EEG classification using Deep Learning

Alleon Antoine
Computational Science & Engineering
Email: antoine.alleon@epfl.ch

Dellafererra Giorgia
Physique
Email: giorgia.dellaferrera@epfl.ch

Zampieri Luca
Computational Science & Engineering
Email: luca.zampieri@epfl.ch

***Abstract*—This first Deep Learning project aims at training models to classify EEG signals for left/right hand movement in order to have a better understanding of the theory seen in class. Three models have been trained trained on the dataset with increasing complexity: a relatively shallow model with two fully connected layers; a deep deep employing one convolutional layer and two fully connected layers; and a deep model composed of three convolutional layers and four fully connected layers. By using data augmentation in order to increase the number of samples, used to train the model, a classification error of 16% has been reached.**

I. Introduction

Brain Computer Interfaces (BCI) is used, through measures of some brain variables, to depict the surrounding environment. Nowadays, BCI is used in engineering, medical research as well as in military. A widely employed interface for BCI is ElectroEncephaloGraphy (EEG), which is characterized by two main advantages: a good temporal resolution and a low setup cost. On the other hand, however, the signals are highly subject to noise.
New approaches and improvements have been made possible in processing signals thanks to the exploitation of Deep Learning techniques.
The use of convolutional neural networks allows to learn filters that, when applied on the signal, allow to give a better understanding of the signal itself. Furthermore, since signals from EEG are separated in channels and time series, it is easy to pre-process them and to apply convolutional layers. This report starts with a description of the dataset used. Then, the different pre-processing methods are explained and the different architectures used are described and justified. Finally, the obtained results are outlined and discussed.

II. Dataset

The dataset consists of a training set of 316 EEG samples sampled at $1[kHz]$ and one testing set consisting of 100 EEG samples sampled at the same frequency. Both sets can be imported directly sub-sampled at a frequency of $100[Hz]$ to reduce dimensionality. Each Sample consists of 28 EEG channels recording over half a second $130[ms]$ before a finger movement. The sets are labeled according to the laterality of the upcoming hand movement (0 for the left hand and 1 for the right hand). Figure 1 shows one channel of one high dimensional sample with EEG reading varying between 20 and $90[\mu V]$. In figure 2, the full sample with the 28 channels is plotted. Overall, for the first sample, the channels vary between -25 and $150[\mu V]$.

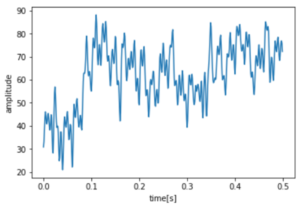

Fig. 1: High Resolution EEG channel

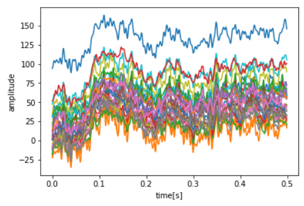

Fig. 2: Full sample with 28 high resolution EEG channels

Figures 3 and 5 report two samples taken from a left hand movement, therefore labeled 0. Figure 4 and Figure 6 show an EEG reading before a right hand movement, with a label 1. At first, those samples are not classifiable yet and are not characterized by traits that can directly differentiate them.

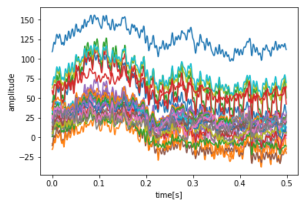

Fig. 3: sample taken before a left hand movement

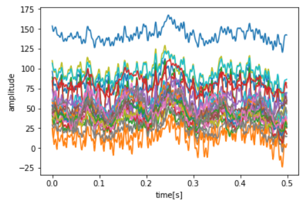

Fig. 4: sample taken before a right hand movement

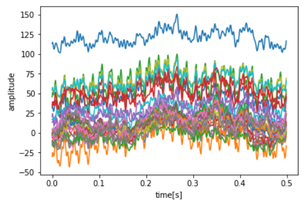

Fig. 5: sample taken before a left hand movement

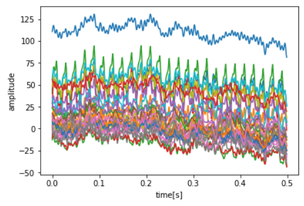

Fig. 6: sample taken before a right hand movement

Summarizing, the training set consists of a collection of 316 EEG samples with 28 channels and 500 points (sampled at $1[kHz]$) or 50 points in time (sampled at $100[Hz]$) with the respective labels. The test set is composed of a collection of 100 similar EEG samples, with their respective labels for test accuracy purposes.

III. Methods

A. Data augmentation

A training set of 316 samples of EEG channels is very small for training a model upon it and models are prone to overfitting. A solution would be to augment the data. There are several state of the art data augmentation methods that can be used to augment temporal datasets.

Window Slicing [1] is exploited with the goal of augmenting the dataset by slicing the initial signal in several overlapping or not windows. Through this method, the model learns from smaller but more numerous samples composed of the different windows.

Window warping [1] is another method to augment the dataset. Data is augmented by sub-sampling or sup-sampling certain windows of the signal. In another words, the signal is accelerated or decelerated over a window. This method allows to give more importance to certain parts of the signal to the detriment of others.

Noising can help augment the dataset for all types of data. In fact, since noise is always present in samples, adding a small amount of noise does not change the data label and helps the model to avoid over-fitting the training set.

Sub-sampling is used to augment time series data. This method is exploited particularly when importing the data with low resolution. The data imported is a sub-sampled version of the $1[kHz]$. Time series can be sub-sampled at a frequency of $f_1 = \alpha * f_0$ and the number of samples decreases to $N_1 = \alpha * N_0$. Given that sampling at f_1 overlaps points sampled at f_0, it is possible to sub-sample the original signal in $1/\alpha$ different ways, starting with a time point shift different. For example, if data is samples with a frequency 10 times smaller than f_0, a first sample will be 10x shorter with time points $x_0, x_{10}, x_{20}, ...$ but more samples can be found starting from $x_1, x_2, ..., x_9$ for a total of 10 samples from one high frequency sample.

Since the purpose of this project is not achieving make state-of-the-art results, more focus has been put on the quality of the deep neural network rather than on much data preprocessing. However, being the overfitting very significant, data has been subsampled using a sampling frequency of $100[Hz]$, which results in a decrease in the overfitting. This allowed also to multiply by 10 the number of samples fed.

B. Baseline methods

Two machine learning methods have been exploited before using Deep Learning to baseline the training of this dataset:

- *K-Nearest Neighbours* computes the L2-norm between high-dimensionality to find closest neighbors to the test set sample. The most recurrent label is found among the k closest training samples and this label has been assigned to the test sample.
- *Support Vector Machine* is a supervised classification method. SVM finds a separating hyperplane which minimizes the distance of the classified points to this hyperplane. The exact optimization problem is given by :

$$\min_{w} \left(\frac{1}{n} \sum_{i=1}^{n} max(0, y_i(\vec{w} \cdot \vec{x}_i + b)) + \lambda \|\vec{w}\|^2 \right)$$

  This method was used with a radial basis function kernel. It is widely used for binary classification problems.
- *Fully Connected Neural Networks* was used to baseline the classification. A simple two layers model of fully connected cells has been implemented by using Relu and Sigmoid activations for the first and last layer respectively; this model will be referred to as *MyNet2* in the results section. The sample is reshaped to enter the network as $28 * 500$ vector. The first layer reduces the dimension from 14000 cells to 256 and finally to 1 cell in the last layer. The loss is optimized using SGD with momentum (0.4).

C. Convolution Layers

As the task involves working with time series, it seems reasonable to add convolutional layers. Convolutional layers are used to depict patterns in the time series such as increases, decreases or bumps in a first layer, and such as more complex patterns when the number of layers and kernels is increased.

The principle of signal convolution can be seen in Fig. 7 where a convolution kernel is used to denoise or smoothen a signal. Similarly, a convolution kernel can be used to find patterns in the time serie using different kernels or filters.

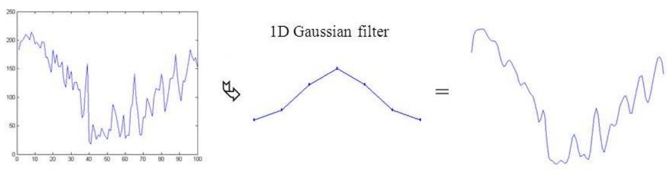

Fig. 7: Denoising using 1D convolution

Convolutions can be combined with strides to subsample the series. Using strides, the kernel is not applied to all the points in the series but skips s points when applying the kernel to the next point. The use of stride is meaningful when the data has a high resolution and some points can be omitted to reduce the dimensionality of the data, but it is also used to **reduce over-fitting**. Convolutions can also be dilated. When convolutions are combined with dilation, the points applied to the kernel are separated by d points in the time series. The effect of dilating the kernels of convolution is similar to that of a low-pass filter, if there are patterns with high frequency, they will be omitted. This method can be used to reduce noise on the sample and **reduce over-fitting**.

D. Dropout Layers

Dropout layers are layers that set some cells to 0 before computing the forward pass to the next layer. By doing this, the model is forced to train with random missing data and

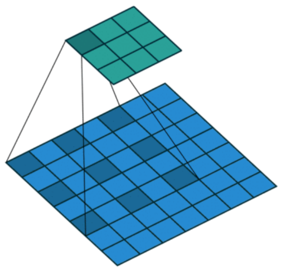

Fig. 8: Dilation

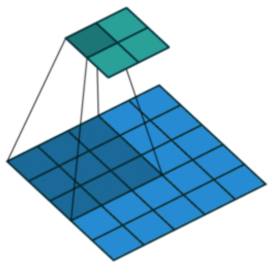

Fig. 9: Striding

over-fits less on the training set. The number of cells set to 0 is a parameter of the dropout function, the probability of setting one cell to 0 is passed as an argument to the function.

E. Binary Cross Entropy Loss

To train a binary classification model, a good choice for the loss to be minimized is the Binary Cross Entropy loss (BCE). The loss is defined in eq. 1 where y represents the probability of the sample being classified as 0. This loss emphasizes on predictions that are very far from the target, with a loss that tends to infinity if the target predicts confidently a wrong label.

$$l(y,\hat{y}) = \sum_{i=1}^{n} -(\hat{y_i}log(y_i) + (1-\hat{y_i})log(1-y_i)) \quad (1)$$

Tested on our problem, the BCELoss reached better results than the Mean Squared Error Loss.

IV. Our models

A. MyNet

To tackle the task ahead, *MyNet* has been designed [4]. *MyNet* takes as input low resolution data of size 28×50. The input is first passed into a first convolutional layer. This layer convolves the temporal signal without any stride, 8 different kernels of size (1×5) and a dilation of size (1×2). The signal is then passed onto a second convolutional layer which outputs convolutions over 16 different kernels of size (1×5) without dilations nor striding. This layer is meant to depict larger patterns in the data. The data then passes through a third convolutional layer which takes all the 28 channels together and passes them through 32 different kernels of size (28×1) to decrease the number of channel to one, that is 32 (1×42) signals. This layer aims to find the best combination of channels and patterns within them thus replacing some data preprocessing such as PCA, which could be used to reduce the number of channels only choosing high variance channels. Here, instead, the model learns what weight to give to each channel using this layer. After each convolution layers, a batch norming operation is done to normalize the data over the entire batch. This operation helps reduce over-fitting as it reduces the covariance shift after each convolutional layer. It also allows for faster training as there are no unit that have too high or too low activations [2]. Before entering the fully connected neural network, the data passes through an ELU activation function and a 2D dropout layer. The activation function allows for faster training and better backpropagation as explained in more details in the next paragraph. The dropout layer sets the activation to several of the 32 channels to 0. We have chosen to use a probability of 0.5 so half of the layers' activations are set to 0.

The data is then flattened to feed it to a fully connected neural network. The input of the fully connected neural network is a vector of size (1216) which is first reduced to (256) in the first fully connected layer. The second fully connected layer reduces further the size to (128) before passing through a second dropout layer which sets to 0 again half of the activations. Finally, the last two layers have (64) cells and (1) cell respectively. The convolution's output can be visualized in Fig. 10, in Fig. 11 and in Fig. 12. They obtained by passing through trained convolution layers and normalization layers and show the output signal for the first sample of the training set and for the first channel for Fig. 10 and Fig. 11.

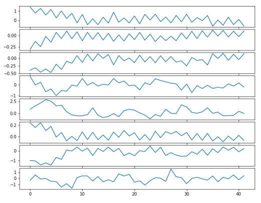

Fig. 10: output of batchnorm1a(conv1a(x)) on the first sample, first channel

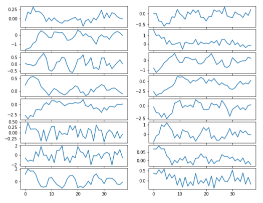

Fig. 11: output of batchnorm1b(conv1b(x)) on the first sample, first channel

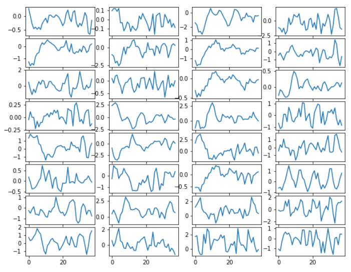

Fig. 12: output of batchnorm2(conv2(x)) on the first sample

The first three linear layers have Elu activation functions. The Elu activation function [3] is defined as : $f(x) = max(0,x) + min(0,(e^x - 1))$ and is advantageous as it reduces the vanishing gradient effect which can be observed when using saturating activation functions such as sigmoid function $(f(x) = \dfrac{e^x}{1+e^x})$ or the Tanh function $(f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}})$. Furthermore, the negative activations are kept in the form of a vanishing exponential. The final activation function is a sigmoid function, the output generated by the function is then

Layer	Type	No of parameters
conv1a	2D convolution	48
batchnorm1a	2D Batchnorm	16
conv1b	2D convolution	656
batchnorm1b	2D batchnorm	32
conv2	2D convolution	14368
batchnorm2	2D batchnorm	64
fc1	Linear	311552
fc2	Linear	32896
fc3	Linear	8256
fc4	Linear	65

TABLE I: Overview of *MyNet* layers

a value between 0 and 1 and can be seen as the approximated label.

The Binary Cross Entropy defined in III-E is then applied to this approximation. The approximation $\hat{y}$ can be seen as the label given by the Neural Network and the error is then the opposite of the log of the distance between $\hat{y}$ and the target value (either 0 for a left hand EEG or 1 for a right hand EEG).

This loss is then back-propagated in the network to update the different weights of the model using a stochastic gradient descent optimizer with momentum. The model summary can be found in table I.

B. MyNet3

MyNet3, and not *MyNet2* as it is the network used for our baseline, is mostly a simplification of *MyNet*. The number of parameters to be trained in *MyNet* is very high and reducing the number of parameters reduces the training time considerably. In fact, for 200 training epochs, the model *MyNet3* trains approximately 10 times faster than the former model.

The first layer is the same as the first layer of *MyNet* with a kernel size of (1×5) and a dilation of 2. However the number of kernels convolving the signal is reduced to 4 to reduce the number of learning parameters. The output of this layer for the first sample and first channel can be seen in Fig. 13. The second layer is a convolution along all 28 channels, similar to the layer *conv2* of the first model and similarly the output can be seen in Fig. 14. This layer however only convolves the 28 channels over 4 different kernels instead of 32 to obtain data with a shape of $(4 \times 1 \times 42)$. Once again batch-norming is applied after each convolution layer.

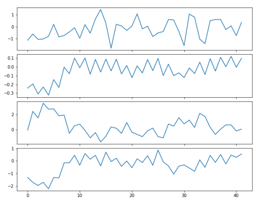
Fig. 13: output of batchnorm1(conv1(x)) on the first sample, first channel

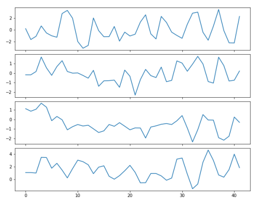
Fig. 14: output of batchnorm2(conv2(x)) on the first sample

Layer	Type	No of parameters
conv1	2D convolution	24
batchnorm1	2D Batchnorm	8
conv2	2D convolution	452
batchnorm2	2D batchnorm	8
fc1	Linear	10816
fc2	Linear	65

TABLE II: Overview of *MyNet3* layers

The output of the convolution layers is then flattened to enter the fully connected network with a size of (168). The first linear layer reduces the number of cells to (64) and the second and last layer to (1). The first linear layer is activated using the Elu activation function while the last layer is activated using a sigmoid function to output a probability like value. Finally, a dropout layer with a probability of 0.6 sets 60% of the cells activation to 0 between the first and last linear layers.
The BCE loss is then back-propagated in the network to update the weights using a stochastic gradient descent optimizer with momentum. The summary of *MyNet3* can be found in table II.

V. Results

A. Baseline

The baseline is composed of three methods, a k-Nearest Neighbors classification, a Support Vector Machine classification and a fully connected network. A grid-search was done on the first two methods giving the best test accuracy with a kNN classification with $k = 38$ and a SVM classification with $\gamma = 5 \times 10^{-5}$ and $C = 50$ for accuracies of 59% and 76% respectively training with the augmented dataset. The fully connected network is benchmarked with and without data augmentation and gives a test accuracy of 65% and 68% respectively.
It can be seen from the baseline methods that SVM scores a fair testing error compared to the k-Nearest Neighbors classification and the fully connected neural network. The baseline methods have been compared also based on the time needed to compute the test labels, that is done in one iteration for kNN and SVM, and in 200 epochs for the fully connected neural network. The time required to obtain the training accuracy is $1.2[s]$ for kNN, $12.2[s]$ for SVM and $0.61[s]$ for one epoch of training using the augmented training set, and $0.06[s]$ using the low resolution dataset.

B. MyNet and MyNet3

The two models were put to test using the BCE loss and SGD optimizer with a momentum of 0.4 and a learning rate of 0.001. A momentum of 0.4 has been used as it generally gives better results over our two models and the learning rate allows for a good convergence of the loss in different epochs number, depending on the model. We have trained our two models and computed the training loss (BCE), the validation accuracy, the testing accuracy as well as the time needed for one epoch. The models were trained using the augmented training set and

without and with a dilation of 2 and a dilation of 1 in the first convolutional layer, to see the impact of this dilation on the model accuracy.
The obtained validation accuracy was computed from 16 non augmented samples from the train set. The accuracy of the validation set is supposed to represent the accuracy of the test set. Because our test was not augmented, our validation set is not augmented either. However, due to the small number of validation samples, the accuracy we obtain is not representative of the test accuracy. The results can be seen in section VIII under table III.

C. Data Augmented MyNet3, i.e. DANet3

As we see it, the advantage of Deep Learning is that much less knowledge about the data is needed with respect to the standard machine learning. Thus it should not be necessary to do data augmentation as the model should be able to learn itself the best transformations needed. We demonstrate the with *DANet3* which is like *MyNet3* but the first layer is modified having now $dilation = 10$ and $stride = 10$ and repeated 10 times shifting the data by one and summing it together reproducing in a way the data augmentation done earlier, but allowing the net to weights each "subsampled" dataset. A dropout of 0.5 has been introduced after the two convolutional layers. Even though we didn't spend time to tune the parameters we reached a stable accuracy of 81%. This might also be explained the implied test "augmentation".

VI. Discussion

From the gathered results, it can be observed that the baseline using basic shallow learning and other machine learning methods is not as effective to train this data as a deep convolutional neural networks. Even though simple machine learning methods are much faster than training a deep neural network, they are not as efficient to classify complexed structured data, unless we are a specialist and know exactly what features should be added to augment the training set. The models we have implemented always indicate that using data augmentation yields a better result than without. Indeed, when working with small dataset, the model tends to quickly overfit the training set and training the model correctly becomes very complex. Data augmentation proves to be very efficient at avoiding this effect as was seen in both models. From the model we have implemented, it can be seen that the more complex model scores a much better test accuracy than the smaller model. Indeed, with a more complex model, the model can learn more complex manifolds and patterns. However, these models are very long to train and sometimes, the time constraint is such that these models are left out for economic reasons. In our case, we see that using a much simpler model (11377 trainable parameters against 367753 for *MyNet*) yields acceptable results in almost 10 times less time and we will therefore run the *test.py* using this model.
Finally, the effect of varying the dilation has not been conclusive as it has different effects on the bigger model than on the smaller. Doing more run with different initialization could show that there is very little to no effect of the dilation on the model.

VII. Conclusion & further improvements

To conclude, this project has proven to be challenging. Undeniably, a good understanding of the data we were given and a careful design of the layers was necessary. The process of understanding the data and implementing layers after layers the model that would best extract the features of the EEG signals was challenging and gave us the opportunity to better understand how deep learning models work and especially deep convolutional neural networks.

There are several limitations to our models and improvements that can be done.
First, the augmented data that used for training is highly correlated one another. When dealing with the augmented training set, we consider the samples as independent and train the model as such. By doing so, we train the model with a certain bias as it is used to see unshuffled signals that come from the same sup-sample. To prevent this, more data augmentation methods could have been applied such as window warping or slicing. Also, the data at hand could have been augmented by appending to it more features that the model cannot really determine, such as the Fourier transform signal of the channels, or the principal frequencies of the signal. However, this would imply a multi-modal implementation of the model. Finally, just the way the train set has been augmented, the high resolution test set could have been augmented and treated as a set of 1000 samples. After predicting the label of each of the augmented samples, the label can be inferred from the mean of the labels from the same high resolution sample. This method would have allowed us to use the full capability of the high resolution test set instead of just using the low resolution one. These problematics have been tried to be solved with *DANet3*: we showed that Deep Learning could learn most of the data augmentation itself, even though in our case we knew that this kind of augmentation should work.

VIII. Appendix

See III resuming the scores found.

References

[1] Arthur Le Guennec, Simon Malinowski, Romain Tavenard. "Data Augmentation for Time Series Classification using Convolutional Neural Networks". ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data, Sep 2016, Riva Del Garda, Italy.

[2] Doukkali F. (2017). "Batch normalization in Neural Networks". Procedia Technology. $https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c$

[3] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," CoRR, vol. abs/1511.07289, 2015. [Online]. Available: http://arxiv.org/abs/1511.07289

[4] EEGNet: A Compact Convolutional Neural Network for EEG-based Brain-Computer Interfaces Vernon J. Lawhern1,*, Amelia J. Solon1,2, Nicholas R. Waytowich1,3, Stephen M. Gordon1,2, Chou P. Hung1,4, and Brent J. Lance1

Model	Train loss	Validation accuracy	Test accuracy	Time for one training epoch
k-NN	-	-	59%	1.2s (1 epoch)
SVM	-	-	76%	12.2s (1 epoch)
MyNet2 no DA	0.0500	50%	65%	0.06s (200 epochs)
MyNet2 DA	0.00941	75%	68%	0.61s (200 epochs)
MyNet no DA, d=1	0.0581	81%	78%	1.14s (200 epochs)
MyNet no DA, d=2	0.00690	69%	80%	0.61s (200 epochs)
MyNet DA, d=2	**0.000518**	82%	**86%**	7.90s (200 epochs)
MyNet3 no DA, d=1	0.0273	81%	78%	**0.09s** (600 epochs)
MyNet3 DA, d=1	0.00759	75%	84%	0.93s (400 epochs)
MyNet3 no DA, d=2	0.00918	81%	75%	0.10s (1000+ epochs)
MyNet3 DA, d=2	0.00760	81%	78%	0.99s (400 epochs)

TABLE III: Results for the different models used. DA = Data Augmentation, d = dilation.