

# Deep Learning Mini-Project 1 Report

## Prediction of Finger Movements from EEG Recordings

Group 78\*

Caterina Bigoni, Luca Pegolotti and Nicolò Ripamonti

18 May 2018

### 1 Introduction

The goal of this project is to design and train a Deep Neural Network to analyse EEG data. In particular, we perform a two-class classification of EEG time signals collected from one patient to predict the direction of her upcoming finger movements (left vs. right). The dataset, known as dataset VI of the “BCI Competition II” [1], is composed of 416 recordings sampled at a frequency of 1000 Hz for 28 channels (i.e., electrodes placed at different locations on the skull). The dataset is split in training and test sets of 316 and 100 samples, respectively.

This report is organised as follows: we will briefly present and visualise the datasets in Section 2, where we introduce the suggested down-sampling strategy and a pre-processing algorithm to reduce noise in the signals. Then, in Section 3, we present four different models and the corresponding results obtained on the cross-validation set. Here we also include a data-augmentation strategy to increase the number of samples and hopefully reduce the generalisation error. Of these, the one that performs best is selected for further tests in Section 4. In particular, we present an optimisation setup to select the most performing hyper-parameters. The set of parameters that gives the smallest error on the cross-validation set is selected and finally tested on the test set. Final conclusions and discussions are provided in Section 5.

### 2 Data Visualisation

Before rushing at adding extra layers or more neurons to increase the complexity of our model, we take a look at the given data. Figure 1 shows the average over all samples of the training and test datasets for three randomly chosen channels, i.e., channels 3, 10 and 25. The averages are shown separately for the two-classes, i.e., left and right, and for both the signals sampled at 1000 Hz (in red and blue) and its down-sampled signals (in black and magenta). We observe that the down-sampled signals present a less noisy average with respect to the 1000 Hz signals. Moreover, we remark that the means over the test and train dataset are very different and often the means over the test dataset are more noisy. This behaviour is observed, although with different intensity, for all 28 channels. The extra noise in the test dataset can be justified by observing that the averages were not performed on equally numerous datasets: 316 samples and 100 samples in the training and test datasets, respectively. Nevertheless, it seems impossible to identify by human eye a difference in the behaviour of signals classified as 0 or 1. Even if we understand that the average is not a sufficient meaningful quantity to describe a complex dataset, the high variability of data makes us believe that this apparently simple two-class 1D classification problem, will instead be quite challenging. To support this idea, Figure 2 shows the behaviour of three randomly chosen samples with frequency 100 Hz for each class (i.e., left and right) for both the training and test dataset. Immediately, we can see that both the training and test

---

\*As agreed with Dr. F. Fleuret, L. Pegolotti has collaborated with group 78 for Project 1 and with group 79, with M. Martin, for Project 2. C. Bigoni and N. Ripamonti have worked together on both projects.

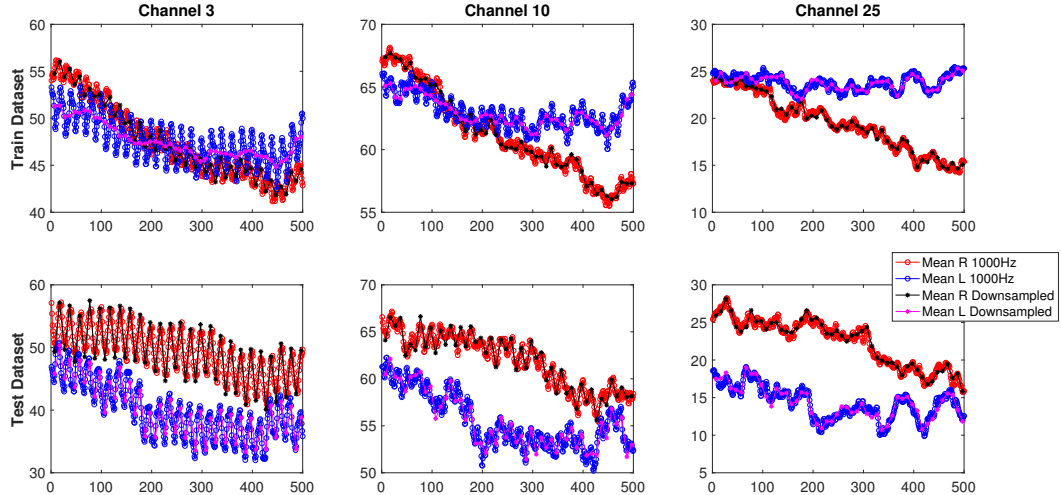


Figure 1: Comparison of training and test datasets averages per channel over 316 and 100 samples, respectively. Three channels (i.e., 3, 10 and 25) are randomly chosen, but the results generalise to all. Each plot shows the mean of the samples with output 0 (i.e., classified as “right”) and those with output 1 (i.e., classified as “left”). The means of the signals sampled with 1000 Hz are plotted in red and blue, while those of the signals down-sampled to 100 Hz are plotted in black and magenta. For some channels (e.g., channel 3) the down-sampled averages present a lower variability than those for the signals sampled at 1000 Hz. In all cases, the averages of the training set are distinctively different from those of the test set.

selected samples are all very noisy and distant from their corresponding average. As expected, we can not recognize a pattern for the signals corresponding to the left or right classification.

Finally, driven by the results already published for this dataset [1] and supported by other studies (see e.g. [4]), we propose a pre-processing strategy to reduce both the generalisation error and the size of the model needed to fit the training set. Indeed, by reducing the noise in the data, i.e., by increasing the signal-to-noise ratio, we aim at reducing the amount of variation that the model has to take into account and thus hope that a simpler model can better classify our data [3]. A *filtering preprocessing* technique is therefore applied to our final model, as explained in Section 4. In particular, we use the 1D Savitzky-Golay filter [2] to smooth the data without greatly distorting the signal. By a convolution, the filter fits successive subsets of adjacent data points in a low-degree polynomial using the linear least squares method. In our tests, we use subsets of 7 points to be fitted in a polynomial of order 3.

### 3 Considered models

In this section, we describe some of the models we tested on the EEG signals and summarise their performance in terms of percentage error on a validation dataset. Each of these models is tested using both the standard down-sampled dataset of size 316 and an *augmented dataset*. The latter is obtained by pre-processing the dataset to reduce the generalisation error [3]. In particular, we exploit the 1000 Hz dataset to extract not one, but 10 signals with frequency 100 Hz. In this way, our training dataset has the following dimensions: 3160 samples, 28 channels and 50 time-steps.

In the following, we provide a small description of each model we considered. The diagrams representing the graph structures show the dimensions of the input and output tensors of each module composing the networks except for the non-linear nodes because they do not modify the dimensionality of the inputs. In all diagrams,  $n$  is the number of input samples. For all models, in order to reduce the

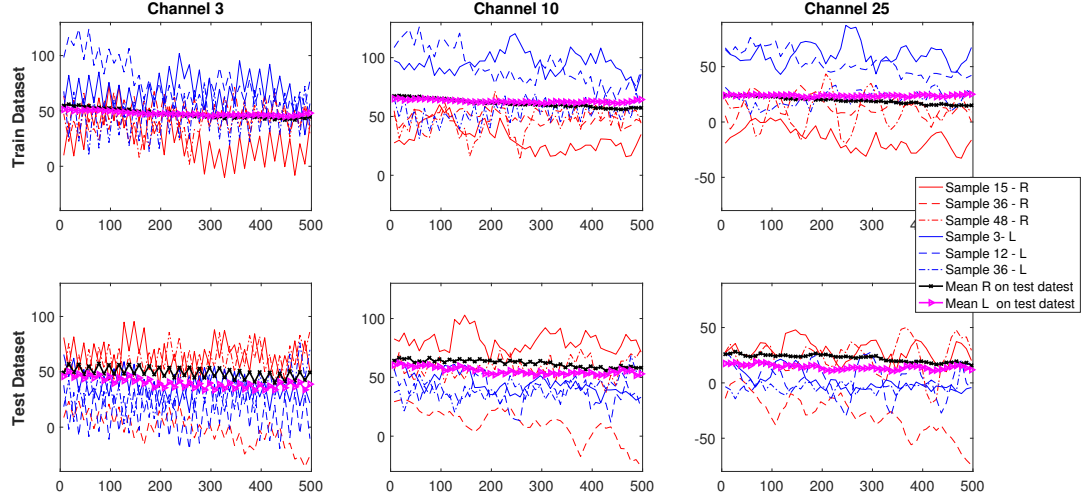
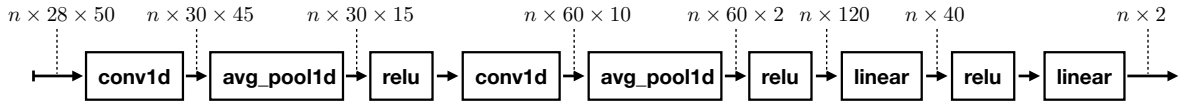


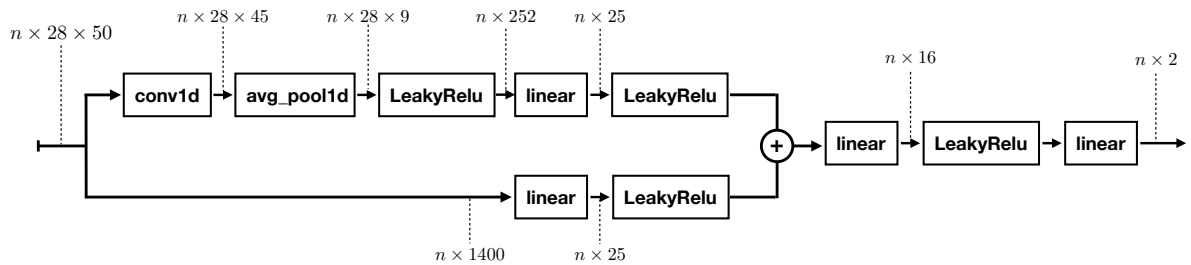
Figure 2: Plot of few randomly chosen samples from training and test datasets for channels 3, 10 and 25. Samples classified as 0 (i.e., “right”) are plotted in red, while samples classified as 1 (i.e., “left”) are plotted in blue. The down-sampled averages corresponding to the training and test dataset are also shown in black. These plots confirm that the data have a high variability.

discrepancies, we normalise the training dataset using its mean and standard deviation for all channels. The same values are used to normalise the test and cross-validation datasets.

- **Linear perceptron (M1).** The simplest model one could devise for a classification problem is the linear perceptron: given an input tensor  $x$ , the output is computed by the model as  $y_{ij} = \sum_k x_{ik} A_{jk} + b_j$ , where  $A$  is the matrix of weights, and  $b$  is the bias vector.
- **Simple convolutional linear network (M2).** TODO



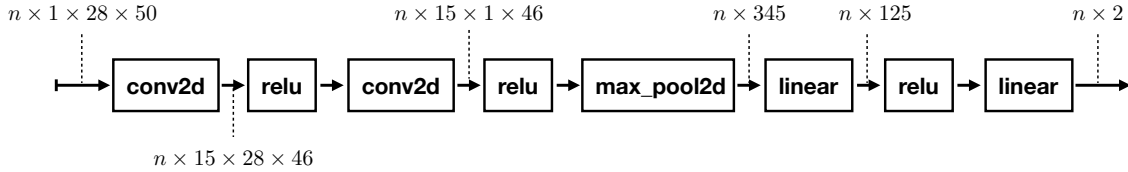
- **Multichannel deep convolutional neural network + linear perceptron (M3).** The convolutional layer in this model is such that different masks are applied to each channel; at the implementation level, this is achieved by setting the `groups` parameter equal to 28 in the constructor of the model. Convolutions of this kind are employed to study uni-dimensional time-series multichannel also in [5].



Trial	non-augmented dataset				augmented dataset			
	M1	M2	M3	M4	M1	M2	M3	M4
1	27	34	32	32	23	37	36	35
2	27	29	33	25	24	32	36	37
3	29	28	30	24	24	33	35	34
4	30	23	29	29	24	34	37	34
5	26	28	30	33	26	30	34	33
6	27	29	32	26	26	33	40	33
7	26	29	33	29	25	31	34	32
8	27	27	29	21	24	32	44	29
9	29	31	30	29	24	32	39	37
10	26	32	31	22	24	32	36	35
Best	26	23	28	21	23	28	34	29
Average	27.4	29.0	30.9	27.0	24.4	32.6	37.1	32.9

Table 1: Validation errors for the linear perceptron (M1), the simple convolutional neural network (M2), the multichannel deep convolutional neural network combined with the linear perceptron (M3) and the shallow convolutional neural network (M4).

- **Shallow convolutional neural network (M4).** In this network, the first and the second convolutions operate in the dimension of the time samples and the channels, respectively. We took inspiration from [4] for the design of this model.



We use the stochastic ADAM for the optimisation step, with batch size equal to 100 or to 1000 for the standard and augmented training datasets, respectively. The learning rate  $\varepsilon$  is equal to  $10^{-1}$  for the linear perceptron and  $10^{-3}$  for the other three models. The models are optimised according to the cross entropy loss function.

Table 1 shows the performance of the four models in terms of percentage error on a cross-validation dataset (obtained by randomly choosing 10% of the samples of the train dataset) after 1000 epochs of the stochastic gradient descent. Surprisingly, the linear perceptron (M1) performs better than (M2) and (M3), leading to a smaller average error over all the runs and also to a smaller variance of the result. In the non-augmented dataset, however, the model that ensures a smaller minimum error and a smaller error on average is the shallow convolutional neural network (M4).

Augmenting the dataset leads to a slightly smaller error for M1, but to a considerable loss of accuracy for the other three models. We suppose that, with a different choice of parameters (e.g., size of convolutional and pool nodes, kernels), the advantage of using a larger datasets could be exploited also for M2, M3 and M4.

Since the shallow convolutional network (M4) showed the most promising results in these preliminary test runs, in the following section we focus on optimising this particular network. It is important to notice that our results are greatly determined by the choices of parameters for each of the networks we considered; it is possible that other combinations for M2 and M3 could lead to smaller errors.

## 4 Hyper-parameter optimisation

The validation error, as function of the number of epochs, shows the typical U-shape (i.e., the validation error slowly grows after reaching a minimum) which implies that the considered models are affected by overfitting [3]. Additionally, the significant difference between the models performances on the train and validation sets confirms this hypothesis. [Non avendo mai detto quant'è la train vuoi commentare] Although the substantial noise in the datasets is probably the main cause behind overfitting, we have adopted different strategies to solve this problem. First, by manually tuning the sizes of the linear layers in M4, we observe that they play an important role in terms of overfitting and the best results are obtained when the number of neurons lies in the interval  $[100, 150]$ . This interval is optimal as it takes into account a direct tradeoff between overfitting and model complexity: smaller layers than the ones in the prescribed interval are not accurate enough and larger layers lead to an even more evident overfitting effect. The second approach consists in the addition of a parameter penalty norm  $L_2$ , which is obtained by passing the penalty constant to the optimiser object of the Pytorch library. Finally, dropout layers are introduced before linear layers to prevent complex co-adaptions and hence giving a network that generalises better.

After identifying the most promising model in Section 3 and choosing the regularisation strategy, we proceed with hyper-parameters optimisation to further improve the performance of the network (see `hyperoptimization.py` for the implementation). Given the shortage of available computational resources, we adopted a random search approach for the exploration of the parameter space using an ansatz on the lower- and upper-bound of each parameter [at intermediate steps – cosa vuol dire?]. We consider three groups of parameters (architecture, training and regularisation) and their intervals are listed in Table 2. Architecture parameters are related to capacity of the network and control the size of weights and biases of hidden layers (see Figure (put reference)). As mentioned in the previous section, the ADAM algorithm is adopted to optimise the network with a stochastic mini-batch procedure to update the weights and biases. Moreover, instead of considering a constant learning rate  $\varepsilon$ , we adopt a geometric progression: the initial learning rate  $\varepsilon_0$  is multiplied by the scaling rate  $d$  once every 30 epochs. The mini-batch size together with  $\varepsilon$  and  $d$  are the training hyper-parameters we aim at optimizing. Finally, regarding regularisation, both the dropout probability and the  $L_2$  penalty constant are treated as hyper-parameters.

This set of 9 hyper-parameters is used for optimisation in three different scenarios:

- the *regular dataset*, i.e. the down-sampled one with 50 time steps per channel;
- the *augmented dataset*, as described in the previous section;
- the *smoothed dataset* obtained from the regular set by applying the Savitzky-Golay filter to each signal.

Di COME SONO ANDATI A FINIRE CIASCUNO DI QUESTI E POI CONCLUSIONI

## 5 Conclusion

## References

- [1] BCI Competition II. <http://www.bbcii.de/competition/ii/>.
- [2] Savitzky-Golay Filter. [https://en.wikipedia.org/wiki/Savitzky-Golay\\_filter](https://en.wikipedia.org/wiki/Savitzky-Golay_filter).
- [3] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [4] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggenberger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human eeg. *arXiv preprint arXiv:1703.05051*, 2017.

<u>Group</u>	<u>Type</u>	<u>Interval</u>
Architecture parameters	Output channels: First convolution	[10, 80]
	Output channels: Second convolution	[10, 80]
	Kernel size: First convolution	[3, 13]
	Output size: Hidden linear layer	[100, 150]
Training parameters	Initial learning rate $\varepsilon$	$[10^{-5}, 10^{-2}]$
	Scaling rate $d$	[0.7, 0.95]
	Batch size %	[10, 50]
Regularization parameters	Dropout probability	[0.2, 0.5]
	$L_2$ penalty constant	$[10^{-6}, 10^{-1}]$

Table 2: Parameters intervals adopted in the hyperparameter optimization step. The reported intervals have been obtained as subinterval of initial guesses, cropped after we noticed that for some values convergence was not obtained.

- [5] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.