Prediction of Finger Mouvement from EEG Recording

Miniproject 1 - Deep Learning Course EE-559 EPFL

Saleh Gholam Zadeh
saleh.gholamzadeh@epfl.ch
EPFL

Neeraj Yadav
neeraj.yadavl@epfl.ch
IIT Kanpur

Salmane Kechkar
salmane.kechkar@epfl.ch
EPFL

Olagoke Lukman O.
lukman.olagoke@epfl.ch
EPFL

***Abstract*—The aim of this project is to build a predictor for two basic finger movements from Electroencephalogy (EEG) records. Effectively, this task falls into the classical classification problem. Three models were built to achieve this task: Support Vector Model (SVM) , Logistic regresion and Convolutional Neural Net (Conv Net). SVM and the Logistic Regression were implemented as baselines with respect to Conv Net. The performance of the Conv Net was compared competitively with the base line. The result of our implementation are thereafter presented. In particular, we observe that conv net performs better but has high computational training cost.**

Keywords: EEG, Support Vector Machine, Convolution Neural Network, Logistic Regression.

I. Introduction

In what follows, we introduce the models that we used in order to binary classify the EEG finger mouvements.

A. EEG data description

The BCI competition EEG data was used with 2 different frequencies: 100Hz and 1kHz. A total of 316 records were used for training the models and 100 unseen records are used to evaluate the performance of the trained models. Each input record is composed of 29 channels and 50 timestamps. The data is normalized before being fed into the models. Each input signal is assigned a target label 0 or 1 to which corresponds to whether it is left or right movement.

B. Support Vector Machine (SVM)

The Support Vector Machine algorithm is a supervised learning model that can be used for both classification and regression. The later constructs a hyper-plane in an n-dimensional space that maximizes the distance of the nearest opposite-class data points. Since the SVM performs usually a linear separation between data points, it could be adapted for those that are not linearly separable, and so, by introducing kernel trick functions that project the data in other spaces. For the sake of our problem we stick on the linear configuration.

C. Logistic Regression

The input data is fed into a simple logistic regression classifier and predicts the output. The Cross Entropy Loss Function and Stochastic Gradient Descent Optimizer with a learning rate of 0.001 were used . We trained the model over 500 epochs with a batch size of 30.

D. Convolution-1d Neural Network

A conv-1d neural network was applied. In this case we have 316 training samples each of 28 channels and 50 dimensions. We applied 3 conv-1d layers and 3 Linear layers. We trained our model on 100 epochs with a batch size of 40. We used batch normalization, dropout, Relu activation function and Adam optimizer with learning rate of 0.001.

E. Convolution-2d Neural Network

We propose a dense neural network with six convolution 2d-layers and five linear layers. Since the dataset is small the convolution layers overfits the dataset hence we introduce ten dropout layers to overcome overfitting. We trained the model using Binary Cross Entropy Loss and MSELoss. For the given dataset, BCE Loss function performs better. To avoid internal covariance shift we used Batch Normalisation every time before feeding the input to a layer. Sigmoid activation function saturates while learning and hence we used the ReLU activation function. We used a batch size of 40 and epoch size of 100 to train the model. We shuffle our data after every epoch. For the optimization we tried out Adam and stochastic gradient Descent optimizer but the model performed better with the Adam optimizer. We used a learning rate of 0.00.1. We observed that if we decrease the input learning rate the model takes much time to converge and slows down the process. However, if learning rate is increased then the loss functions misses the minima. We use both max pooling and average pool layers in our model to reduce the number of parameters and time complexity.

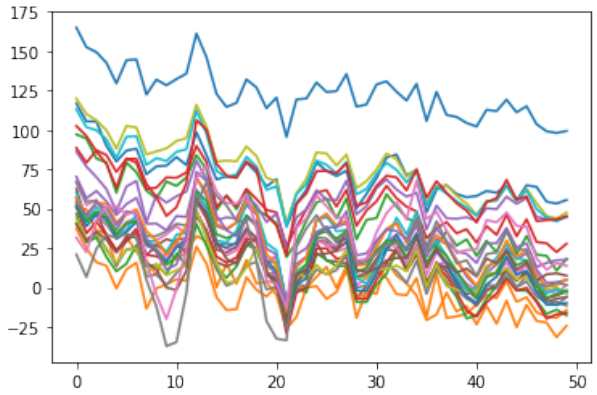

Fig. 1: EEG Signal for each channel

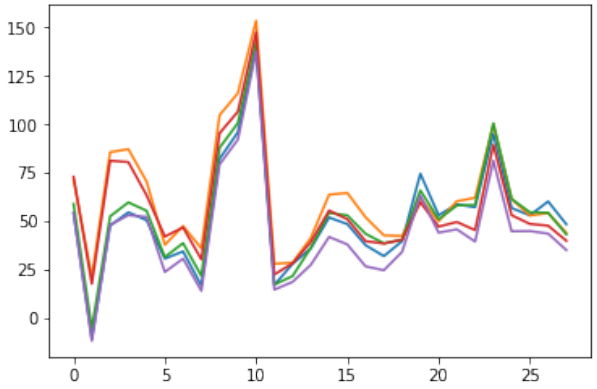

Fig. 2: data

F. Implementation

The code is in Python and for convenience a commented jupyter notebook is provided. The SVM and the logistic regression algorithms were respectively implemented using the scikit-learn library and the convolution neural networks algorithm was implemented using the PyTorch deep learning framework.

II. Experiments

A. CNN-2d architecture

The EEG dataset provided contained 316 datasamples with 28 channels and 50 dmensions but we cannot apply Conv-2d filters over the given dataset so we transformed the dataset to have 1 channel and height and width of 28,50 respectively. This way we made the data liable to apply Conv-2d filters. When we feed the EEG dataset to two convolution layers and then pass it to the fully connected layers we got an accuracy of 62% so we kept on gradually increasing layers to get better accuracies. In between, we also added Dropout layers as the data samples are very few and the chances of overfitting are very high so we carefully added dropout and then chose the Conv-2d filters. We chosen the stride of 1 to get a better grasp of data and also kept the padding size of 1 in most of layers to avoid decreasing the dimension of dataset. Also we kept the kernel size of 3 to get most out of the dataset.

We tried out max pooling after the first convolution layer but that didn't work out as the information was already less.So we decided to apply max-pooling on later stage. After experimenting with few layers we found that it is better to apply average pooling after second layer because else there was an issue of computational time . As we keep on increasing the layers there was a necessity to reduce the dimensions further hence we applied max-pooling after 5th convolutional layer as at that stage the channel size of data was 128 and before applying another conv-2d layers of 256 filters we found it efficient to half the dimensions.The table I shows the summary of our model

B. CNN-2d, CNN-1d & Logistic Regression performances

During the CNN-2d training phase we recorded the instantaneous loss of the model in the graph depicted in Fig 3. From the graph below we observe that the normalized training loss increases suddenly after 8 epochs and then decays gradually as the number of epochs increases. This can be explained by the fact that the model started learning and capturing the main signals featuring the EEG data. On the other hand, We can report that the test loss stays stable more less around the value one, since the shape of the training curve keep decreasing we expect that as long as the number of epochs increases the model would learn more patters and the test loss would subsequently decrease. Regarding the conv-1d model, at first glance, there is an important gap between the train and test losses also we can notice that the train loss as usual is decreasing whereas the test loss remains unstable and fluctuating between 3 and 4. finally, the logistic regression's loss decreases with respect to the number of epochs, we can find some spikes of loss for a couple number of epoch periods.

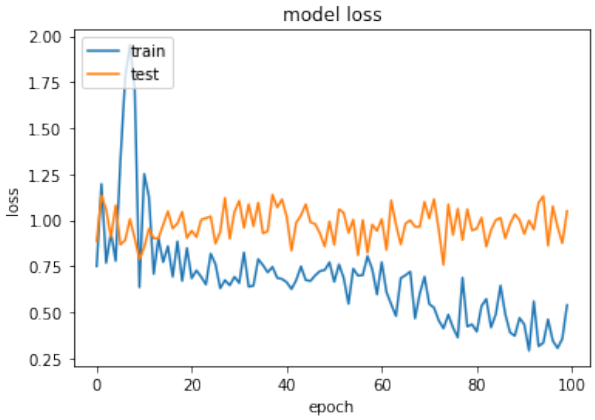

Fig. 3: Training loss over epochs for Conv2d Net

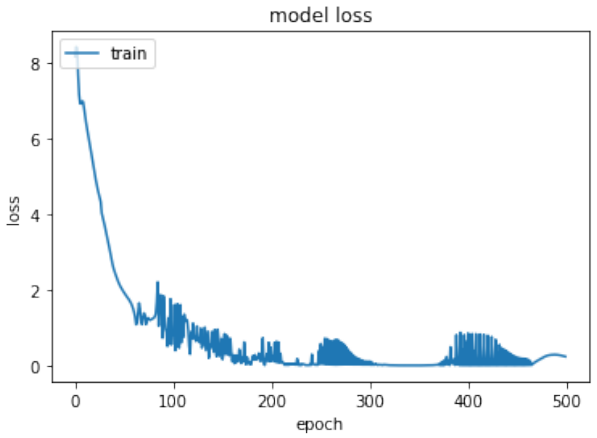

Fig. 4: Training loss over epochs for Logistic Regression

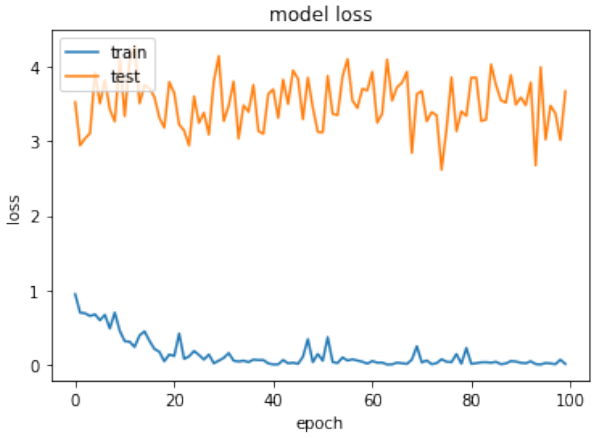

Fig. 5: Training loss over epochs for Conv1d Net

C. Basic Simulations

Both the CNNs, SVM and Logistic_Regression models were trained and tested with the aforementioned training and testing data sets and we have obtained prediction accuracies of the test datasets depicted in the Figure1 below. The training dataset was normalized before training both SVM and Logistic_Regression.

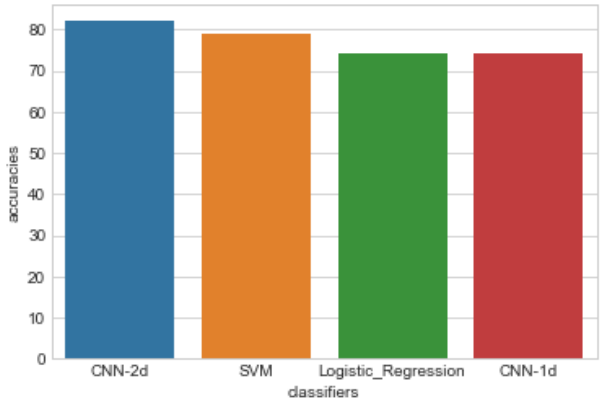

Fig. 6: prediction accuracies of the classification models

D. Discussion

From the fig 6 above, we notice that both CNN-1d and Logistic_regression scores 74% of accuracy, whereas SVM got 79% and CNN-2d scores 82%. The CNN-2d slightly outperforms the SVM classifier : in the other words, the accuracy of the Conv Net increased from 74% using Conv1d to 82% using conv2d. This is possible because the Conv2d model is able to capture more details from data. We have also to report that the normalization helps to improve the accuracy of the classifier, indeed, it allows to jump from 65% to 79%.

III. Conclusion

In the light of this Miniproject, we were able to efficiently leverage the deep learning algorithms to which we have been exposed in the course Deep course. A convolution neural network has been built and trained for classification of finger movement given EEG signal as input. Some traditional binary classifiers were built as benchmarks in order to assess the performance of the convolution neural network. Convolution neural networks in general perform better for large dataset. But, with small dataset, it gets difficult to train the large number of parameters in the convolution network . Thus simple models like SVM and Logistic Regression show comparatively advantage in real time application. Thus in such scenario it is possible to use an orchestration of simple classifier and Conv net - the conv net does the initial model building while the binary classifiers do the real time prediction and update of parameters.

Reasonable accuracies were achieved, and potential further improvements are still attainable if enough training data set is available.

IV. Further improvements

We believe that our convolution neural network model could be widely improved and many possibilities to be explored. One way to improve the accuracy is to inspect the EEG features and select only most relevant ones using techniques such as PCA, chi-squared measures ...etc. On the other hand it would be possible to ensemble the models having weak accuracies and making predictions using majority vote schemes. Finally more

interesting algorithms could be involved such as RandomForst, AdaBoost as the literature has shown those are also efficient in such problems.

TABLE I: Architecture Summary for Conv2d model

Layer	Input	Operation	Output
1	n,1,28,50	Conv2d channel=16 Kernel=(3,3) stride=(1,1) padding=0	n,16,26,48
	n,16,26,48	BatchNorm2d	n,16,26,48
	n,16,26,48	Dropout2d(0.2)	n,16,26,48
2	n,16,26,48	Conv2d channel=32 kernel=(5,5) stride=(1,1) padding=(1,1)	n,32,24,46
	n,32,24,46	BatchNorm2d	n,32,24,46
	n,32,24,46	Dropout2d(0.2)	n,32,24,46
	n,32,24,46	AveragePooling2d Kernelsize=(2,2)	n,32,12,23
3	n,32,12,23	Conv2d channel=64 Kernel=(5,5) stride=(1,1) padding=0	n,64,8,19
	n,64,8,19	BatchNorm2d	n,64,8,19
	n,64,8,19	Dropout2d(0.2)	n,64,8,19
4	n,64,8,19	Conv2d channel=128 Kernel=(3,3) stride=(1,1) padding=1	n,128,8,19
	n,128,8,19	BatchNorm2d	n,128,8,19
	n,128,8,19	Dropout2d(0.2)	n,128,8,19
5	n,128,8,19	Conv2d channel=128 kernel=(3,3) stride=(1,1) padding=(1,1)	n,128,8,19
	n,128,8,19	Maxpooling2d	n,128,4,9
	n,128,4,9	BatchNorm2d	n,128,4,9
	n,128,4,9	Dropout2d(0.2)	n,128,4,9
6	n,128,4,9	Conv2d channel=256 kernel=(5,5) stride=(1,1) padding=2	n,256,4,9
	n,256,4,9	BatchNorm2d	n,256,4,9
	n,256,4,9	Dropout2d(0.2)	n,256,4,9
7	n,256*4*9	Linear Layer	n,3024
	n,3024	BatchNorm1d	n,3024
	n,3024	Dropout(0.2)	n,3024
8	n,3024	Linear Layer	n,2128
	n,2128	BatchNorm1d	n,2128
	n,2128	Dropout(0.5)	n,2128
9	n,2128	Linear Layer	n,1024
	n,1024	Dropout(0.6)	n,1024
10	n,1024	Linear Layer	n,512
	n,512	Dropout(0.5)	n,512
11	n,512	Linear Layer	n,2