gpu
    [trained_net, tr] = train(trained_net, P, T, [], [], err_weights, 'UseParallel', 'yes', 'UseGPU', 'no');

end
```

## 3.3 Convolutional Neural Networks

In the Convolutional Neural Networks, it can be reach accurate results and they also work well with transfer learning, although there is no need for manual feature extraction.

```
% multiclass classification (deep learning)
function [trained_net, tr] = deep_net_CNN(P, T, feature_dim)
    % Revert T from output format to 3Class format.
    T = T_convert_output_to_3class(T);

    % function to format data into images -->
    [P,T] = input_format_CNN(P, T, feature_dim);

    % format to categorical and transpose
    T = categorical(T);
    T = T';

    % net settings
    output_size = 3;
    inputSize = [feature_dim feature_dim 1]; % 29
    layers = [ ...
        imageInputLayer(inputSize,'Normalization','rescale-zero-one');
        convolution2dLayer(5,20)
        reluLayer
        maxPooling2dLayer(2,'Stride',2)
        fullyConnectedLayer(output_size)
        softmaxLayer
        classificationLayer];

    options = trainingOptions('sgdm');

    % train & save net
    [trained_net, tr] = trainNetwork(P, T, layers, options);

end
```

The data has to be pre-processed into images and then it could be applied the Matlab's function *trainNetwork*.

```
function [P_formatted, T_formatted] = input_format_CNN(P, T, feature_dim)
    [~, y] = size(T);
    % 29x29x1 ||  15x15x1 ||  10x10x1 || 3x3x1
    P_formatted = P(1:feature_dim, 1:feature_dim, 1);
    T_formatted = ones(1,y);

    T = categorical(T);
    T = T';

    % dataset example --> 15x29187 feature_size = 15 --> 15x15 images
    % this means total image should be less than 29187/15 = +/-1900
    % start at second iteration
    num_imgs = 2;
    i = feature_dim+1;
    while i+feature_dim-1 <= size(P,2)
        % only have same class in an image
        count = 0;
        for j=i : i+feature_dim-1
            if T(j) == T(i)
                count = count+1;
            end
        end

        % update formatted data
        if count == feature_dim && i+feature_dim-1 <= size(P,2)
            % save P image in the size x size x 1 format
            P_formatted(:, :, :, num_imgs) = P(1 : feature_dim, i : i+feature_dim-1, 1);
            % save T class
            T_formatted(num_imgs) = T(i);

            num_imgs = num_imgs+1;
        end
        i = i + count;

    end

    % round so size of P and T are equal
    if size(T_formatted) ~= num_imgs
        T_formatted = T_formatted(1 : num_imgs-1);
    end

end
```

### 3.4 Long Short-Term Memory Neural Networks

In the LSTM Neural Networks, we have a structure similar to Recurrent Neural Network which, however, solve the short-term memory problem: the LSTM neural net can learn long-term dependencies between time steps of sequence data.

```
function [trained_net, tr] = deep_net_LSTM(P, T, feature_dim)
    % data format for input and target. LSTM is picky about inputs.
    % format to time sequence and transpose
    P = con2seq(P);
    P = P';

    % Revert T from output format to 3Class format.
    T = T_convert_output_to_3class(T);

    % format to categorical and transpose
    T = categorical(T);
    T = T';

    % net settings
    inputSize = feature_dim;  % 29
    output_size = 3;
    num_hidden = 100;

    maxEpochs = 100;
    miniBatchSize = 1000;

    layers = [ ...
        sequenceInputLayer(inputSize)
        bilstmLayer(num_hidden, 'OutputMode', 'last')
        fullyConnectedLayer(output_size)
        softmaxLayer
        classificationLayer];

    options = trainingOptions('adam', ...
        'ExecutionEnvironment', 'cpu', ...
        'GradientThreshold', 1, ...
        'MaxEpochs', maxEpochs, ...
        'MiniBatchSize', miniBatchSize, ...
        'SequenceLength', 'longest', ...
        'Shuffle', 'never', ...
        'Verbose', 0, ...
        'Plots', 'training-progress');

    % train & save net
    [trained_net, tr] = trainNetwork(P, T, layers, options);

end
```

## 4. Methodology

The assignment required to:

- To build, train and test multilayer networks for classification of big data sets;
- To build, train and test dynamic neural networks (with delays) for multidimensional time series prediction;

- To face the problem of features reduction with autoencoders;
- To configure, train and test Convolutional Neural Networks for multiclass classification (deep learning);
- To configure, train and test LSTM (Long Short-Time Memory Neural Networks) for multidimensional time series classification.

To predict and detect seizures of the patients, we used a training set and a test set. Because of the few seizures in each dataset and since we divided into training set and testing set, we used our test as validation set to validating the model.

In our shallow networks we did small tests to decide on the functions to be used. Taking into account tp2a and these tests, we decided to use *tansig*, *pureline*, and *trainsgc*. *Softmax* and *logsig* were also tested.

## 4.1 Autoencoders

Due to the number of features, the noise intensified when providing information to deep neural networks and shallow neural networks.

A multiple encoder was used where the number of features is reduced to 15, 10, and 3, to achieve discriminative features with less noise.

This approach didn't improve classification in both architectures.

Feature reduction was considered in order to select only the most discriminative features and it was used in both shallow networks. This required us to also reduce the input size of classifiers from static 29 to the number of features.

```
function [features] = autoencoder(dataset, number_of_reductions)
    P = dataset;
    feature_reduction = [15, 10, 3];
    % reduce none
    if(number_of_reductions == 0)
        features = P;
    end

    % reduce 1 time
    if (number_of_reductions == 1 || number_of_reductions == 2 || number_of_reductions == 3)
        encoder_output = trainAutoencoder(P, feature_reduction(1), 'MaxEpochs', 50, 'L2WeightR
        features = encode(encoder_output, P);
    end

    % reduce 2 times
    if (number_of_reductions == 2 || number_of_reductions == 3)
        encoder_output = trainAutoencoder(features, feature_reduction(2),'MaxEpochs', 50, 'L2W

        features = encode(encoder_output, features);
    end

    % reduce 3 times
    if (number_of_reductions == 3)
        encoder_output = trainAutoencoder(features, feature_reduction(3),'MaxEpochs', 50, 'L2W
        features = encode(encoder_output, features);
    end

end
```

# 5. Results

## 5.1 Feedforward Neural Network

The number of neurons per layer and the training function were changed to check which one would suit more. We used *trainscg* for training because it is appropriate for the classification of multilayer problems.

Regarding the first patient, most of the configurations provided passable results for Detection.

Regarding the second patient, the results were similar because the number of seizures was almost the same.

```
 FF DET

TP:75168.0   FP:2883.0  TN:153363.0     FN:2955.0
accuracy = 97.5%
Sensivity = 1.0%
Specificity = 1.0%
```

```
 FF CLASS

TP:0.0    FP:0.0      TN:234369.0      FN:0.0
accuracy = 35.9%
Sensivity = NaN%
Specificity = 1.0%
```

Our best results were 98% accuracy while the average results hovered on the 78% mark.

### 5.2 Layer Recurrent

In the Layer Recurrent Neural Network, each layer has a recurrent connection and there is a tap delay associated with it to achieve a dynamic response to time series input data. The delay chosen was 1:2 per layer. For this network, we used 3 layers with the same functions used in Feedforward Neural Network.

Again, most of the configurations provided passable results for Detection and bad for classification.

```
 REC DET

TP:75222.0   FP:2830.0  TN:153416.0     FN:2901.0
accuracy = 97.6%
Sensivity = 1.0%
Specificity = 1.0%
```

```
 REC CLASS

TP:0.0    FP:0.0     TN:234369.0     FN:0.0
accuracy = 35.8%
Sensivity = NaN%
Specificity = 1.0%
```

For what concern the second patient, the results were similar to the previous scenario.

Our best results were again 98% accuracy while the average results hovered on the 86% mark. This leads us to believe both the shallow networks have similar accuracy, although recurrent runs a lot faster and is more consistent, so it seems the preferable option.

### 5.3 LSTM and CNN

While we were able to create, train and test both the LSTM and CNN classifiers, the results were below expectation, which leads us to believe there is an error with how the metrics are calculated for these classifiers or how they are tested since these classifiers require a lot more data formatting and, unfortunately, we could not dedicate enough time for testing and metrics. We tested both with level 0 and 1 autoencoding.

```
 LSTM DET

TP:78123.0   FP:156246.0    TN:0.0   FN:0.0
accuracy = 33.3%
Sensivity = 1.0%
Specificity = 0.0%
 CNN DET

TP:5206.0    FP:10412.0     TN:0.0   FN:0.0
accuracy = 33.3%
Sensivity = 1.0%
Specificity = 0.0%
```

```
 LSTM CLASS

TP:0.0   FP:0.0     TN:234369.0     FN:0.0
accuracy = 33.3%
Sensivity = NaN%
Specificity = 1.0%
 CNN CLASS

TP:0.0   FP:0.0     TN:15618.0  FN:0.0
accuracy = 33.3%
Sensivity = NaN%
Specificity = 1.0%
```

```
 LSTM DET

TP:69412.0   FP:138566.0    TN:17680.0  FN:8711.0
accuracy = 29.6%
Sensivity = 0.9%
Specificity = 0.1%
```

```
TP:2820.0    FP:5674.0  TN:150572.0     FN:75303.0
accuracy = 1.2%
Sensivity = 0.0%
Specificity = 1.0%
 CNN DET

TP:2688.0    FP:5378.0  TN:5.0  FN:4.0
accuracy = 33.3%
Sensivity = 1.0%
Specificity = 0.0%
```

We belive this is caused by the lack of capability of the classifier to relate data, which might be caused by noise. This noise then overfits the data.

# 6. Conclusion

For all the architectures considered, the detection was easier to approach while classification appeared as a bigger challenge. This was an expected result as detection only requires us to find the start of a seizure while classification requires us to predict one. Both patient results were similar because the number of seizures was also similar.

We believe we successfully completed a big portion of our goals with this project.

- We managed to create and train all targeted classifiers successfully, and we correctly formatted the training data to train said classifiers, including converting the data to square greyscale images. Most of the time, the training gave us results near the 70% or higher for all classifiers.
- We successfully created autoencoders to reduce the features and we managed to balance the classes in classification.
- We also successfully attributed weight to each class within the dataset, when required.

- We successfully trained half the classifiers with decent results, with the other two requiring more time to debug.

Our research led us to believe that the LSTM would be the most indicated network for this type of problem (out of the tested networks), followed by the multilayer network with delays. The reason is that CNN is effective for image recognition and it is not for data that varies in time (although possible to use). We believed from the start that the results for each patient would be similar, since the number of seizures is similar and the amount of data is not too different. We consider the data pre-processing crucial in this type of problems, and that there is room for improvement in the way that we balance divide and format the dataset and assign error weights.

Time constraints were our biggest problem. Because of this we were unable to properly debug the problem with testing our deep networks, and so are unsure whether the problem lies with the metrics function or the test data. Training gave us good accuracy, so we believe that the problem lies with the previous two functions. Our metrics function was very rushed and we don't fully trust the results in classification. We also believe there's a problem with the test data given to the deep networks. While we created the train data and were able to debug and confirm it was correct, our tests were also rushed for the deep networks as they were only finished close to the deadline. Even so, we managed to achieve decent results with the shallow networks.

Had we gotten more time, we would have wanted to further improve our metrics function which had the least time devoted to. We would have also wanted to dedicate more time to confirm the test data was correctly formatted before being given as input to the deep networks.

## 7. GUI

Due to time constraints, we could not finish a GUI. As a substitute we created a simple input area at the start of the main file that's made up of four lines of settings.

```
% DEFAULT - Patient 1, 70% training 30% test, create classification sets with autoencode level 1 which reduces features to 15

% SETTINGS

%  autoencode_lvl
%  0 = none
%  1 = 15
%  2 = 10
%  3 = 3

%  patient
%  1 = 44202
%  2 = 63502

% percent_train - how much of the dataset is used for training, rest is used for testing

% create_networks_from_scratch - This will override all the old classifiers and create new ones
% from scratch for both classification and detection if set to anything
% besides -1. Leave as -1  if you dont need new classifiers.

autoencode_lvl = 1;
patient = 1;
percent_train = 0.7;
create_new_networks_from_scratch = -1; % no = -1 | yes = anything else


% USER DOES NOT NEED TO CHANGE ANYTHING PAST THIS POINT
```