Prediction of Finger Movements from EEG Recordings

Tao Sun Wenlong Deng Yaxiong Luo
{tao.sun, wenlong.deng, yaxiong.luo}@epfl.ch

***Abstract*— The objective of this project is to train a predictor of finger movements from Electroencephalography (EEG) recordings, which is actually a standard two-class classification problem. This report gives a step-by-step introduction of our attempts and experiments. The final robust model has a good performance in both training and test set, with an overall 78% test accuracy. Data augmentation, batch normalization, drop out and cross validation are also involved in this project.**

I. Problem Description

The data set used in our project is the Data Set 4 of the ”BCI competition II” organized in May 2003 [1]. It is composed of 316 labeled training recordings and 100 labeled test recordings, each composed of 28 EEG channels sampled at 1kHz for 0.5s. In this project, the datasets have been downscaled to a 100Hz sampling rate and thus, the length of each down-sampled channel is 50. The goal is to predict the laterality of upcoming finger movements (left vs. right hand) 130 ms before key-press from EEG recordings.

The project can be considered as a 2-class classification problem based on 1D data of length 50 and of 28 channels. Four recordings from two data samples with label 0 and 1 respectively are presented in Fig. 1. It is clear that the difference between two samples are large, however, for each sample the trends of two channels are very similar. This kind of dependence between channels indicates conventional neural network (CNN) may be a good choice.

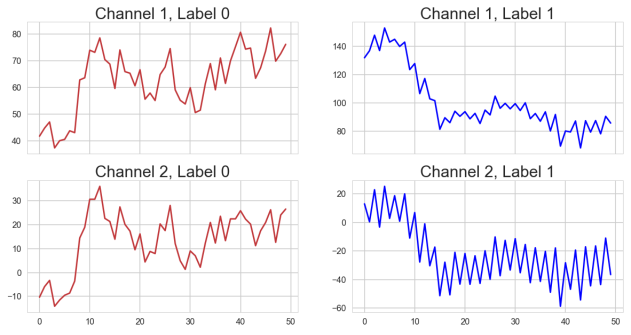

Fig. 1. Recordings from of 2 Samples (Red/Blue)

II. Pre-Experiment

A. Data Normalization

Normalization is the process of scaling individual samples to have unit norm. The mean and standard deviation of the training data are used to normalize both training and test datasets.

To be noticed, all experiments below are conducted on those normalized data.

B. Accuracy Baseline: Logistic Regression

Before conducting further experiment, we use a very simple linear predictor, the Logistic Regression model, to fit the data and thus, build the baseline of accuracy. The test accuracy is around 72%, so the aim of our design is to reach an accuracy higher than that.

III. Final Network Model

In this section, we propose the final robust network model, as shown in Table I, to give a general idea. And in the following sections, there is an step-by-step introduction of our experiments which explains how we get the final model.

TABLE I
Final Network Model

No	Structure	Parameters	Activation
1	Convolution Layer	28, 50, 5	
	Batch Normalization	50	
	Max Pooling	3	
	Drop Out	0.5	ReLU
2	Convolution Layer	50, 80, 4	
	Batch Normalization	50	
	Max Pooling	3	
	Drop Out	0.5	ELU
3	Convolution Layer	80, 160, 3	
	Batch Normalization	50	
	Drop Out	0.5	ELU
4	Convolution Layer	160, 160, 3	
	Batch Normalization	50	
	Drop Out	0.5	ELU
5	Linear Layer	800, 300	
	Batch Normalization	300	ReLU
6	Linear Layer	300, 30	
	Batch Normalization	30	ReLU
7	Linear Layer	30, 30	
	Batch Normalization	30	ELU
8	Linear Layer	30, 30	
	Batch Normalization	30	ELU
9	Linear Layer	30, 2	
	Add Result of Layer 7	Residual	Softmax

*This is report for the 1st mini-project of the course Deep Learning at EPFL 2018 given by Prof. François Fleuret.

At Table I, the final robust model consists of 4 Convolution 1D-layers (CNN) and 5 Linear layers. Parameters of CNN are input channel, output channel and kernel size respectively; parameter of Batch Normalization is the batch size; parameter of Max Pooling is the kernel size; parameter of Dropout the drop probability; parameter of Linear Layer are the sizes of input and output features.

IV. Step by Step Experiment

A. Simple Model

At first, a simple neural network is applied, which includes 2 1D-CNN Layers with max-pooling and 5 Linear Layers with ReLU. In PyTorch, SGD should be used with mini-batch. However, in our case, there are only 316 training data so we choose to use batch gradient descend instead.

The problem with this approach is

- Under-fitting happens sometime as training error rate varies a lot between epochs. It takes more than 300 epochs to well fit the training data.
- In average, we get 40% test error rate which isn't good.

B. With Optimization on Board

The reason why it takes time to reach the optimal value in training is that the batch gradient descend and constant linear rate make the convergence slower. In order to fix it, we use Adam and adjust learning rate based on the change in error rate. And with new optimization methods,

1) Adam: Adam in general has faster convergence rate and uses less memory in the calculation. Also, because there are few parameters needing tuning, it's quite straightforward and useful.

2) ReduceLROnPlateau: Although Adam has the ability to adjust the learning rate, the learning rater scheduler also helps to solve the under-fiting problem in our practice. We use `ReduceLROnPlateau` which reduces learning rate when a metric, error rate in our experiment, has stopped improving, to kind of restarting Adam and thus, to reach optimal point more quickly.

C. Trying to Avoid Over-fitting

In order to solve the over-fitting problem and make accuracy of the model more stable. Batch Normalization, Dropout and Data Augmentation are used to improve our model accuracy.

1) Batch Normalization: Simply speaking, batch normalization does data preprocessing at every layer of network. It allows each layer to learn by itself a little bit more independently of other layers. In our model, Batch Normalization is applied before the the data get into each CNN layer.

2) Dropout: Dropout is a simple but powerful method in solving over-fitting problem. In our model, Dropout is used after every layer and 50% units are dropped.

3) Data Augmentation: One major problem of project is the lack of training data which is the reason of high chance of over-fitting. In order to avoid it, we decide to do data augmentation.

We choose 200 samples from the dataset. In each sample, we resample 28×40 points out of 28×50, and do linear interpolation to make them of a length 50 again, and then give them the same label as the original sample, as shown in Fig. 2. As a result, the expanded dataset contains 516 samples.

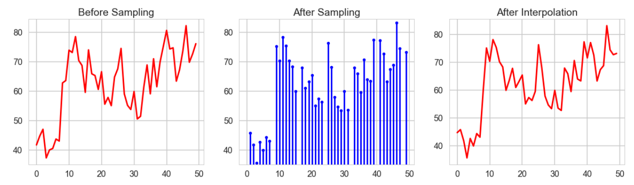

Fig. 2. Data Augmentation Example

In our opinion, this augmentation method adds some noise in the dataset and is able to make our model more robust. This method is different from simply adding Gaussian noise, which is irrelevant to our data and might destroy the original dependence between channels.

D. Other Useful Structures

1) Initialization in CNN: In the CNN module of PyTorch, the weights are not well initialized. Thus, we define a initialization function on the weights using a normal distribution, $\mathbf{W} \sim \mathbf{N}(0, 0.2)$, which is widely used in lots of open source projects at GitHub.

2) From ReLU to ELU: For ReLU, if the units are initially negative and not activated, they are always in the off-state as zero gradients flow through them, which is known as Dead Neurons. This can be solved by enforcing a small negative gradient flow through the network which is what ELU does, however, ELU is computationally more complex. To balance the trade-off, ReLU is only used in the first CNN layer since there are lots of units and network wouldn't be influenced so much if some of them are dead and in all other CNN layers, we use ELU as activation function.

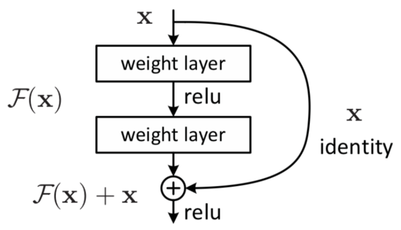
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig. 3. Residual Learning: A Building Block

3) Residual Learning: A residual building block as described in [2] is shown in Fig. 3. Although residual block is often used in deep network with dozens of layers to avoid gradient varnishing, in our experiment it somehow helps to make our result more stable.

In the following sections, we will discuss the methods about how we choose hyper-parameters and train/test our model in details.

V. Train and Test

A. Cross Validation

Cross-validation involves partitioning a sample of data into training and evaluation subsets, performing the analysis on training set, and validating the analysis on evaluation set. In our project, 10-fold Cross Validation has two main tasks, hyper-parameter selection and early-stop (performing not well in our project).

B. Hyper-Parameter

Dropout probability is set to 0.5 which can generate largest permutation of units and increase the robust ability of our model [3].

Learning rate is considered as a regularization parameter. We try 5 different initial learning rate from 10^{-2} to 10^{-3} with a step size 10^{-3}. Small learning rate is sort of "more regularizations" which helps avoid over-fitting in our model.

As for the weight-decay in Adam optimizer, it is set from 0 to 0.5 with a step size 0.05.

Table II shows the result of three scenarios, which is gotten from 10-fold cross-validation.

TABLE II
Choices of Hyper-Parameters

Scenario				**Accuracy**
No.	Learning Rate	Dropout/%	Weight Decay	
1	5e-3	0.5	0.5	0.746
2	2e-3	0.5	0.2	0.7524
3	1e-3	0.5	0	0.7515

As shown above, scenarios 2 and 3 have larger accuracy. In order to find a best combination of hyper-parameters, grid search is then applied. Finally, we choose:

- Initial Learning Rate: 2e-3
- Dropout Rate: 0.5
- Weight Decay: 0.225

C. Early Stop

We also try early stop with the help of cross-validation, however, it's not very helpful and thus, not a part of our final configuration.

D. Test Result

With the parameters combination above, we train our model once to get the so-far most robust result, as shown in Fig. 4. The test error of this single run is 22% while training error is around 18%. As the gap between training error and testing error is less than 10%, we can say that there is no obvious over-fit in our model.

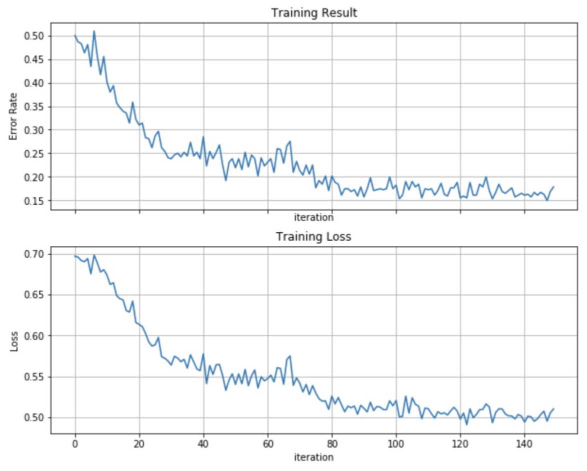

Fig. 4. Result of Single Run

We train and test our model for another 10 times to build a final evaluation metric using the average accuracy. The accuracy of 10 runs are as following:

75%	78%	74%	79%	80%
81%	78%	73%	75%	79%

On average, the accuracy is 77.20% which is good enough.

Thus, we can conclude with confidence that our model is robust and accurate to a large extent.

VI. Conclusions

We propose a robust model to predict finger movements from EEG recordings with around 78% accuracy. The proposed model consists of 4 CNN 1D-Layers, 5 Linear Layers with one residual block. We also make use of batch normalization, dropout, data augmentation to avoid over-fitting. Cross-validation is applied to search the hyper-parameters and provide evaluation result.

Due the lack of time and the unfamiliarity with practical meaning of EEG signals, we cannot make the predictor more robust and stable. Some more meaningful data augmentation methods can be applied to enrich the dataset and thus, helps build a better predictor.

References

[1] B. Blankertz, G. Curio, and K.-R. Müller, "Classifying single trial eeg: Towards brain computer interfacing," in *Advances in neural information processing systems*, 2002, pp. 157–164.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[3] P. Baldi and P. J. Sadowski, "Understanding dropout," in *Advances in neural information processing systems*, 2013, pp. 2814–2822.