Prediction of finger movements from EEG recordings

Sergii Shynkaruk, Roman Bachmann
EE-559 Deep Learning, EPFL, Switzerland
{sergii.shynkaruk, roman.bachmann}@epfl.ch

***Abstract*—In this report, we summarize our findings for the first project of the EPFL EE-559 Deep Learning course [1]. The goal of this project is to implement a neural network to predict the laterality of finger movement (left or right) from the EEG recording. Key aspects of the project are exploratory data analysis, feature processing, implementation of several deep learning network architectures, their performance analysis and accuracy assessment using k-fold cross-validation.**

I. Introduction

In the modern world, Electroencephalography (EEG) is one of the most efficient methods among known approaches of electrical brain activity monitoring. It is not superfluous to say that EEG brain activity detection during physical exercises (finger movements in our case) has a significant importance in the diagnosis of various types of brain dysfunction.

In-depth study of such problems has become feasible also because of the increasing computational power of computers to perform deep learning tasks. Indeed, the use of General Purpose GPU (GPGPU) computing allows deep learning algorithms execution time to be drastically decreased. In addition, finger movements prediction is a topic of a great interest, since principles of this two-class classification problem could be applied and extended to m-ary cases.

In this report, we provide an overview of the different deep learning approaches that we performed for the purpose of finger movement classification. In section V-A, we describe the Convolutional Neural Network model we built to perform the classification task. It showed us the main challenges with regards to the classification of such EEG data samples. Then we applied a LSTM method that we briefly present in Section V-B. Finally, our work about finger movement prediction highlights the fact that Recurrent Neural Networks (RNN) give good accuracy while processing time series data, and since Long Short-Term Memory Networks (LSTM) outperform RNN with long time series, our final choice of LSTMs are well-suited for our classification task.

II. Data exploration and feature processing

A. Input Data Analysis

We are provided with a set of 316 EEG recordings - train data, each of size 28 channels * 500 time samples. Every channel-trial of the train data is labeled with $\{0,1\} \rightarrow \{\text{left, right}\}$ finger movement accordingly. The testing data set is composed of another 100 EEG recordings. Our task is to classify testing data trials per channel to be a left or right finger movement.

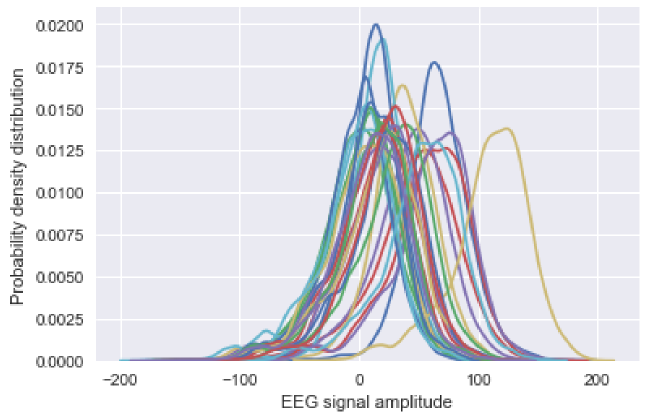


Figure 1. EEG data distribution. Each curve represents the data distribution obtained from a particular EEG sensor (28 sensors in total).

Plot 1 demonstrates that most data from all 28 EEG sensors are centered around 0, while some of them (for example sensor number 10) are shifted far from 0. In the corresponding visualization 2 of a single EEG data point, sensor 10 can be easily distinguished as a horizontal bright yellow line.

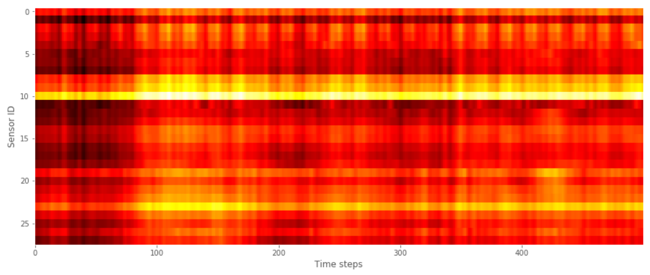


Figure 2. Heat map of one EEG recording showing the intensity of all sensors at each point in time.

III. Data Preparation

A. Standardization

From the data analysis, we take away that the data is not of very similar amplitude for each sensor. This results in the neural network needing larger weights for some sensor and smaller for others, which can present a problem for weight regularization. We therefore standardize the dataset feature-wise over all time steps by subtracting the mean and dividing the result by the standard deviation.

B. *Data Augmentation*

Our dataset has only 316 training and 100 testing data points. This size might be sufficient for training a shallow model, but neural networks have a tendency to overfit when the number of degrees of freedom is much greater compared to amount of input data. For images, data augmentation is straight forward. We might rotate, scale, shift the hue and add noise to the images to increase the size of our dataset. With our EEG data however, there is no way we could take each data point and meaningfully rotate or scale the signal, as the features are in a fixed order.

According to [2], EEG data can be augmented by introducing small Gaussian noise over the whole training set. Like this, they could increase the size of their small dataset 30-fold, allowing them to train deeper models and getting a significantly better testing accuracy. Being centered, Gaussian noise does not change the amplitude value of EEG samples. Since we pursue a goal of improving accuracy and robustness of a classifier, different standard deviations were experimented with, as a too large noise would destabilize the network, while too little noise would have no big effect.

Having multiplied the size of the dataset, we can also train the network for fewer epochs than before, but get multiple passes over slightly different data instead, making the network more robust.

IV. Simple Ideas

A. *Feed-Forward Neural Network*

The first experimental model we tried was a Multilayer Perceptron - a Neural Network with 5 hidden layer: Input $\rightarrow$ Fully Connected (FC) $\rightarrow$ ReLU $\rightarrow$ ... $\rightarrow$ FC $\rightarrow$ Output. We trained this model over 25 epochs without data augmentation and got 65.08% of accuracy. With this non-satisfactory result we decided to apply more advanced models described in V.

V. Advanced Ideas

For some more advanced ideas, we decided to implement a Convolutional Neural Network architecture, as well as a LSTM architecture inspired by [3]. These two models aim to include a strong inductive bias that takes advantage of the fact that we are dealing with a time series. In a time series, we do not expect large changes in the strength of each sensor over consecutive time steps. With our LSTM and CNN architectures, we hope to capture this locality in an efficient way to improve our model performance over the simple fully connected model.

A. *Convolutional Neural Network*

As each data point of the EEG data is two-dimensional (time and features), another possible approach would be to leave those dimensions intact and not flatten them like with the fully connected network. Like this, we could treat each input data point like an image and run a convolutional neural network (CNN) on it. The order of the 28 sensors might be arbitrary and does not necessarily have the same properties of local information that CNNs can handle very well. We therefore chose to convolve the layers with kernel width K over all time steps, taking in the whole breadth of sensors at every step. This can be done by implementing a 1D convolution, taking the sensors to be different input channels. Our network is therefore invariant to the ordering of the sensors and does not rely on them being close together on the head implying them being close together on the input data stream.

The result of the first convolution then has C channels and is fed to at least one more convolutional layer. Taking the output from only one convolutional layer did not yield any meaningful results and we will therefore not pursue networks with fewer than two convolutional layers.

Because a single CNN with kernel size 3 might not pick up all the relevant information, we chose to run three different convolution pipelines in parallel, with kernel sizes 3, 5 and 7 respectively. The output of the three will be flattened , concatenated and fed into a fully connected layer. This layer can then choose the usefulness of the three different pipelines. The output one-hot vector will be trained using the CrossEntropy loss.

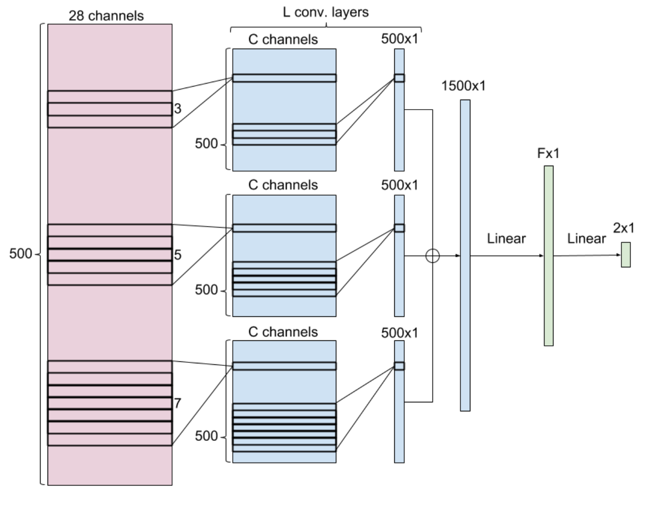


Figure 3. CNN network architecture

B. *Long short-term memory network*

Since with the EEG data we are dealing with time data, it would be a natural choice to employ a variant of a recurrent neural network (RNN). As each data point contains 500 time steps, we need a network that can remember some useful information over the whole time sequence. The LSTM and GRU modules handle such cases very well.

For this, we look at the data as being a time series of 28-dimensional feature vectors which we feed into the LSTM one by one. Using the learned weights, the LSTM will modify a hidden state as to remember information about the past and will try to predict the training target at each time step. In our classification problem, we dont need a response after every time step, so we will take only the last hidden vector after the network has seen all 500 feature vectors. This output will then be fed to a fully connected layer doing the classification using the CrossEntropy loss. We will also carry out experiments in having multiple LSTM layers stacked one after another.

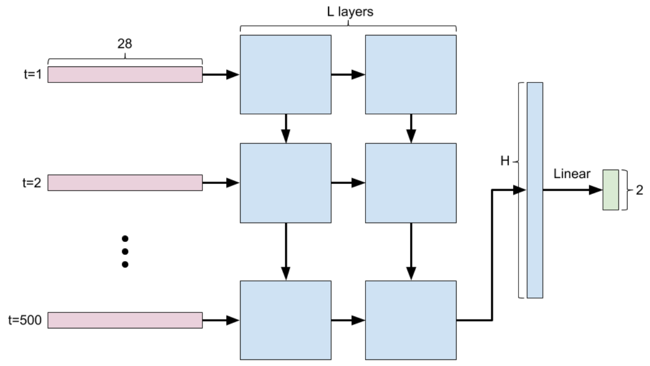


Figure 4. LSTM network architecture

VI. Results

A. Cross Validation

When trying many different neural network architectures with different hyperparameters, one should not verify the strength of each network configuration on the test set, which should be reserved for the very last test as a simulation of production data. Instead, one should create a separate validation set to validate each network on and then pick the best one. As we have a very limited amount of data (316 samples), we decided to use 5-fold cross validation and take the model that yields the best average validation loss and accuracy. After that, we train the best model one final time using the whole dataset. This model is then evaluated on the test set.

A problem that arises when using data augmentation with cross validation is, that if we augment the dataset first and then split it into train and validation sets, the validation set will be highly correlated to the train set and therefore perform very well. To remedy this problem, for each cross validation fold we first spit the 316 training samples into train and validation set and then augment only the train set.

C	M	STD	Train loss	Val loss	Train acc	Val acc
[32, 1]	1	-	0.005096	0.010414	86.01%	69.52%
[32, 1]	5	0.01	0.000785	0.010473	97.98%	74.60%
[32, 1]	5	0.1	0.000756	0.010581	98.02%	75.87%
[32, 1]	15	0.01	0.000089	0.009985	99.86%	76.83%
[32, 1]	15	0.1	0.000114	0.010638	99.83%	74.92%
[64, 1]	1	-	0.004836	0.009499	84.90%	68.89%
[64, 1]	5	0.01	0.001011	0.010644	97.39%	68.89%
[64, 1]	5	0.1	0.000874	0.010357	97.79%	72.70%
[64, 1]	15	0.01	0.000131	0.010595	99.75%	75.87%
[64, 1]	15	0.1	0.000137	0.010816	99.77%	72.06%
[32, 16, 1]	1	-	0.010305	0.010864	61.98%	59.68%
[32, 16, 1]	5	0.01	0.000767	**0.008500**	98.23%	78.41%
[32, 16, 1]	5	0.1	0.001060	0.009171	97.61%	77.78%
[32, 16, 1]	15	0.01	0.000090	0.009869	99.83%	80.95%
[32, 16, 1]	15	0.1	0.000157	0.009115	99.74%	**81.27%**
[64, 32, 1]	1	-	0.009629	0.010662	63.72%	60.95%
[64, 32, 1]	5	0.01	0.001731	0.010532	95.11%	72.38%
[64, 32, 1]	5	0.1	0.004076	0.010271	84.84%	66.67%
[64, 32, 1]	15	0.01	0.000640	0.009515	97.50%	77.14%
[64, 32, 1]	15	0.1	0.000751	0.010352	97.43%	72.06%6

Table I
C: convolutional output channels, M: data augmentation multiplication factor, STD: data augmentation standard deviation

For each fold, the validation data will then be independent of the train data. Since for each fold we calculate the augmented set anew, each fold will see slightly different data.

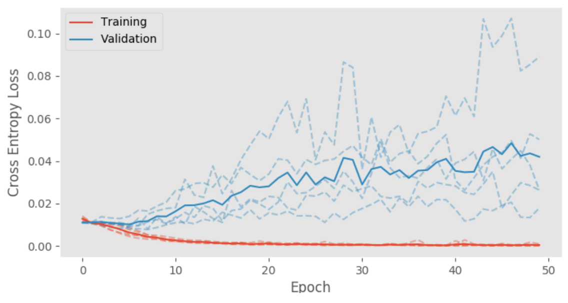
B. Experimental results from CNN model

All CNNs were trained using the Adam optimizer on a learning rate of 0.001, weight decay 0.0001 and dropout of 0.1 after each layer for 50 epochs. Table I highlights the experimental results from the CNN networks, by trying combinations of different hyperparameters and data augmentation methods. From those simulations, we get that the best model uses three convolutional layers (the first one with 32, the second one with 16 and the third one with 1 output channels), and data augmentation factor of 15 with standard deviation 0.1. This model achieves a validation loss of 0.009115 and a validation accuracy of 81.27.

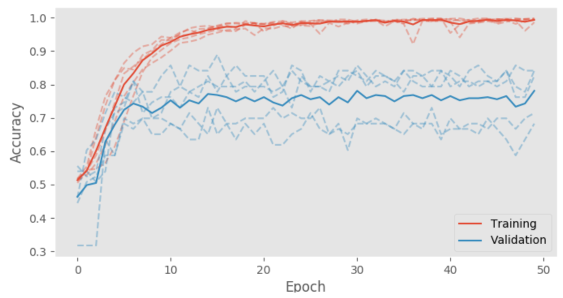
All cross validation training losses and accuracies of the best model are shown in Figure 5(a) and 5(b). We observe that the models validation accuracy saturates at around epoch 6. As the best CNN model, we train it now on the whole training data set for 6 epochs. This model achieves **76%** accuracy and a loss of **0.0156** on the unseen test set.

C. Experimental results from LSTM model

All LSTMs were trained using the Adam optimizer on a learning rate of 0.0001, weight decay 0.001 and dropout of 0.1 inside the LSTM layers for 25 epochs. Like for the CNN architectures, we experiment with different hyperparameters and data augmentation settings to find the optimum using cross validation as shown in Table II. The average best validation accuracy of 80.95% and loss of 0.007513 was



(a) Cross validation loss



(b) Cross validation accuracy

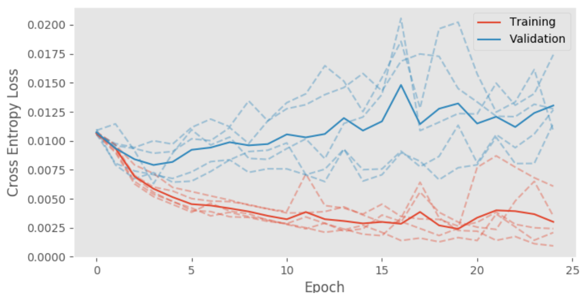
Figure 5. Training curves from best CNN model

L	H	M	STD	Train loss	Val loss	Train acc	Val acc
1	128	1	-	0.010240	0.010686	67.67%	60.63%
1	128	5	0.01	0.005472	0.008560	85.80%	77.14%
1	128	5	0.1	0.005385	0.009210	85.96%	76.19%
1	128	15	0.01	0.003211	0.008810	92.79%	75.87%
1	128	15	0.1	0.002953	0.008575	93.60%	77.14%
1	256	1	-	0.009722	0.010285	70.91%	67.62%
1	256	5	0.01	0.004281	0.008182	89.96%	77.46%
1	256	5	0.1	0.004200	0.008307	89.33%	79.37%
1	256	15	0.01	0.002470	0.008290	95.19%	78.41%
1	256	15	0.1	0.002565	0.008249	94.49%	79.05%
2	128	1	-	0.010033	0.010483	67.99%	64.44%
2	128	5	0.01	0.003947	0.008642	91.38%	77.78%
2	128	5	0.1	0.003877	0.008932	91.18%	77.14%
2	128	15	0.01	0.002119	0.008251	96.19%	80.00%
2	128	15	0.1	0.002244	0.009269	95.92%	77.14%
2	256	1	-	0.007742	0.009364	76.84%	73.33%
2	256	5	0.01	0.003648	0.008120	92.03%	80.32%
2	256	5	0.1	0.003574	0.007921	92.11%	80.32%
2	256	15	0.01	0.001769	**0.007513**	96.60%	**80.95%**
2	256	15	0.1	0.001277	0.008413	97.70%	**80.95%**

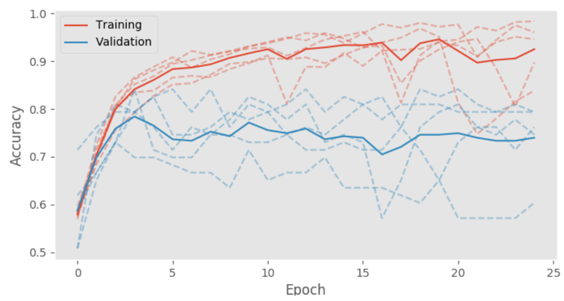
Table II
L: NUMBER OF LSTM LAYERS, H: SIZE OF LSTM HIDDEN STATE, M: DATA AUGMENTATION MULTIPLICATION FACTOR, STD: DATA AUGMENTATION STANDARD DEVIATION

achieved by a model with 2 LSTM layers, each with a hidden state size of 256, a 15-fold data augmentation factor and a standard deviation of 0.01.

All cross validation training losses and accuracies from the best performing model are shown in Figures 6(a) and 6(b).



(a) Cross validation loss



(b) Cross validation accuracy

Figure 6. Training curves from best LSTM model

We then finally trained the best model on the whole training data set for 3 epochs, since the cross validation accuracy was the best at this point. From this model we get an accuracy on the unseen test set of **75%** and a loss of **0.0116**.

VII. Conclusion

To learn the laterality of finger movement from EEG data, we trained two different architectures, namely a temporal CNN and a LSTM. The best LSTM architecture performed very similarly to the best CNN architecture, with both achieving around 75% to 80% accuracies! An advantage of the CNN model over the LSTM model is that the former takes much less time to compute all required training steps. Data augmentation for EEG data was done by applying Gaussian noise to the data set, improving the performance and robustness of the models. For future work, we would like to experiment with other types of data augmentation, like taking the real position of the sensors into account and analyzing the frequency domain.

References

[1] F. Fleuret, "EPFL EE-559 Deep Learning Course," 2018. [Online]. Available: https://fleuret.org/dlc

[2] F. Wang, S.-h. Zhong, J. Peng, J. Jiang, and Y. Liu, "Data Augmentation for EEG-Based Emotion Recognition with Deep Convolutional Neural Networks," in *MultiMedia Modeling*, K. Schoeffmann, T. H. Chalidabhongse, C. W. Ngo, S. Aramvith, N. E. O'Connor, Y.-S. Ho, M. Gabbouj, and A. Elgammal, Eds. Cham: Springer International Publishing, 2018, pp. 82–93.

[3] J. Fedjaev, "Decoding EEG Brain Signals using Recurrent Neural Networks," 2017. [Online]. Available: http://mediatum.ub.tum.de/doc/1422453/552605125571.pdf