ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Deep Learning
EE-559

Prediction of finger movements from EEG recordings

Groupe 36:
Victor Faraut
Martial Bernard-Michel
Mickaël Bressieux

May 11th 2018

1 Introduction

An electroencephalography (EEG) is a non-invasive method aiming to measure the electrical activity of the brain by putting a set of electrodes on a subject's scalp. It can be used for the analysis of different phenomenons, like epilepsy, coma or sleep disorders. In this project, the data given to analyze are resulting from an experience where the subject had to move either a finger on his right or his left hand. The goal is to predict the most accurately possible the laterality of the upcoming finger movement, and this 130 $[ms]$ before it happens. In order to classify the data, different types of deep learning networks will be tested to create a model able to reach the objectives.

2 Data-set

The training data consists of 316 experiments lasting 0.5 $[s]$, each composed of 28 electrodes measuring the subject's brain voltage fluctuations as he chooses the right or left finger. There is also a test set of 100 experiments using the same setup for testing purposes.

Two kinds of data-set are available: one with a sampling frequency of $100[Hz]$ and another one with a frequency of $1[kHz]$. The first one is chosen with the reasoning that if the trained model can learn on the less detailed version of the data set it will be less likely to encounter problems dealing with the more exhaustive one.

3 Pre-treatement of the data

First a set of data is extracted from the 316 data to build a validation set. This is very important as the model should be accurate on a validation set before doing any testing on a real test set. This is done be removing a random 16 value of the train set and using them in the validation set.

As 300 experiments is not a huge amount of data, we choose to increase this number by creating "fake"data from the real ones. To do so, a Gaussian noise was created and added to two copies of the initial dataset to increase its initial size by a factor 3. Those new data still have the same general features as the original ones and should help the model to focus on some low-frequency features when training. The validation set is not increased because there is no need for fake data to validate our model. At the end of this treatment we remove the mean as well as the std from all the data-set independently.

4 Optimizer and loss function

The choice of a good optimizer is key in deep learning, and particularly in terms of time of convergence. For the optimizer selection, the stochastic gradient descent (SGD) was first used and then the optimizer was finally chosen to be ADAM. This was interesting as we saw a clear difference in the way the algorithm learns with one optimizer or another. For SGD we have a reduction lot slower in the loss function as opposed to the one with ADAM for the same learning rate.

For the loss function, a natural choice for classification could be the cross-entropy loss. Indeed, unlike the mean-square error loss, it rewards the strong scores when they are assigned to the right class in the output vector.

5 Linear model

To begin with, here is a simple linear model. To use it, the data need to be converted from a sort of 2D matrix (28 channels and 50 time value) to a 1D vector. To do so the two dimensions can be mixed together and fed through the neural network as only one dimension. For the number of units in each hidden layers, it was chosen to incrementally reduce the dimension size and the implemented values were found by experimenting with the network. At the end of the model, the size is chosen to fit the number of classes.

The linear model could also be composed of some dropout function between the layers. This should improve the test error by disabling some weights during the training. Of course they need to be disabled for the testing of the forward pass. This function helps to reduce over-fitting and generally leads to better results.

Function	Parameters	Input size	Output size	Function	Parameters	Input size	Output size
Linear		28*50	1000	Linear		28*50	1000
ReLU		—	—	ReLU		—	—
Linear		1000	200	Linear		1000	200
ReLU		—	—	ReLU		—	—
Linear		200	20	Dropout	p = 0.5		
ReLU		—	—	Linear		200	20
Linear		20	2	Relu		—	—
				Droupout	p = 0.5		
				Linear		20	2

Table 1: The linear models used during this project

At the end we got our best model with ADAM optimizer and a Learning rate of $1e^{-4}$ and 50 epochs

For the "without dropout" model:
train error AVG 0.00% validation set error AVG 30.94% test error AVG 28.90% train error BEST 0.00% validation set error BEST 12.50% test error BEST 25.00%

For the "with dropout" model:
train error AVG 0.31% validation set error AVG 30.62% test error AVG 28.65% train error BEST 0.00% validation set error BEST 6.25% test error BEST 25.00%

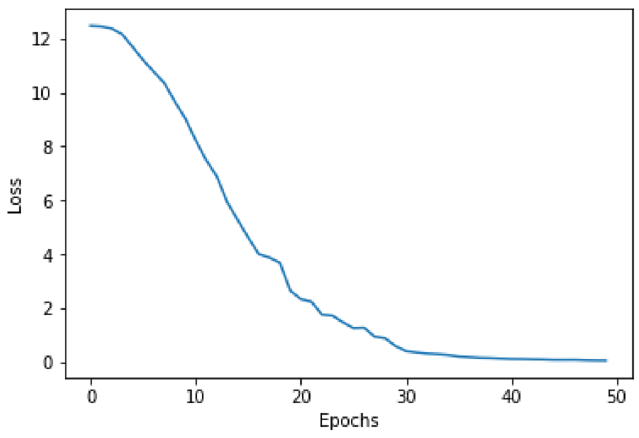

Figure 1: Plot of the evolution of the loss function during one of the model generation using the network given in the table 1

So the basic linear model is able to extract some features of the data but not all of them correctly and the accuracy can be improved.

6 Convolutional model

The advantages of conventional layers is that they are able to handle large-sized data while extracting the useful informations. A trained convolutional layer will apply to the input one or several kernels full of optimized parameters. This operation will filter the input in the best way to get the information needed to minimize the total loss function of the whole model.

For this project, different convolutional networks have been tested. The nature of the input allows to perform 2 types of convolutions:

1. a 1D convolution with its kernels moving along either the time or the voltage output of an electrode.
2. a 2D convolution with, as feature channel, a new dimension created with the function *unsqueeze()*.

The convolutional network using convolutions with 2D kernels did not show the best results. With a network of two 2D convolution and two linear layers, the generated models had an average test set error percentage of 33.14%. Some networks can achieve better results as it will be seen later in the report. Furthermore, this network was quite slow as 2D kernels usually imply a lot of parameters involved in the optimization.

The convolutional network giving among the best results was a simple network using two 1D convulational layers and two linear layers. It also contains two max pooling functions, which are useful functions for reducing the dimensionality of their input while preserving their structure. The two convolutions involved are applied along the time dimension.

Function	Parameters	Input	Output
Convolution 1D	kernel_size=3	28x50	32x48
Max pool 1D	kernel_size=2, stride=2	32x48	32x24
ReLu			
Convolution 1D	kernel_size=3	32x24	32x22
Max pool 1D	kernel_size=2, stride=2	32x22	32x11
ReLu			
Dropout	p=0.5		
Linear		32x11	50
Relu			
Dropout	p=0.5		
Linear		50	2

Table 2: The first convolutional model used during this project

The results of the models generated by this network are good. The test set error percentage is in average of 22.05% and at best of 18% over 20 training sessions. The results on the validation set were also good, with an average error percentage of 20.31%. The model was created with a learning rate of $1e-04$ and a number of epochs of 190. Those two hyperparameters were tuned in order to keep the convergence of the optimization while avoiding the over-fitting of the training set.

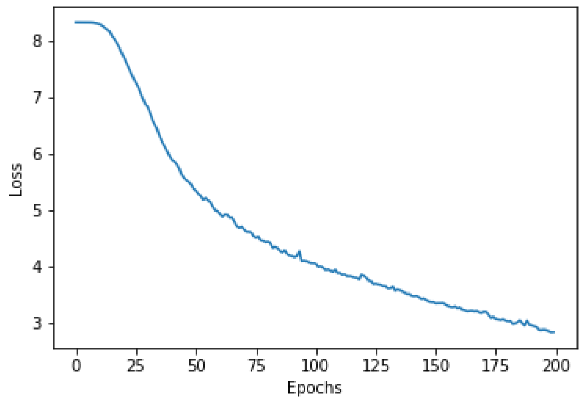

Figure 2: Plot of the evolution of the loss function during one of the model generation using network of table 2

Note that, on figure 2, the algorithm stops before a plateau is reached. This is done by tuning the learning rate and the number of epochs.

A lot of different convolutional networks have been tested before this model was chosen. Some were composed of more convolutional layers and/or more linear ones, but they resulted in comparable or worst results than the thinner network. Furthermore, those took an overall longer time to optimize even though their loss function converged for less epochs. The same conclusion seems to hold for convolution using bigger kernels, thus requiring more parameters: their results were not clearly better in terms of validation and tests percentages.

This fact does not seem logical, as a network which is given more parameters to tune should have more means to achieve the minimization of its loss function, thus achieving better results at this task. An explanation to these poor performances could be that there are not enough data to train our model with and that the network is reaching the limits of its performance.

Another network that was tried was a more complex version of the one described in table 2. This time it is composed of three 1D convolutional layers(still along the time dimension) and three linear layers. The different functions used are recorded in the following table.

Function	Parameters	Input	Output
Convolution 1D	kernel_size=3	28x50	64x48
Max pool 1D	kernel_size=2, stride=2	64x48	64x24
ReLu			
Convolution 1D	kernel_size=3	64x24	128x22
Max pool 1D	kernel_size=2, stride=2	128x22	128x11
ReLu			
Convolution 1D	kernel_size=4	128x11	256x8
Max pool 1D	kernel_size=2, stride=2	256x8	256x4
ReLu			
Dropout	p=0.5		
Linear		256x4	150
ReLu			
Dropout	p=0.5		
Linear		150	50
ReLu			
Dropout	p=0.5		
Linear		50	2

Table 3: Second convolutional model used in this project

The results of this second network are good, even though the generated models are less accurate and take a longer time to get computed than with the simple convolutional model. The average test error percentage is 23.80% while the best model computed showed an test error percentage of 18%.

7 Other ways

Given the time limitations for this report, we will only present those networks, but we tried more without better result (at least at first sight). It already takes a lot of time to test and fine tune parameters that it's not necessary to go everywhere with the testing. We decided to focus on more "basic" but easy to tune model for this project.

Other leads that were investigated during the project:

1. Increasing data size by computing the mean of some data with the same target to reduce noise and enhance some features
2. Doing more deep network with all the hyperparameters tuned
3. Using other methods than ReLU E.g. ELU, sigmax, tanh,....
4. Play with the parameters initialization. Less std or more, other initialization method
5. Use some learning rate scheduler with reduce-on-plateau or step base activation
6. Use different optimizer
7. Use a different loss function

8 Conclusion

This Mini project shows us a real example of use for a neural network and it makes us realize how hard it is to tune finely a network as well as how much time it takes to do it. There were a lot of different options at our disposal and we tried a lot of them. This fact led us to a wall of alternatives to try and we did not have the time and sometimes knowledge and understanding to apply them. This project taught us to be more methodical while trying things in such a large field and to not lose ourselves in testing. It also showed us how big this field is as well as how hard it is to master it.