Deep Learning models for the treatment of multichannel EEG signal

COLBOIS Laurent, HERNANDEZ Sergio Daniel, REISE Wojciech
Deep Learning Course, EPFL Lausanne, Switzerland

Abstract—**The aim of deep learning architectures is often to find function approximations, in the case of phenomena which are too complex to be described by a physical model. In this report, the task of predicting a lateral (left or right) movement from EEG data is tackled. In machine learning vocabulary, it corresponds to a binary classification problem, with a multi-channel time-series as input. We propose our models and evaluate the performance of two models that have been proposed in the literature for similar problems.**

I. INTRODUCTION - CONTEXT - CONVOLUTIONS

The dataset comes from the Brain Computer Interface (BCI) competition II, whose objective is to create good models for classifying EEG signal, i.e. determining which experimental event among several created the considered signal. We work more specifically on dataset IV, where the task is to predict with 130 ms of advance if an upcoming finger movement will be towards left or right. There are 316 training trials and 100 test trials. One data sample consists of a time sampling of the EEG signal (500 ms at a 1000 Hz sampling rate), measured simultaneously at 28 different positions of the brain [1].

The fact that the signal has some temporal and spatial continuity directly suggests the use of convolutional neural networks, as they might be able to make use of patterns at different scales. We will start by determining baselines for the prediction accuracy, using two very simple models, one linear and one convolutional on the time dimension only. We'll then develop a network consisting of a first stage of feature extraction using convolutional layers, and a second stage of classification using a linear layer. Finally, we will implement ShallowConvNet [4] and two variants of EEGNet [3], which are two reference models that show very good performance. We discuss how to adopt these two, initially presented for a different dataset.

II. BASELINES

As baselines, we have introduced a few very simple models. The aim was to get a benchmark, to compare other models against.

First, we have 2 rudimentary models composed of linear layers and non-linear activation functions. In this case, we treat the signals, which are of sizes 50×28 and 500×28, for sampling rates $100Hz$ and $1000Hz$ respectively, using layers taking as inputs mono-dimensional vectors. Hence, we transform the vectors by concatenating all the channels. It is arguable if this data representation makes any sense.

Two models $m_i : \mathbb{R}^{d\times 28} \to \mathbb{R}^2$, $i = 1, 2$, $d \in \{50, 500\}$ were tested. First, a perceptron (a single linear layer) $m_1(x) = Wx + b$, where $W \in \mathbb{R}^{2\times(d\times 50)}$ and $b \in \mathbb{R}^2$. It is equivalent to projecting the whole sample onto a 2-dimensional vector space.

Notice that, for a dataset with 316 samples and a model with 1400 parameters, the fit will be perfect and largely under-determined. So, we cannot hope for good results in the general case. We verify experimentally that indeed, the train error is 0, while the test errors are around 26% and 30% for the $100Hz$ and $1000Hz$ sampling rates respectively.

The second model is composed of 2 layers of sizes (d, hidden size) and (hidden size,2), with a ReLU activation in between. This model is even richer than what we had before - ((d+2) $\times$ hidden size) parameters- but contains a non-linearity.

III. MODELS

As mentioned in the introduction, the nature of the data, namely a signal having one spatial and one temporal dimension, drives us to use a convolution. From a signal processing perspective, we want to apply the same feature extraction process at each time step. Moreover, it allows us to significantly reduce the number of parameters, which, if we do not want to discard features, prevents us from using even very simple nets, as seen in (II).

All the models were trained using the Adam ([2]) optimization algorithm, with learning rate 0.005 and 0.001 for data-sets with sampling rates $100Hz$ and $1000Hz$ respectively. The rest of the parameters were unchanged and kept at PyTorch's default. This choice was inspired by the proceedings in [3] - one of the reference models. As for the loss function, the cross entropy was used. Here, we have to deal with a binary classification task, so the notion of proximity is not essential - being close/far from one category implies automatically being far/close to the other.

We first present two rudimentary convolutional networks we have devised and then, present 3 architectures found in the literature that we have adapted for our problem. We start by introducing them and outline what data-specific choices we have made.

A. Convolutional + Linear

This model, called $m_{convLin}$, consists of 3 layers, including an initial, single convolution layer, followed by max-pooling and a linear classifier. To be more precise, an input sample is of dimension $(28, 500)$. We start with a 1D-convolution with a kernel size $5 + 1$, $50 + 1$ (for different sampling rates), going from 28 to 1 channel. It is followed by a max-pooling layer with kernel size and stride of 3. The output of the second unit is used as input for the final, linear classification layer - mapping to a vector of size $(2, 1)$, suitable for the cross entropy loss.

One could see the initial convolution and max-pooling as learning an embedding of the signal into a lower-dimensional space. This is the second reason to choose a kernel size variable in the sampling rate. The embedded data is input to a perceptron baseline model, with the weight matrix being of size $(2, 15)$ this time. It allows us to apply a linear classifier, without the risk of over-fitting. However, we still need to train the embedding.

One could argue that we need a fully convolutional baseline. However, we argue that this model is equivalent to a fully convolutional one. Indeed, we tend to output the classification labels in the dimension of the channels, rather than time - so, a vector $(2, 1)$. This leads to using two filters, whose kernel size is the size of the time dimension. Notice though, that this is mathematically equivalent to a fully-connected linear layer.

B. ShallowConvNet

The ShallowConvNet network proposed by Schirrmeister in ([4]) is inspired by the *feature bank common spatial pattern* (FBCSP) algorithm. This algorithm aims to extract pertinent features from raw EEG signal by processing it with carefully chosen time filters, and space filters (i.e. filtering among the electrodes). The idea of ShallowConvNet is to imitate the structure of FBCSP, but using a machine learning approach to determine convenient filters through the backpropagation algorithm. Schirrmeister tests his architecture on the BCI IV 2a) dataset, which happens to have the same dimension magnitude as BCI II. After some small adaptation to match with our dataset we came up with the architecture presented in 1. Schirrmeister et al. [4] The both the batch normalization and the dropout are optionally proposed in the original model. In our case, we observed the train data was really easy to overfit (even with very simple models), thus we chose to add both batch normalization and dropout as they act as regularizers and should limit the overfitting issue.

C. EEGNet

Lawhern et al. introduced EEGNet in 2016 [3] looking for a general EEG-data neural network robust enough to not be task or data-set-specific. EEGNet is a 4 layer *compact* CNN. By *compact*, Lawhern et al. reffer to having as few as possible parameters to fit in the CNN. The number of parameters are exposed in table I. Lawhern et at. have continued since 2016 to refine their model. As Schirrmeister et al. [4] did, Lawhern et at. also took inspiration from the state of the art EEG-Analysis methods to design the architecture.

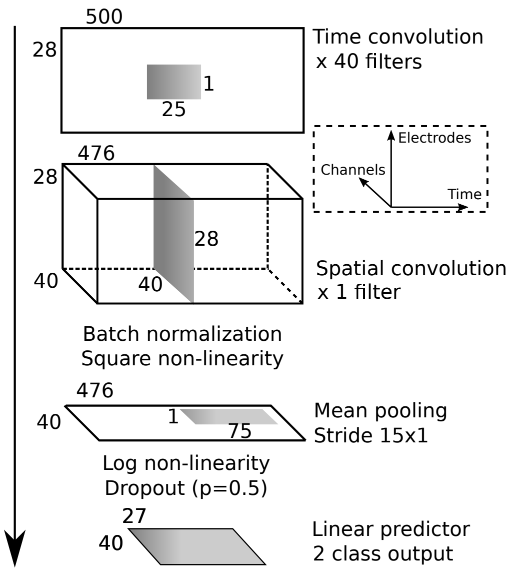

Fig. 1: *Architecture of the network, inspired by the ShallowConvNet architecture proposed by [4]. This version is the one used for the signal sampled at* ***1000 Hz****; another version with rescaled filters was also implemented.*

We implemented both the latest version of EEGNet (2018) detailed in figure 2 and the first version (2016). Nevertheless, as the latest version presented more robust and consistent results, and has nicer properties we omit describing the first version.

- First Layer

At the very beginning of the first layer, a number F of temporal filters, 8 in our case, of convolutions with a filter length thought to be half of the sampling rate are in place. In the case of the authors, the sampling rate was 128Hz, so the kernel of these F temporal convolutions was (1,64). With the whole signal being seen as 2-dimensional (channels $\times$ time), this is a 2-dimensional convolution. In our case with the sampling rate as 1000Hz, the kernel chosen is slightly smaller - (1,400). Then, as in the previous implementation, the same number of spatial convolutions are in place with kernels (C,1) with C being the number of channels (28 in our case). This second convolution is a depth-wise convolution, which means that it is performed independently over every channel of the input - similarly to a 2D convolution performed in the first layer, but with a different filter for each channel. These two step convolution is inspired by the FBSP algorithm. Batch normalization and

Layer	Layer Type	# filters	size	# params	Output dimension	Activation
1	Input				(C, T)	
	Reshape				(1, C, T)	
	Conv2D	F	$(1, 400)$	$64 * F$	(F, C, T)	Linear
	BatchNorm			$2 * F$	(F, C, T)	
	DepthwiseConv2D	F	(C, 1)	$C * F$	(F, 1, T)	Linear
	BatchNorm			$2 * F$	(F, 1, T)	
	Activation				(F, 1, T)	ELU
	SpatialDropout2D				(F, 1, T)	
2	SeparableConv2D	F	$(1, 20)$	$20 * F + F^2$	(F, 1, T)	Linear
	BatchNorm			$2 * F$	(F, 1, T)	
	Activation				(F, 1, T)	ELU
	AveragePool2D		(1, 4)		(F, 1, T // 4)	
	SpatialDropout2D				(F, 1, T // 4)	
3	SeparableConv2D	2 * F	$(1, 20)$	$2 * F * 20 + (2 * F)^2$	(2 * F, 1, T // 4)	Linear
	BatchNorm			$2 * F$	(2 * F, 1, T // 4)	
	Activation				(2 * F, 1, T // 4)	ELU
	AveragePool2D		(1, 4)		(2 * F, 1, T // 16)	
	SpatialDropout2D				(2 * F, 1, T // 16)	
4	Flatten				(2 * F * (T // 16))	
Classifier	Dense	N * (2 * F * T // 16)			N	Softmax

Fig. 2: *Architecture of EEGNet, slightly modified from the 2018 version of [3]. This version is the one used for the signal sampled at* ***1000 Hz****; another version with rescaled filters was also implemented for the signal samples at* ***100 Hz***.

2-dimensional dropout (with rate of 0.25) are in place to ensure the robustness and to drop entire filter maps instead of individual parameters.

- Second and Third layers

Separable Convolutions (in the Deep Learning Sense) are in place here. They are composed of a depth-wise convolution, of kernel (1,8) in the author's case, (1,20) in our case, followed by a (1,1) convolution, a point-wise convolution. The first convolution captures short-term information in time. The second combines information only across channels. An Average Pooling of (1,4) is then used to reduce to one fourth the effective sampling rate of the signal. Finally, Batch Normalization and 2-dimensional dropout (with dropout rate of 0.25) are in place as in the first layer.

- Final Layer - Classification

The classification is done via a soft-max function, which has a probabilistic interpretation - the first component of the output corresponds to the estimation (by the model) of the probability of being of class 1.

IV. General remarks on the training

Overall, during the study of the performance of every model, we were confronted with large variance on the results on different runs. Especially, between two successive epochs, the classification error varies quite much, thus leading to an unstable score estimate. In order to try to minimize this variance, we tried the following approaches :

- In general, solely rely on the cross-validation to assert the performance of a model, the error is computed as a mean on several folds and should show less variance. Due data scarcity, increasing the number of folds reduces drastically the size of the validation sets, giving even more unstable results. Note that we will also give the score on the test set, but this one is in general less stable and should be considered with cautiousness.
- Changing the size of the batch : it is expected that a large batch will lead to less general models, while a short batch will lead do very noisy loss evolution during the training. We sought for a intermediate batch size that mitigates both those effects.
- The learning rate was optimized by grid search, but even then the loss curves remained very noisy. Using a exponential learning rate decay scheduler (that multiply the learning rate by η slightly smaller than one after each epoch), it was possible obtain smoother curves.

V. Data augmentation

Using several of the above models, we have encountered the problem of over-fitting very soon. Recall for example, that the perceptron baseline fits the data perfectly, while it gives more than 20% of classification error on the test set. Moreover, the data is scarce and probably very noisy. There are numerous ways of dealing with overfitting and making the model more resilient to bounded random perturbations of the data.

The first two namely, dropout and batch normalization, have been described previously. The effect not being too satisfying, we have tried data augmentation. To find a good way to do it, we recall that the signal is a 28-channel recording of a time-series. We need to be aware of the fact, that we want to predict an event happening 130ms after the recording ends. Thus, the time seems to have a real meaning, so we have decided not to time-shift the signal. One could also imagine scaling each input signal. However, we have found no reason to believe that the classification result should be amplitude-invariant. Another natural approach could be to simply add small, random noise at each time step. This is in fact what we do, but in our case, the noise chosen is not independent. The usually added independent noise is a high frequency signal. In our case though, the high-frequency band-with (bounded by 50Hz and 500 Hz respectively due to the sampling rate) contains relevant information, so we opt for adding very low frequency signal.

More precisely, we sample uniformly at random a frequency $f \in [f_{low}, f_{high}]$, an amplitude $a \in [0, a_{max}]$ and a phase shift in $\phi \in [0, 2\Pi]$. This allows us to generate a time series T with the corresponding parameters. Given a sample, we can generate 28 different time series (each time, sampling f, a, ϕ uniformly, independently at random) and add them to the channels, to obtain one new sample.

For our results, we chose $f_{max} = 2$. It is motivated by the choice of the kernel size in the first convolution of ([3]). Namely, the kernel is chosen to be half the sampling rate to capture information in frequencies $2Hz$ and above. So, we can assume that we can modify the spectrum outside of that range.

VI. Comparison of the models

To properly compare these different models we performed 4-folds cross-validation on the train data. Namely, we divide the shuffled train data into 4 parts and train a newly-initialized model 4 times - each time, we leave out one of the parts for evaluation and train on the remaining 3. Before evaluating on the test data, which remains untouched until the very end, we train the models on the full train data set.

Before going further into the comparison of the models it is important to have in mind the number of trainable parameters each model carries. In table I we observe how the number of parameters scales with the sampling rate. EEG Net has indeed 1 order of magnitude less parameters than ShallowConvNet as described by Lawhern et . [3].

Model	100 Hz	1000 Hz
Linear baseline	2802	28002
Convolutional baseline	201	1731
EEG Net (2018)	1378	5362
ShallowConvNet	46722	48082

TABLE I: **Number of trainable parameters** in our different models with respect to the two versions of our data set.

A. Cross-Validation

In figures 3 and 4 we observe how our simple models performed. As expected the linear model presents a baseline (30%-40% validation error) for us. As for the convolutional baseline mode, it does out-perform the linear one and it is also clear how adding a fully convolutional classification layer does not seem to give the model any advantage with respect to the other. This is stated by Lawhern et al. [3] and seen in Schirrmeister et al. [4] ShallowConvNet implementation.

As for our more elaborated architectures, we observe they performance in figures 5, 6. In the cross-validation, our implementation of EEGNet seems to outperform ShallowConvNet. Nevertheless, ShallowConvnet performs well but has a high variance in its validation results.

B. Testing on Test Data

In table II we show the rate of missclasifications of the models on the test data after training on the train data with the same parameters specified on the cross-validation.
As expected, the linear baseline has the highest missclassification rate on the test data consistently. The different CNN performed better and achieve results similar as reported in the literature [4] [3], with EEG Net having the smaller

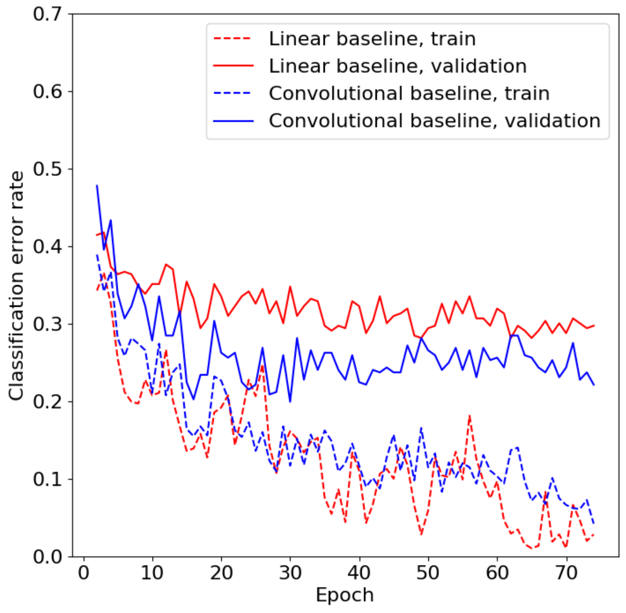

Fig. 3: *Error rate for simple models. Using the version of the data sampled at **100 Hz**. The optimizer is Adam, with an initial learning rate $\gamma = 0.2$, and a exponential learning rate scheduler that multiply γ by $\eta = 0.99$ at each epoch.*

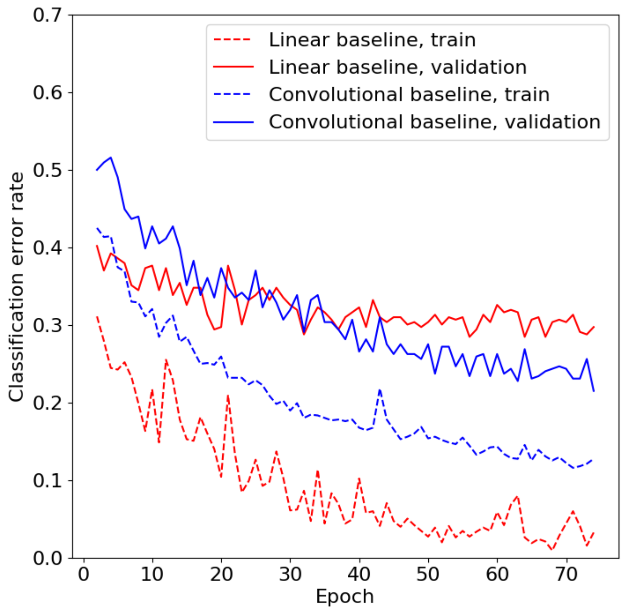

Fig. 4: *Error rate for simple models. Using the version of the data sampled at **1000 Hz**. The optimizer is Adam, with an initial learning rate $\gamma = 0.2$, and a exponential learning rate scheduler that multiply γ by $\eta = 0.99$ at each epoch.*

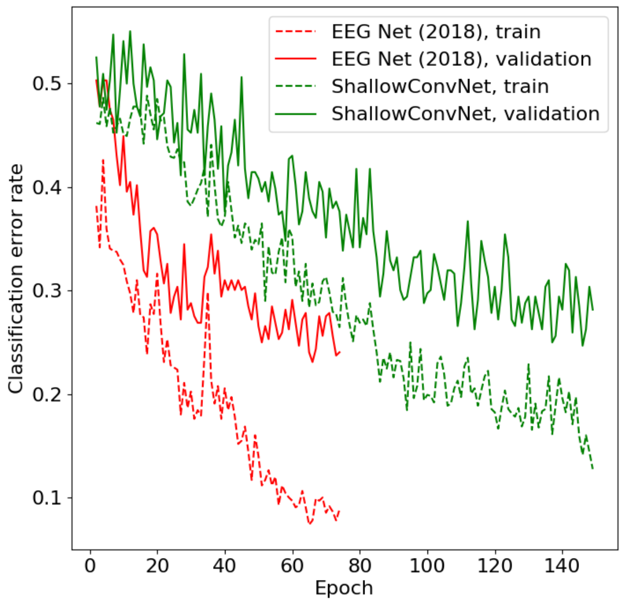

Fig. 5: *Error rate for the EEGNet and ShallowConvNet. Using the version of the data sampled at* ***100 Hz****. The optimizer is Adam, with an initial learning rate* $\gamma = 0.2$*, and a exponential learning rate scheduler that multiply* γ *by 0.995 at each epoch. The decision to stop training was motivated by the stagnation of the loss (not represented here), even though the error rate keeps decreasing.*

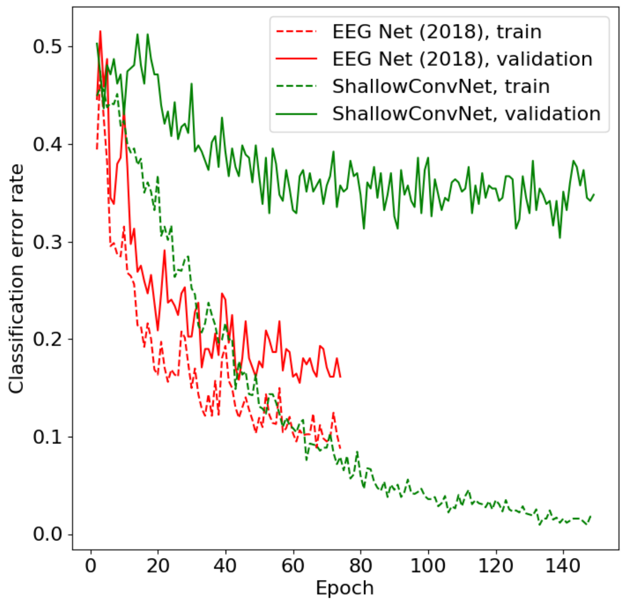

Fig. 6: *Error rate for the EEGNet and ShallowConvNet. Using the version of the data sampled at* ***1000 Hz****. The optimizer is Adam, with an initial learning rate* $\gamma = 0.2$*, and a exponential learning rate scheduler that multiply* γ *by 0.995 at each epoch. The decision to stop training was motivated by the stagnation of the loss (not represented here), even though the error rate keeps decreasing.*

Sampling rate	100 Hz		1000 Hz	
Model	Mean	St.Dev.	Mean	St.Dev.
Linear Baseline	29.5	2.9	30.2	1.6
Convolutional Baseline	27.2	5.0	29.0	4.0
ShallowConvNet	27.6	8.6	23.1	4.2
EEG Net	27.0	3.1	24.4	1.9

TABLE II: *Missclassification rate on the test data in %, averaged on 10 runs. This averaging was done in order to have a best estimate on the model actual performance, given the large variance of the results.*

missclassification rate on the 100 Hz sampled data, and with ShallowConvNet in the 1000 Hz sampled data. It is important to notice how variable are the results of ShallowConvNet and the Convolutional Baseline compared to the ones of the Linear Baseline and EEG Net.

VII. Conclusions

We implemented two state of the art architectures of CNN, explicitly design to treat EEG raw data, ShallowConvNet and EEGNet. We validated these models through a 4-fold cross validation scheme and found accuracies in the 70%-80% range, as it is found in the literature [3] [4]. EEGNet and ShallowConvNet give us very well-performing results. Nevertheless, EEGNet results are the more stable, with 10% of the number of parameters to train than ShallowConvNet. Eventhough our results demand a more in depth exploration of the differences between the two models, we can conclude that EEGNet is a better choice to treat our data.

It is discussed in the literature how Residual Neural Networks (RNN) can be an alternative to treat EEG data. Nevertheless, as Schirrmeister et al. found that RNN perform worst than CNN for EEG data [4] we decided not to investigate further in this direction.

References

[1] B. Blankertz, G. Curio, and K.-R. Müller. Classifying single trial eeg: Towards brain computer interfacing. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 157–164. MIT Press, 2002.

[2] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[3] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance. EEGNet: A Compact Convolutional Network for EEG-based Brain-Computer Interfaces. *CoRR*, abs/1611.08024, 2016.

[4] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for EEG decoding and visualization: Convolutional Neural Networks in EEG Analysis. *Human Brain Mapping*, 38(11):5391–5420, Nov. 2017.