Epilepsy

João Almeida & Ricardo Margarido

I. Introduction

This project was made on scope of the course Neural Computation and Diffuse Systems in the University of Coimbra. The objective of this work is to allow a user in a simple way, through a GUI, to predict or detect a seizure on a 29 feature vector extracted from EEG's of several patients. This is a problem that has been studied for many years, yet nobody has been able to find a general solution that can be applied for every patient. The key points that we noticed that affected our results the most were the data set itself (patient 1 revealed better results than our patient 2) and the post processing used to analyze the classifier results.

II. GUI

In this section it will be briefly explained how to work with our GUI, that is represented in Figure1. To start using it, just run main.m .

A quick notice: given that the GUI is pre-configured to use GPU so the number of epochs is set to 1000.

If you wish to train using the CPU or using a function that is not supported for GPU training just go to CostumClassifier.m and change net.trainParam.epochs to the desired epochs and remove the 'useGPU' , 'yes' of net = train(net,TheVector,TheTarget,'useGPU','yes');

The GUI is divided in two parts: one to train a new network, and other to use networks that we already pre-trained. To train a network, you must first fill in the parameters required. You must choose between a Layer Recurrent Network or a FeedForward Network. After that, you must decide which data set to use: one of the patients we were given for this assignment or a new patient. Just make sure it contains it's vectors and targets on variables named Trg and FeatVectSel.

If you decide to train data from a new patient, the data will be processed. On the Layers and Neurons sections, you have to write a vector. If you write, for example [10 20], your network will have two hidden layers with 10 and 20 neurons respectively. On the classifier functions, you must write only the functions you want to use. For example, if you want to use 'traingd' as your training function, just write traingd in the box. After filling these parameters, you're all set. Click Train, and the network will start being trained. If you chose 'Train with different data' as an option, you will be asked to choose a .mat file of the new data.

To use pre-built networks, just chose the data you want to use and click Test. You will be asked to chose a existing net to test your data.

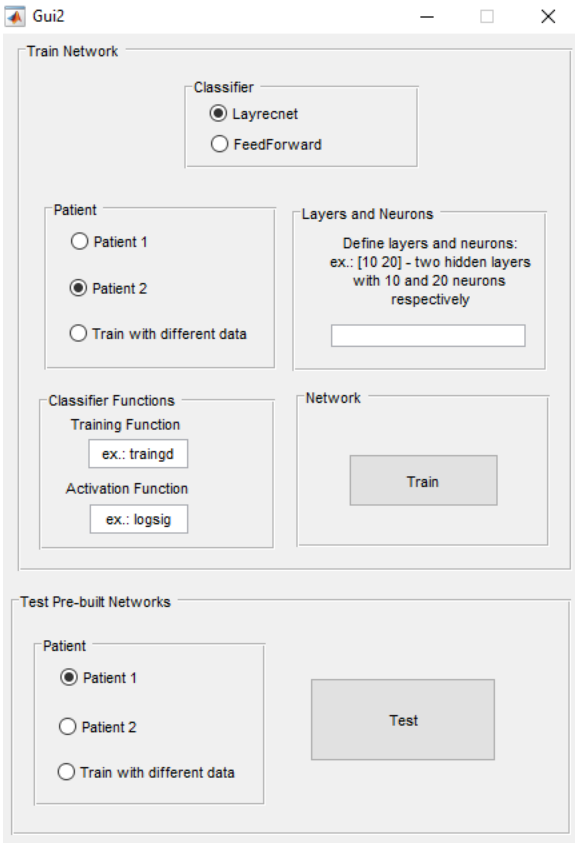

Figure 1: *GUI - example*

III. Methods

i. Data Pre-Processing

When we started, it quickly became obvious that processing the data would be a very important part of this work. We began by labeling the pre-Ictal and post-Ictal on our Target variable. We realized that our Inter-Ictal data was about 90% if not more of all the data from our patients, so we started by down-sampling it.

We tried our best to make all the classes have a good representation (making them the same size proved to not be the best solution since it doesn't simulate what happens in the real case being studied), so that the classifier could have a easier time distinguishing them. Since the data was already normalized and composed by extracted features from the EEG signal, we decided there wasn't much else we could do to pre-process it.

ii. Networks

On this project we used Layer Recurrent Networks and Feed Forward Networks. The Feed Forward Networks were the ones who achieved better results. This result was a bit counter intuitive since we were expecting Recurrent Networks to be better because they tend to have memory just like the human brain and could be a better approximation of the in-vivo conditions.

On this part, we mostly used GPU's to train all the networks (given the size of the data, the complexity of some networks and the number of epochs run we greatly recommend to run it on GPU if that option is present) , and see which parameters were revealing the best results. We used several hidden layers with a big number of neurons to try and achieve the best results. This was not a great method to use, since the network started over-fitting (there were more neurons and connections than points in our data).

As we tried different approaches combining layer numbers with neurons per layer it became clear that some networks response was better than others. That happens because sometimes adding more layers or neurons makes the problem more complex but doesn't give a better solution. Most of the times the simplest solution is indeed the best.

Matching the number of neurons+bias to the size of the data set proved to be a very good first approach that served as base comparison to the networks tested.

iii. Data Post-Processing

The data post-processing was what allowed us to have much better results. First, we went point by point, and made the classifier detect classes in all points. After that, we tried to predict the ictal phase, only looking for pre-Ictal points. Both of these methods resulted in pretty low sensibilities and specificities. To try

and fix that we used our best networks, and instead of classifying point by point, we used a window of 10 points, and checked if there were at least 5 points of a given class in that windows. This method allowed to overcome the low accuracy of the classifiers, and improved our results by a big margin. This increase can be seen in the results section below but it basically helps with de-noising the neural network output.

IV. Results

Table 1: *Point By Point*

			Detect PbP		Predict PbP	
	Function	Net	Sens	Spec	Sens	Spec
10 100 10	logsig	FF	66.331	97.651	1.294	99.593
30	purelin	FF	20.525	99.707	0	99.848
170 100	logsig	FF	61.856	98.64	0.039	100
30 100	purelin	FF	25.335	99.605	0.019	99.68
600	logsig	FF	61.017	98.83	0.156	100
30 100	purelin	R	30.257	99.551	0.039	99.965

Table 2: *Method 2*

			Detect Met 2		Predict Met 2	
	Function	Net	Sens	Spec	Sens	Spec
10 100 10	logsig	FF	69.728	97.511	45.945	99.465
30	purelin	FF	74.397	99.181	57.142	99.901
170 100	logsig	FF	93.09	98.298	NaN	99.991
30 100	purelin	FF	66.444	98.925	44.444	99.766
600	logsig	FF	92.47	98.473	NaN	99.964
30 100	purelin	R	77.959	98.849	NaN	99.964

To calculate the sensitivity and specificity of the classifiers we used the function testarRede.m, which compares the output of the simulation of each network for the processed data and compares it to the targets vector that was given to train the neural network. Besides computing these two key parameters point by point we also take a more relaxed approach since most of the times point by point will induce error in the predictions. We can see by the table 1 that the results vary a lot. Only some of the neural networks trained and tested are displayed since with this report comes an .xlsx file with all the computed scores of all networks for a more in-depth analysis. Besides that the user can also check this scores manually by testing the networks in our GUI. By looking at table 2 and comparing with table 1 it is noticeable the increasing in performance by not using the point by point method of interpretation of the classifier output. This simple processing revealed to be a very good choice of looking at our predictions and try to find meaning in them.

V. Discussion

i. Data Set

The data set is one of the most important parts when designing a classifier. If the data set does not represent a correct sampling of the data we want to train, the classifier will have a very poor generalization ability.

In this particular example we could see this effect when we first tried to run with all the data that was given and without any processing. The scores were very low because it was overfitting to the inter-ictal state since 90% of all the data was in fact inter-ictal. From this experience we can see how much of a role does data processing has in the performance of our classifiers.

ii. Neural network architecture

We tried different kinds of networks, with a variable number of hidden layers, neurons in those layers, transfer functions and training functions. We became quickly aware of the fact that adding more layers or putting a lot of neurons in a single layer wasn't doing any good for the performance since most of the times it was overfitting to one class.

We also noticed that most of the times it was best to make all transfer functions the same because when we tried to mix things up with different functions in different layers the scores were decreasing fast.

With that in mind with tried to match the number of neurons to the size of the data set, use always the same transfer function and compare the performance of different architecture.

We had mixed results regarding what architecture was the best. We need to look at the problem in 3 distinct ways: Do we want to predict? Do we want to detect? Or do we want to do both at the same time?

If we want to only predict it is clear the a Feed Forward net with one hidden layer of 30 neurons and purelin as transfer function serves us with somewhat satisfactory results (57% Sensibility and 99.9% Specificity). This simple network also appears to be the jack of all trades, being able to also detect with some degree of certainty (74.4% Sensibility and 99.2% Specificity). This network was a surprise because it is extremely simple in it's structure and we initially thought it wouldn't be capable of differentiating the data given.

If on the other hand our focus is purely on detecting we can look at the table and see that a Feed Forward net with 2 hidden layers, the first one with 170 neurons and the second with 100 and logsig as the transfer function proves to be the best one (93.1% Sensibility and 98.3% Specificity). This results comes with no surprise since it was obtained by calculating the number of neurons equal to the data set size.

iii. Results

As expect when we first started to work on this topic the results vary a lot and most of the classifiers were unsuccessful in providing a good solution to the problem presented. This stems from various reasons: the biggest one is the fact that we are not sure of what happens when someone has an episode. We can see some frequencies being over stimulated while others drop abruptly in strength. That coupled with the fact that many times the causes of said episodes are also unknown makes it extremely difficult to say with features are more relevant than others. Other of the problems come from the fact that normally this episodes are very short (in one of our patients it was around 100 seconds while the other had 10 seconds) which makes it hard to collect information about the features being expressed on this particular event.

The pre-ictal and pos-ictal points are also a problem when trying to do a classifier like this because they are estimated. It is not known if the pre-ictal state really corresponds to 300 seconds before the episode and if the pos-ictal is really 150 seconds. This lengths could vary and we could be assuming points that are inter-ictal as one class or the other degrading even more the overall performance

of the classifier.

As said because of the length of the episodes we could only use one of the patients to train our neural networks since the other patient's episodes were too short to distinguish from the other states. This is also a major concern of this academic paper because the results obtained and the performances will drastically with the data provided.

VI. Conclusion

When we first tackle this problematic we were very perplexed because even with the correct segmentation we were getting no results at all. All the points were being classified as inter-ictal.

After some reading about epilepsy and neural network architectures we could finally manage to build some classifiers that were providing some insight in regards to what would be the better solution we could achieve.

As we got further into the work we started to tweak some small parameters and noticed an increase in performance. We changed the number of pre-ictal and pos-ictal points for example because these boundaries are not fixed yet.

We also arrived to the conclusion that sometimes some problems are not solvable with the knowledge we have at this time. This was the case of the second patient given that had episodes that were too short to work with. That being said, the networks that were able to be created with patient 1 prove to be somewhat successful when applied to patient 2. This proves to be a good marker on the generalization of the classifiers obtained since they are not too specific to the patient they are trained on.