Deep Learning Mini-Project 1

Pierre-Antoine Desplaces, Raphaël Steinmann
Department of Computer Science, EPFL, Switzerland

***Abstract*—This report presents how to build a deep learning classifier to predict finger movement from Electroencephalography (EEG) recordings. The results of our experiment indicate that a model with 1D-convolutional layers and a dropout layer obtains the best performance.**

I. Introduction

In this project, our goal is to find the best model to fit a real-world dataset from BCI. This data set composed of a training set of 316 recordings, and a test set of a 100. Each recording has 14'000 features representing the measures of 28 EEG channels over 500 timesteps (0.5 seconds sampled at 1Khz). The class targets correspond to the direction of the finger movement following each recording, marked as 0 for left and 1 for right. This is a standard two-class classification problem.

II. Data processing

First of all, we normalized the data by subtracting the mean of each specific channel over the whole dataset and dividing it by its standard deviation. By specific channel, we mean that we averaged each EEG sensor over all its measures (500 times the number of recordings) and we substracted that value to the same measures it was calculated with. We did the same thing with the standard deviation.

Considering the high dimensionality of the data compared to the low number of samples, we chose to balance them by augmenting the dataset while reducing the number of features. To do so, we resampled the original data at 100Hz by slicing each data point into 10 different recordings of 50 equally distant measures, and thus multiplying the size of our dataset by 10 and reducing its dimension by the same factor. Each slice has the same target as the recording it was sampled from.

Finally, we randomly split the training set into a smaller training set of 2500 recordings and a validation set of 660 (roughly a 80/20% split), in order to use the validation to tune the parameters of our models.

The metric used to compare the efficiency of each model is the test error rate : the number of misclassified test samples over the total size of the test set.

III. Methodology

A. Regressions

As a first step, we implement a few baseline methods. These include linear regression, ridge regression and logistic regression. We used the `scikit-learn` library to implement them. As expected, each one performed better than the previous one : ridge regression uses regularization over linear regression, and logistic regression is more adapted to the task of binary classification.

The linear regression yields a test error of 34.9%, the ridge regression 30.4% and the logistic regression 24.3%. These results can be seen compiled in table I in the final part of the report.

B. Networks

The `pytorch` library was used to implement all the following models.

As this is a classification task, the most indicated loss criterion is the logistic loss, also called cross-entropy loss.

For the optimizer, we initially compared SGD and Adam and chose Adam because its adaptive learning rate makes the model more robust. The only parameter we tuned is the learning rate, varying at a linear pace between 0.001 and 0.01.

We used the hyperbolic tangent (tanh) activation function after every layer, as it is known to perform better on convolution layers than ReLU.

For training, we tested different sizes of mini-batches, and ended up using 250. Considering our training set is 10 times larger, this value allowed for the best balance between performance and accuracy.

To determine the optimal number of epoch and learning rate, we trained each model for 500 epochs and computed their error rate on the validation set every 5 epochs. We did so for every different value of learning rate, and we kept both hyper parameters of the best performance. In case of equality, we chose the one with the smallest number of epochs (quicker training) and the smallest learning rate (safer training).

All of these parameters and the test error they resulted in are presented in table II.

1) Multi-Layer Perceptron: As a first attempt to design a neural network, we start with a simple architecture of a Multi-Layer Perceptron : it has 1400 input units, two output units and two hidden layers of 140 and 28 units. This basic model obtained a test error of 31.2%, inferior to the previous baseline regression models.

2) Convolutional Neural Network: Considering the 2D format of the data samples, it is standard to use convolutional layers. The design of our basic CNN is shown on figure 1.

This first attempt resulted in an unconvincing test error of 32%.

3) 1D-Convolutional Neural Network: If we take a closer look at the nature and format of our data, we know these are not images, but temporal data. The ordering of the channels is irrelevant and thus combining them probably makes little sense. Considering that, we changed our 2D convolution layers to 1D convolution layer, and the same thing for our max-pooling. We redesigned the layers a little and obtained the architecture represented on figure 2. This allowed us to reach a test error of 23.8%, which is the first sign of improvement over the logistic regression

4) Dropout Layer: After a few runs of our last network, we observed a training error of 0%. This is a sign of overfitting. In order to mitigate that effect, we add to that previous model a dropout layer after the first fully connected layer. We kept the parameter of 0.5 as it yielded the most consistent results over different runs. This technique allowed us to obtain the best performance out of all the models on this dataset, with a 20.6% test error

5) Batch Normalization: Although we normalized our data during the pre-processing, it might prove useful to use batch normalization between layers to ensure a smoother optimization. In that idea, we normalized the output of every layer in the model from figure 2. This did not reach the same performance as the model without batch normalization, only 24.5%.

6) Both dropout and normalization: Combining both layers did not achieve better results than only with the dropout.

IV. Results and discussion

Our two first attempt at a neural network with the MLP and the 2D-CNN, interestingly, show worse results than simpler linear models, like the logistic regression. We then reflected more on the data itself and adapted the architecture to the input, by using 1D convolution filters to use the temporal aspect of the recording while not mixing the different EEG channels. This instantly gave better results than the previous models.

After this, we noticed that the main problem with such little amount of training data, is to overfit it and completely miss its general structure. Adding a dropout layer after the first fully connected layer helped to lessen this effect and yielded our best results.

We then unsuccessfully tried to improve these by using batch normalization.

One thing we could have done with bigger computational power would be to run all the models several times and average their results or even do cross-validation to ensure better and more consistent performance.

	Linear	Ridge	Logistic
Test error rate (%)	34.9%	30.4%	**24.3%**

Table I
Results for regression models

	MLP	CNN	1D-CNN	w/ Dropout	w/ Batch Norm	w/ both
# of epochs	30	20	60	45	40	50
Learning rate	0.001	0.0025	0.005	0.001	0.0025	0.0025
Error rate (%)	31.2	32	23.8	**20.6**	24.5	22.6

Table II
Results for our models

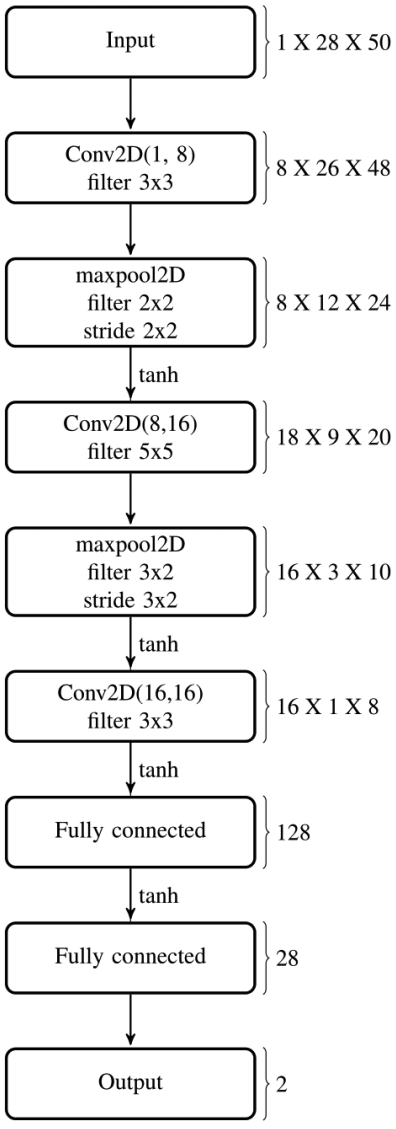

Figure 1. Architecture of the first CNN

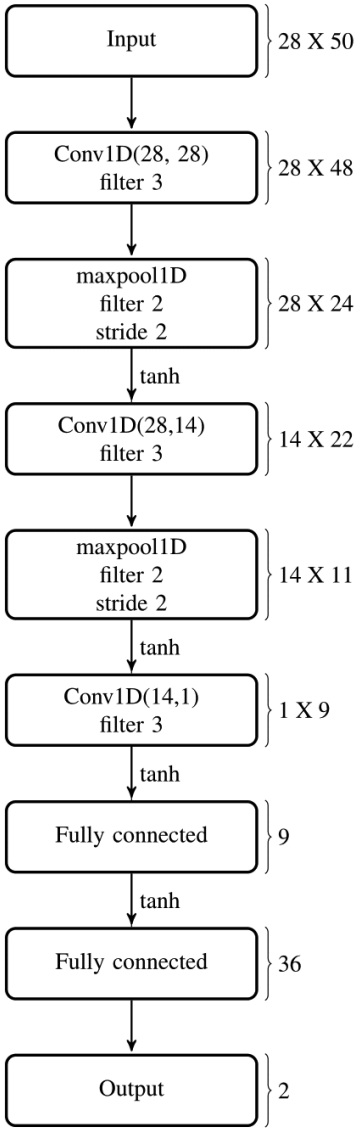

Figure 2. Architecture of the CNN with 1D filters