Deep neural networks for analyzing recordings of Brain Computer Interfaces

Elsa Scola Martín and Enya Goñi Maganto

Abstract

In this document we report our proposal for the application of deep neural networks to classify Brain Computer Interfaces (BCI) signals of the BCI Competition III. The performed tasks were: Firstly, load the data and preprocess it by using the window method, adapting the label values and splitting the training data into train and validation data. Secondly, classify the signals with Recurrent Neural Networks (RNN) using Tensorflow, both with Long short-term memory units, and with Gated Recurrent Units. In addition, a Convolutional Neural Network (CNN) was implemented using Keras. Finally, our results show that in the contest our position would be slightly above the mean, obtaining our best results with the CNN. We can infere from this results that using CNNs for this sequence classification has been the best approach, as even if the data is sequential, LSTM and GRU do not perform so well when the time window is short.

Contents

1 Description of the problem **3**

2 Description of the dataset **3**

3 Description of our approach **4**

4 Preprocess the data **4**

4.1 Window method 4

4.2 Adapt the label values 5

4.3 Split the training data 5

4.3.1 Overfitting 5

4.3.2 Underfitting 5

4.3.3 Validation set 5

4.3.4 Our particular case 5

5 Classification with RNN - Tensorflow **5**

5.1 Tensorflow 6

5.2 Recurrent Neural Networks (RNN) 6

5.2.1 How they work 6

5.2.2 Backpropagation Through Time (BPTT) 7

5.3 Long Short-Term Memory (LSTM) 7

5.4 Gated Recurrent Units (GRU) 8

6 Classification with CNN - Keras **8**

6.1 Keras 8

6.2 Convolutional Neural Network (CNN) 8

7 Results - Analyze and contrast **9**

7.1 RNN (LSTM vs. GRU) 9

7.2 CNN 9

7.3 Comparison with the contest results 10

8 Conclusion **10**

9 Implementation **10**

1 Description of the problem

The Berlin Brain-Computer Interface site (BBCI) hosts, every year, its Brain Computer Interface (BCI) competition. Teams all over the world are invited to solve the problems available in the site and submit their results for grading. This year, 8 problems where described and teams were encouraged to ask for any of the datasets needed to solved them to submit their work [2].

BCIs are used to translate electrical signals into commands without the need for motor intervention. They are particularly useful for implementing assisting technologies providing communication and control to people with severe muscular or neural handicaps. They require a decoding component in which brain signals are translated into commands. Usually, classification algorithms are applied to predict the human intention from the analysis of the signals.

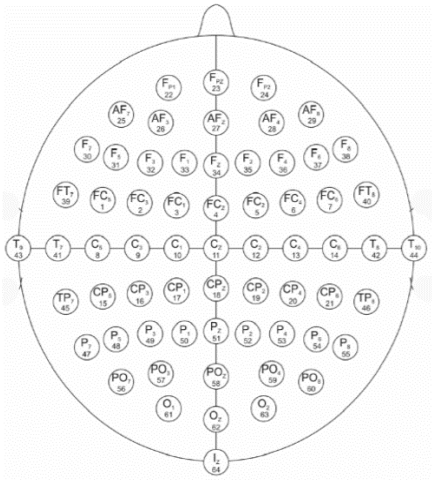

Figure 1: Brain Computer Interfaces: head of the subject seen from above. The circles displayed on the image show the electrode channels.

Several classification algorithms have been used to analyze brain data in the context of BCI applications. In this project we use Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) to solve the problem. We consider RNNs appropriate classes of artificial neural networks as several studies claim that they are proven to perform better when analyzing data in a time series. In addition, we have also chosen to use CNNs for sequence classification because they can learn from the raw time series data directly, and in turn do not require domain expertise to manually engineer input features. The model can learn an internal representation of the time series data and ideally achieve comparable performance to models fit on a version of the dataset with engineered features.

2 Description of the dataset

The dataset chosen from the 8 available [3] was "Data set I: motor imagery in ECoG recordings, session-to-session transfer". A common task in BCI is to apply a classifier that was trained during previous sessions during a later session without retraining it. The challenge of this task is that the electrical patterns of the patient might show some different characteristics on a new session. This kind of nonstationarity can be caused, for example, by changed levels of motivation, arousal, fatigue, etc. In addition, the recording system might have undergone slight changes concerning electrode positions and impedances.

Data set I reflects this situation: training and test data were recorded from the same subject and the same experimental task, but on two different days. As electrocorticography (ECoG) was used and not electroencephalography [6] (EEG), the variation of electrode positions and impedances are expected to be rather small.

Electrocorticography (ECoG), or intracranial electroencephalography (iEEG), is a type of electrophysiological monitoring that uses electrodes placed directly on the exposed surface of the brain to

record electrical activity from the cerebral cortex. In contrast, conventional electroencephalography (EEG) electrodes monitor this activity from outside the skull. ECoG may be performed either in the operating room during surgery (intraoperative ECoG) or outside of surgery (extraoperative ECoG).

The task was to set up a classifier based on the labeled training data of the first session and apply it to the unlabeled test data of the second session. The performance criteria used for evaluation was the percentage of correctly classified test trials.

The train dataset had 2 parts:

- Part 1: A 3D matrix containing the following information: 278 trials, where each one had the information that 64 electrode channels had gathered in parallel for 3 seconds, and 3000 samples of time series contained in each one of the 64 electrode channels.
- Part 2: A vector of -1/1 values, that corresponded to the labels, one for each trial.

The test dataset, on the other hand, had no labels.

3 Description of our approach

We organized the implementation of the project according to the tasks:

1. Load and preprocess the data
   (a) Window method
   (b) Adapt the label values
   (c) Split the data
2. Classification with RNN - Tensorflow
   (a) LSTM
   (b) GRU
3. Classification with CNN - Keras
4. Analize and contrast the results

4 Preprocess the data

4.1 Window method

In the context of ECoG recordings (similarly for EEG recordings), as it is commented in several papers [1] we have to take into consideration that the signal is highly influenced not only by external sources of noise but also by the spatial characteristics of the data recording process. The electrodes positioned according to a standard 10-10 or 10-20 electrode placement system collect data which may be heavily distorted by the activity of the adjacent cortex areas—potentially irrelevant to the state of the user's brain activity we want to classify.

Thus, for ECoG and EEG data, our assumption is that reducing the number of samples produced by a single channel within a given time window can produce information that is more valuable and free of noise than if the signal is treated as a whole (see the following study [5]). To reduce the number of samples we compute the mean for each 100 samples, therefore the window size is 100, with an overlap of 50 samples, which means that for each 100 samples we take 50 from the previous set of 100 samples; we do this to convert time values into cross sectional attributes.

Sliding time window methods are very useful in terms of fetching important patterns in the dataset that are highly dependent on the past bulk of observations. Moreover, using this technique we will obtain a reduced size of observations, this is specially convenient for us as Backpropagation Through Time can be computationally expensive when there is a high number of time steps. As it has been mentioned in the description of the dataset, our data has a shape of 278 x 64 x 3000, after applying this technique, we reduce the 3000 samples to 59.

4.2 Adapt the label values

As the Sigmoid function returns a value between 0 and 1, we need to change the -1 labels to 0. In this way, we can consider that the values above or equal to the threshold 0.5 classify as 1, whereas the ones below that value are classified as 0.

4.3 Split the training data

We usually split our data into two subsets: training data and testing data (and sometimes to three: train, validate and test), and fit our model on the train data, in order to make predictions on the test data. When we do that, one of two thing might happen: we overfit our model or we underfit our model.

4.3.1 Overfitting

Overfitting means that the model we trained has trained "too well" and is now fit too closely to the training dataset. This usually happens when the model is too complex. This model will be very accurate on the training data but will probably not be very accurate on untrained or new data. This happens because this model is not generalized, meaning you can't make any inferences on other data.

4.3.2 Underfitting

In contrast to overfitting, when a model is underfitted, it means that the model does not fit the training data and therefore misses the trends in the data. It also means the model cannot be generalized to new data. This is usually the result of a very simple model.

4.3.3 Validation set

The validation set is used to evaluate a given model, but this is for frequent evaluation. We use this data to fine-tune the model hyperparameters. Hence the model occasionally uses this data for "testing" and validating, but never does it "learn" from this. Validation set actually can be regarded as a part of training set (see Figure 2), because it is used to build your model. It is usually used for parameter selection and to avoid overfitting.

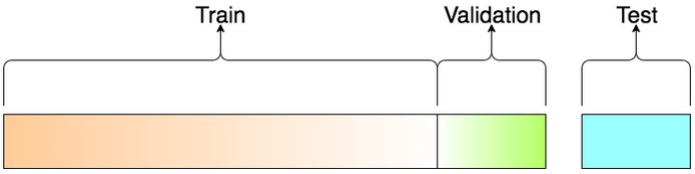

Figure 2: A visualization of the splits.

4.3.4 Our particular case

As it has been explained in the "Description of the dataset" part, our test data has no labels, therefore, we can't check the accuracy of our network once it is trained.

Thus, we divided our training dataset into 2 parts: train and validation. We decided that around 25% of the training data (70 cases) would be used for validation, and 75% for training (208 cases). The test dataset will only be used to make predictions and send our results to the challenge waiting for a response of the accuracy of our results.

5 Classification with RNN - Tensorflow

For our first implementation we chose to use Recurrent Neural Networks. Apart from the recommendation of the description of the problem, we consider it an appropriate class of artificial neural

network as several studies claim that RNNs are proven to perform better when analyzing data in a time series. For this implementation we have used Tensorflow.

5.1 Tensorflow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. The nodes of the diagrams represent mathematical operations and the edges reflect the arrays of multidimensional data (tensors) communicated between them, as shown on Figure 3.

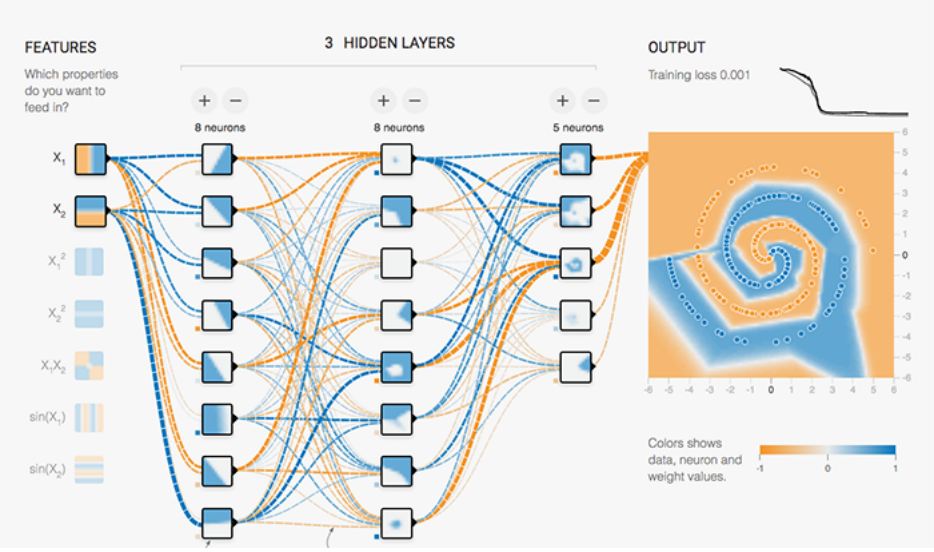

Figure 3: Tensorflow: structure of a neural network using Tensorflow.

The original goal of TensorFlow was to conduct research in the field of machine learning and deep neural networks, but now it is a system broad enough to be applicable in many other fields.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

5.2 Recurrent Neural Networks (RNN)

Because of their internal memory, RNNs are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next.

This is the reason why they are the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more, because they can form a much deeper understanding of a sequence and its context, compared to other algorithms.

5.2.1 How they work

In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously. See Figure 4 to understand the visual difference between a RNN and a Feed-Forward Neural Network.

RNN has a short-term memory. In combination with a LSTM they also have a long-term memory, but we will discuss this further below. A RNN produces output, copies that output and loops it back into the network. Therefore, a RNN has two inputs: the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

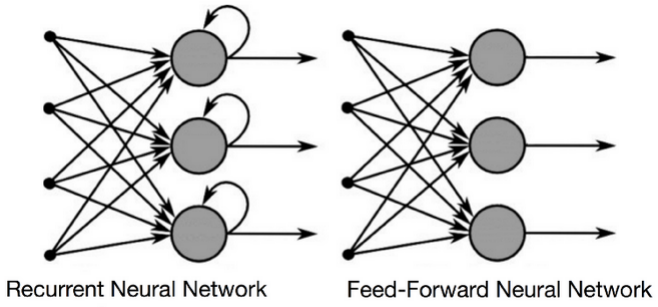

Figure 4: Recurrent Neural Network vs. Feed-Forward Neural Network.

5.2.2 Backpropagation Through Time (BPTT)

In most cases, when implementing a Recurrent Neural Network in the common programming frameworks, they automatically take care of the Backpropagation, nevertheless, it is needed to understand how it works.

A RNN can be viewed as a sequence of Neural Networks that can be trained one after another with backpropagation. The Figure 5 below illustrates an unrolled RNN. On the left, can be seen the RNN, which is unrolled after the equal sign. Note that there is no cycle after the equal sign since the different time steps are visualized and information is passed from one timestep to the next.

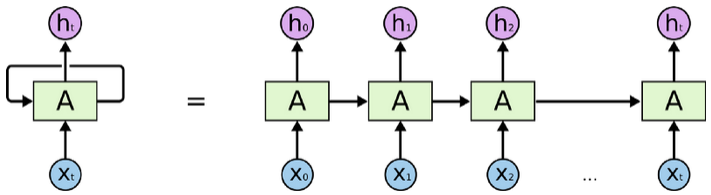

Figure 5: Backpropagation Through Time.

This illustration also shows why a RNN can be seen as a sequence of Neural Networks. If you do Backpropagation Through Time, it is required to do the conceptualization of unrolling, since the error of a given timestep depends on the previous timestep.

Within BPTT the error is back-propagated from the last to the first timestep, while unrolling all the timesteps. This allows calculating the error for each timestep, which allows updating the weights. Note that BPTT can be computationally expensive when you have a high number of timesteps. For this reason we decided to apply the Window method (see "Window method" part) and reduce the number of timesteps.

5.3 Long Short-Term Memory (LSTM)

Our first RNN uses Long short-term memory units, which are powerful and increasingly popular [7] models for learning from sequence data. They effectively model varying length sequences and capture long range dependencies.

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between. LSTM's enable RNN's to remember their inputs over a long period of time. This is because LSTM's contain their information in a memory, this memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information, based on the importance it assigns to the information. The assigning of importance

happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not.

5.4 Gated Recurrent Units (GRU)

For our second RNN we implemented another approach by using Gated Recurrent Units as a gating mechanism for the reason that they have shown to exhibit better performance on smaller datasets.

The Gated Recurrent Units (GRU) is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM, the difference is that GRU's don't use the cell state and use the hidden state to transfer information. In addition, it also has a less complex structure, and therefore, it is more efficient.

6 Classification with CNN - Keras

6.1 Keras

Keras is an open source neural network library written in Python, designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Furthermore, it allows the same code to run on CPU or on GPU, seamlessly. User-friendly API which makes it easy to quickly prototype deep learning models.

Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier. See an example of images of digits in Figure 6.

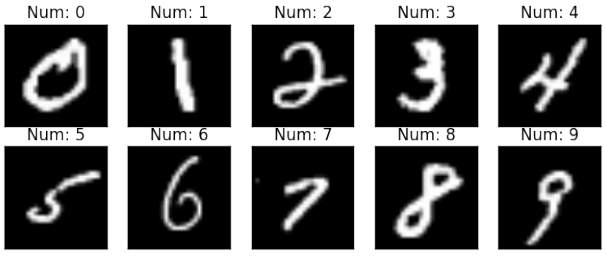

Figure 6: Example of Keras.

In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

6.2 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery where the model learns an internal representation of a two-dimensional input, in a process referred to as feature learning.

This same process can be harnessed on one-dimensional sequences of data [9] [8] , such as we have done in the notebook for signal classification.

They are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing.

The benefit of using CNNs for sequence classification is that they can learn from the raw time series data directly, and in turn do not require domain expertise to manually engineer input features. The model can learn an internal representation of the time series data and ideally achieve comparable performance to models fit on a version of the dataset with engineered features. This independence from prior knowledge and human effort in feature design is a major advantage.

In this approach we decided to use Keras as an alternative to Tensorflow.

7 Results - Analyze and contrast

In this chapter we will analyze and contrast the obtained results.

7.1 RNN (LSTM vs. GRU)

Analysing the results obtained from the LSTM and GRU executions, we can observe that they result in similar ROC AUC scores. The performance of GRU is on par with LSTM, but it is computationally more efficient, because it lacks of memory unit and therefore has a less complex structure.

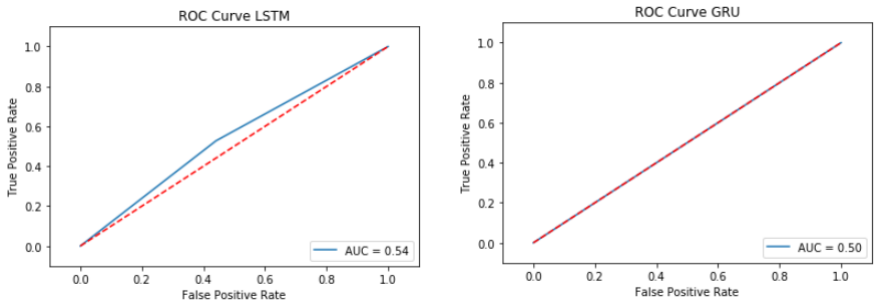

Figure 7: ROC AUC scores of LSTM and GRU executions

We can also see that ROC AUC scores have some peak values where it reaches a higher score and after that starts to decrease. From our point of view, in order to get most from the model an early stop should be applied when a peak is detected to ensure we end up with the model that gives us the best results.

7.2 CNN

The overall accuracy of the model using CNN is pretty decent. Nevertheless, as we have repeated the execution several times we should choose as our final model the one that has reached a higher accuracy value from the total amount of experiments.

```
>#1: 48.571
>#2: 51.429
>#3: 51.429
>#4: 67.143
>#5: 72.857
>#6: 70.000
>#7: 48.571
>#8: 78.571
>#9: 51.429
>#10: 77.143
[48.571428656578064, 51.428574323654175, 51.428574323654175, 67.14285612106
323, 72.85714149475098, 69.9999988079071, 48.571428656578064, 78.5714268684
3872, 51.428574323654175, 77.14285850524902]
Accuracy: 61.714% (+/-11.863)
```

Figure 8: Accuracy obtained with CNN

We have observed that when the model gives us an accuracy of 48.571 or 51.429 is because the model has predicted that all the labels are from class 0 or class 1. This proves that sometimes the model can predict random values and make the accuracy decrease. It behaves the same way as when

someone who is taking an A/B test decides to choose always A or always B to have a 50/50 chance of succeeding.

7.3 Comparison with the contest results

As it can be observed in the contest results [4] for the Data set I [Tübingen], almost no one used neural networks to solve the problem, therefore, we consider ours was a pretty innovative approach. In comparison to the results of the contestant teams, we can see that we are above the mean of accuracy obtained by other contestants, taking as our final model the one that has reached a higher accuracy value from the total amount of experiments executed on the CNN.

8 Conclusion

We can observe, that in the context of signal classification, the CNNs perform better than RNNs when the time window is short (3 seconds in our case). This happens because remembering important characteristics about the input does not add much value to the model, and therefore, the results do not improve.

In conclusion, from the results obtained during this project, we can infere that not in all sequential data is preferable to use RNNs, specifically LSTM and GRU.

9 Implementation

All the project steps were implemented in Python. We used Scipy for loading the data and Tensorflow, Sklearn, Numpy and Keras for the implementation of the neural networks. We illustrate how the implementation works in the Python notebook `Project42-Scola-Goni-Notebook.ipynb`. Inside of the compressed file that we submitted, the Jupyter Notebook, this report and a txt file can be found. The txt file contains the instructions to download the dataset, as it was too hefty to upload with the rest of the work.

References

[1] Hill NJ1, Gupta D, Brunner P, Gunduz A, Adamo MA, Ritaccio A, Schalk G. Recording human electrocorticographic (ECoG) signals for neuroscientific research and real-time functional cortical mapping.

[2] Description of the problems, datasets, and approaches of other teams: http://www.academia.edu/7971741/The_BCI_Competition_III_Validating_Alternative_Approaches_to_Actual_BCI_Problems

[3] Datasets of the competition: http://www.bbci.de/competition/iii/#datasets

[4] BCI results: http://www.bbci.de/competition/iii/results/

[5] Opałka S, Stasiak B, Szajerman D, Wojciechowski A. Multi-Channel Convolutional Neural Networks Architecture Feeding for Effective EEG Mental Tasks Classification. Sensors (Basel). 2018;18(10):3451. Published 2018 Oct 14. doi:10.3390/s18103451

[6] F. Lotte, M. Congedo, A. Lecuyer, F. Lamarche, and B. Arnaldi. A review of classification algorithms for EEG-based brain–computer interfaces. Journal of Neural Engineering, 4:R1–R13, 2007

[7] Zachary Chase Lipton, David C. Kale, Charles Elkan, Randall C. Wetzel. Learning to Diagnose with LSTM Recurrent Neural Networks. 13 Aug 2018. abs/1511.03677

[8] Kiranyaz, Serkan & Ince, Turker & Gabbouj, Moncef. (2015). Real-Time Patient-Specific ECG Classification by 1D Convolutional Neural Networks. IEEE transactions on bio-medical engineering. 63. 10.1109/TBME.2015.2468589.

[9] Li, Dan & Zhang, Jianxin & Zhang, Qiang & Wei, Xiaopeng. (2017). Classification of ECG signals based on 1D convolution neural network. 1-6. 10.1109/HealthCom.2017.8210784.