EE 559 : Deep Learning
Mini-project I

Armand BOSCHIN Quentin REBJOCK Shengzhao LEI

***Abstract*—In this paper, we propose a model of classifier for finger movements from electroencephalography (EEG) recordings. The aim is to predict the laterality of finger movements (left of right) from the EEG recordings. This is a standard two-class classification problem. The structure of the paper is the following : after a short introduction in section I, we present the data set in section II, the algorithms used in section III and IV and eventually we discuss the results in section V. Particularly we show that one-dimensional convolutional neural networks seem to be more suited for the task than the recurrent models in spite of the temporal structure of the data.**

I. INTRODUCTION

The goal of this project is to build a classifier able to predict the laterality of finger movements based on EEG recordings. The two possible labels are either 0 or 1 for left or right. We use the accuracy as a performance measurement for our models.

II. DATA SET

The data comes in the form of 416 times series (316 for training and 100 for testing). Each time series is composed of 28 channels (EEG measured with 28 sensors) sampled at 1kHz for 0.5s ; that makes 500 values for each channel. Additionally, a downscaled data set with a sampling rate of 100Hz is available (that makes 50 values for each channel). We explain below how we made use of both sampling rates.

III. BASELINES

Before trying deep models, we tried to train usual models as baselines : random forest, logistic regression and support vector machines (SVM). Even if we don't expect them to give the best accuracies, it's important to know how well they perform to be able to judge the deep models. For all of them, the hyper-parameters were tuned using a grid search and k-fold cross-validation (with 5 folds) on the training set.

IV. ARTIFICIAL NEURAL NETWORK MODELS

In order to get better performances, we trained several neural network architectures. We can indeed expect much better accuracies, especially since the classification is not linear and since we expect the patterns to be complex in the time series. We then need a model with large capacity. Deep models seem appropriate for that.

A. General setup and tricks used to improve the models

We present here the general setup regarding the preprocessing of the data, the several tricks that were used, and the approach that we followed.

1) Data preprocessing: Real-world data is always incomplete, duplicated and contains outliers. Applying some data preprocessing techniques before training help to cope with those problems and can significantly improve the results by improving the quality of the data.

Normalization is a common data preprocessing method. It scales data into a specific range (usually smaller). The following assignment scales the data in the range $[0, 1]$:

$$X \leftarrow \frac{X - X_{min}}{X_{max} - X_{min}}$$

where X_{min} and X_{max} are respectively the minimum and the maximum of the original training data.

Standardization is maybe the most common preprocessing method. It scales the data distribution into a new one with mean 0 and standard deviation 1. It is particularly suitable if the data has an underlying Gaussian distribution as it makes fluctuations of different features comparable. That way it can increase the model performance and the convergence speed.

$$X = \frac{X - \mu}{\sigma}$$

where μ and σ are respectively the mean and the standard deviation of the original data (used for the training).

2) Data augmentation: One of the biggest problem encountered in this project is the small size of the training data (only 316 samples). As the limited amount of training data may lead to over-fitting (among other problems), we implemented several data augmentation techniques to increase the size of our training set. The goal was to find ways to diversify the data but maintain its characteristic patterns at the same time. We applied the following methods : down-sampling the 1kHz data set, adding noise, and cropping random windows.

Down-sampling As explained in the section V, we trained several models on both the 100Hz and the 1kHz data, and it appears that the later doesn't improve the results, it actually makes them even worse. We think that it comes from useless additional data points (noise) from which the network can't learn anything. Thus, we decided to use only down-sampled data (from 1kHz to 100 Hz). There are however several ways to do that. On the one hand, we can pick 50 points regularly separated from the 500 original ones, so the size of the new data set is 10 times bigger (3160 samples). On the other hand, we can divide the sequence of length 500 in 50 intervals of length 10 and randomly pick one point in each of these

intervals. In this case the separation is not regular but we can get a lot of more data (10^{50}).

However, that data augmentation technique did not perform very well. We suspect that this is because signals are often constant for long periods of time, many down-sampled sequence are very similar and it doesn't help the network to learn anything new. When splitting the data into a validation set, we noticed that the validation accuracy was very close to 100%. It came from the fact that the validation set is comparable to the training set.

Adding noise to the original data can help create new usable data points. We add a Gaussian noise of mean 0 and of standard deviation not too large so that the data is slightly modified while keeping its characteristic patterns.

Random crop Finally, it is possible to randomly crop a smaller window of size (e.g. 42 in a sequence of length 50). This allows us to build a couple of new data points out of a single one. The resulting shape of the data would thus be 28×42. In order to make predictions, we need however to crop the input as well and that is a loss of information. To cope with that we can crop the input sample many times, predict a label for each crop and eventually return the label using majority voting on all the crops.

Note that the two last data augmentation techniques presented (Gaussian noise and random crop) are made dynamically through the training process : when a data point is considered, a *GaussianNoise* and a *Crop1d* transformations are applied to it. PyTorch offers a simple way to do so and epoch after epoch, the data considered is not exactly the same (but the data set remains the same).

3) Dropout: Over-fitting is greatly reduced by randomly ignoring a fixed proportion of the hidden units at each training step. This can improve the generalization of the network by preventing the complex co-adaptations of certain features [1].

4) Learning rate decay: Reducing the learning rate can help the convergence of the training : when the parameters are close to a local minimum, having too a large learning rate makes the training unstable. Moreover, it appeared to decrease the variance of the models that we trained. To do so, we can multiply the learning rate by a factor γ in $]0, 1[$ after each epoch.

5) Cross validation: A proper way to tune the hyper-parameters of a model and to estimate its accuracy is to use cross validation (because tuning them using the test set is obviously cheating). Most of the time, we split the data in 5 folds, which is a good compromise between the pertinence of the estimations and the computation time.

B. Recurrent neural networks

1) Initial LSTM model: The first RNN model we tried is a simple LSTM model with a fully connected layer as output. Their memory makes them ideal to learn features from time series data. The strategy is the following : start from a simple model, make it over-fit and then use the tricks presented in section IV-A (data augmentation, dropout, cross-validation).

2) Convolution before the RNN: LSTM can however not take advantage of the fact that the data has 28 channels like the CNNs as it cannot capture inter-channel patterns. A possibility to cope with this limitation is to add a convolutional layer before the recurrent layer, to take advantage of both structures.

We then decided to move on to Convolutional Neural Networks (CNN) as it seems to currently be the best performing method (in the papers we read and the discussions we had with the teaching team).

C. Convolutional neural networks

Since each data point's shape is 28×50 (28 channels), it is natural to apply one dimensional convolutional neural networks.

1) Initial model: The global approach that we followed is the same as previously : start from a very simple model, make it over-fit the data, and then try to improve it gradually using tricks presented in IV-A.

The initial model is composed of a single one dimensional convolution, a pooling layer and two fully connected layers. The convolution is done on the temporal dimension. The parameters are the number of filters, the kernel size, the filter size, the number of neurons in hidden layers, the batch size, the number of epochs and the learning rate. As we cannot use a full cross-validation grid-search to tune all of them (it would take too much time), some of them were set using a exploratory approach.

2) Deepen the model: After using some of the preceding tricks to tune the parameters of the initial model (such as dropout rate, kernel size and numbers of filters) we tried to increase the capacity of the model to make it able to learn complicated patterns from the data. By adding a convolution layer (making the network deeper), we can expect a CNN to learn more complex patterns. However, the capacity increases fast with the number of layers and because of the small size of the training set, over-fitting is never far away. Thus, we try to design the CNN with two or three convolution layers top and then reduce over-fitting (using dropout and data augmentation).

V. Experiments and results

In the following results, validation accuracy means that it was computed using k-fold cross-validation ($k = 5$) on the training set and so it has nothing to do with the test set.

A. Baselines

We tried to train the baseline models on both data set (sampled with 100Hz and 1kHz frequencies) and it seemed that the additional data from the 1kHz data set was not improving the accuracy of the predictions : the accuracies were not significantly changing with the 1kHz frequencies and were actually even slightly worse. That is why, as explained in the section IV-A, we decided to keep only the 100Hz sampled data set to train these models.

1) Random Forest: For this model, we tune the number of estimators, the criterion (gini or entropy), the maximum number of features, the minimum sample leaf, and the maximal depth. It turns out that the model performs very poorly with a validation accuracy of $61.74 \pm 6.55\%$ with the following parameters :

Hyper-parameter	Value
Number of estimators	130
Criterion	Gini
Max. number of features	7
Min. sample leaf	16
Max. depth	8

Not only is the validation accuracy very low, but in addition its variance is very high.

2) Linear model: Regarding the logistic regression, only the regularization parameter C has to be tuned. After tuning this parameter, we achieved a validation accuracy of $72.16 \pm 1.33\%$. It is a great improvement compared to the last model, especially since the variance is now much lower.

3) Support vector machines: Finally, for the support vector machines model, we tuned the regularization parameter C and the kernel (RBF, linear, polynomial or sigmoid). The value $C = 0.0372$ with a linear kernel gave the best accuracy : $73.10 \pm 3.08\%$. The mean accuracy is slightly better than with the logistic regression, but the variance is also much higher.

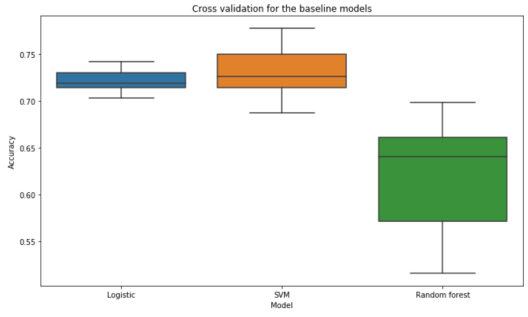

Fig. 1. Comparison of the baseline models' performances.

4) Comparison of the models: As the boxplots of figure 1 show, the SVM model has the highest average accuracy but its variance is larger than the one of the logistic regression model. Thus one could all the same choose the logistic regression to get a more stable model.

B. Recurrent neural networks

1) Initial LSTM model: The first RNN model we tried is a simple LSTM model with a fully connected layer and dropout (with probability 0.5). This model did not perform really well as it is long to train. Even with dropout and data augmentation, it over-fits and the resulting validation accuracy is bad ($\pm 65\%$). We could have improved it by further tuning the parameters but instead we decided to try another architecture with a convolution layer before the LSTM layer as explained before.

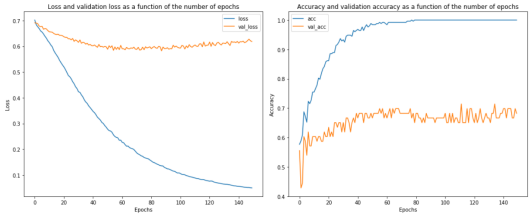

Fig. 2. Learning curve of the LSTM model (with dropout)

2) Convolution before the RNN: In order to use the channel structure of the time series, we added a convolution before the LSTM layer. It didn't improve the cross validation results a lot with a validation accuracy of $\pm 64\%$ (model E in table III).

C. Convolutional neural networks

1) Performances of the initial model: Even with only one single convolutional layer, the model overfits easily on the initial $316 \times 28 \times 50$ data set without using any trick from the section IV-A, as the figure 3 shows.

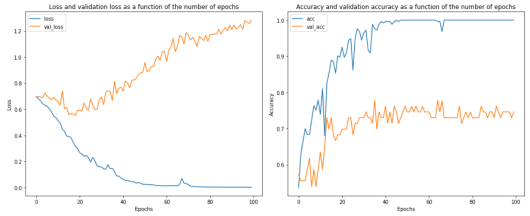

Fig. 3. Learning curves of the initial 1conv model with 20% validation split

We moreover noticed that there is no significant difference of performance when using normalization instead of standardization ; we chose to continue with standardization as it is more common.

Regarding the batch size, the smaller it is, the better the results are but the longer the training is. We chose to keep this value fixed to 16 for all the trainings, because it allows to train the models relatively quickly and we did not notice significant improvements by taking lower values.

In order to prevent this model from over-fitting we used dropout and data augmentation (adding noise, cropping). We first tuned the learning rate and the noise's standard deviation using cross validation (5 folds) as showed in the figure 4.

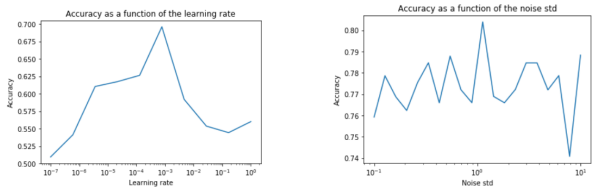

Fig. 4. Cross validation accuracies for respectively the learning rate and the noise standard deviation

The figure 5 shows the results of the cross validation for various configurations of dropout and data augmentation.

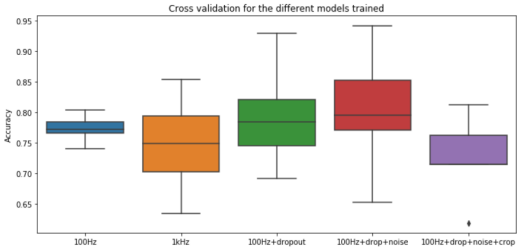

Fig. 5. Comparison of the performances of the initial model with several tricks added

Interpretation :

- 1 kHz data seems to slightly lower the average validation accuracy and increase the variance of the model a lot. We then decided to use the down sampled data only.
- adding dropout (rate of 0.5) after the convolution layer helps increase the validation accuracy
- adding Gaussian noise to the training data helps increase the validation accuracy but slightly increases the variance.
- augmenting data using random cropping decreases the performance which is surprising. We won't use this trick anymore.

The best model so far has one convolution, dropout, data augmentation in training using Gaussian noise. In order to reduce the variance of this model we then introduced the *Learning rate decay* trick (as explained in section IV-A). The decay has to be slow however. In the figure 6 ($\gamma = 0.8$ and $step = 10$) it appears that the validation accuracy is more stable with the learning rate decay. This is why we kept using this decay for the other models.

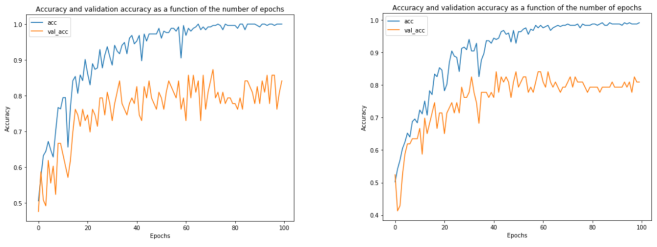

Fig. 6. Effect of the learning rate decay on the learning curves

The final one convolution model has a cross-validation (10 folds) accuracy of $81.30\% \pm 4.69\%$ (cf. model I in tables II and III).

Eventually, we note that even with dropout and data augmentation, over-fitting is only postponed and the learning curves still have the same profile as in figure 3. This makes us suspect that the model capacity is already high enough. We will confirm it in the following section on deeper models.

2) Two-dimensional convolutions: Even if two-dimensional convolutions don't seem adapted to this problem a priori, we tried to apply them viewing the data's shape as $316 \times 1 \times 28 \times 50$ (as if each sample was a 28*50 image with one channel). The results were however not very encouraging (accuracy around 70%) and we didn't go further in this direction.

3) Deepen the model: As explained earlier, in order to increase the capacity of the network, we added convolution layers to create a deeper network (up to 2 and 3 convolutions). This comes with even more regularization and data augmentation steps to prevent over-fitting.

After tuning the parameters with cross-validation (standard deviation of the Gaussian noise for data augmentation, learning rate in the optimization) we noticed that even if there is no improvement in the accuracy, the learning curves tend to be more stable. This is why we decided to keep this model as our final one. A description of this final model with three convolutions can be found in table I. The cross-validation (on training set) accuracy can be found at model K entry in table III ($81.05 \pm 6.75\%$).

After tuning the parameters and training the model using only the training set, we tested it on the test set (which is data that the model has never seen). The accuracy tends to be a little under what we obtained on cross-validation (around 76 % but unfortunately this fluctuates) and that makes us suspect that the train and test set are different from each other.

D. Comparison of the models

We finally summarize the various models in table II and their performance in table III.

As explained right above, the final model we retained is the deeper one with three convolution layers. We reported the learning curve of the final tuned model in fig 7. For this figure, the orange curve was plot using the test set in order to assess the performance of the model on data that it has not seen during tuning. This does not impact either the tuning or the training of the model (it's not cheating!).

TABLE I
CNN STRUCTURE

Layers	Parameters and Output
Input layers	28*50
Conv1dLayer	filter_size=5 padding='same' activation=ReLU output=64*50
PoolLayer	kernelsize=2 pool=MaxPooling output=64*25
Conv1dLayer	filter_size=3 padding='same' activation=ReLU output=64*25
PoolLayer	kernel_size=2 max_pool=2 output=64*12
Conv1dLayer	filter_size=3 padding='same' activa='ReLU' output=64*12
PoolLayer	kernel_size=2 max_pool=2 output=64*6
DropoutLayer	dropout_rate=0.2
FlattenLayer	input=384 neurons=32 output=32
DenseLayer	output=2

TABLE II
DESCRIPTION OF THE MODELS

Model	Dropout	Noise	Crop	Description
A	No	No	No	Random forest
B	No	No	No	Logistic regression
C	No	No	No	SVM
D	Yes	0.167	No	LSTM
E	Yes	0.167	No	1conv + LSTM
F	No	No	No	1conv NN
G	Yes	No	No	1conv NN
H	Yes	1.129	Yes (42)	1conv NN
I	Yes	1.129	No	Final 1conv NN
J	Yes	1.129	No	Two convolutions
K	Yes	1.129	No	Three convolutions

TABLE III
PERFORMANCE OF THE MODELS

Model	Accuracy
A	$61.74 \pm 6.55\%$
B	$72.16 \pm 1.33\%$
C	$73.10 \pm 3.08\%$
D	$63.51 \pm 10.30\%$
E	$64.32 \pm 8.46\%$
F	$76.57 \pm 7.63\%$
G	$79.14 \pm 7.63\%$
H	$74.37 \pm 9.66\%$
I	$81.30 \pm 4.69\%$
J	$80.05 \pm 4.90\%$
K	$81.05 \pm 6.75\%$

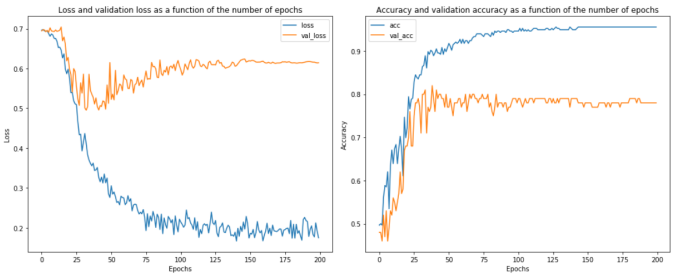

Fig. 7. Learning curve of our best model with the test set (in orange)

VI. CONCLUSION

To conclude, the main challenge we faced in this project was the lack of data. It proved difficult to prevent our models from over-fitting. We noted that the recurrent models do not naturally perform better than the convolution ones (in spite of the temporal structure of the data). The final best model we obtained has three convolution and even if it tends to over-fit faster than the one with a single convolution (because larger capacity) it seems to generalize better to the test set. Unfortunately we did not manage to lower significantly the variance of the model as it can be seen by running the `test.py` file provided.

REFERENCES

[1] Hinton, Geoffrey E and Srivastava, Nitish and Krizhevsky, Alex and Sutskever, Ilya and Salakhutdinov, Ruslan R. *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv preprint arXiv:1207.0580, 2012.

[2] Glorot, Xavier and Bengio, Yoshua. *Understanding the difficulty of training deep feedforward neural networks*. Proceedings of the thirteenth international conference on artificial intelligence and statistics.2010