EEG Signal Classification Using Parallel Integration of CNN and RNN

Hao Lee, Yiqin Liu
University of California, Los Angeles
Los Angeles, CA, 90024
haolee0702@engineering.ucla.edu
mollyliu@ucla.edu

Abstract

This project investigates different deep learning models such as Convolutional Neural Network, Recurrent Neural Network and Fully Connected Neural Network to perform on the classification task of the motor imaginary signals from EEG data. By using parallel CNN + RNN and batch-normalization on different axis, we manage to use relatively less trainable parameters, and reach validation accuracy as high as 65.96%, and 60.5% accuracy on test data with our integrated 4 parallel CNN-GRU serials model.

1. Introduction

The goal of this project is to classify test subjects moving intention with 22 EEG electrodes' recordings. Each datum is a time-series of electrical activity in test subject's brain. These signals are notoriously noisy, some researchers have suggested to pre-process the data before applying to machine learning[1]. However, traditional frequency based filters may not fit our purpose(Sec. 3.1.2, adaptive filters are required. In this project we reference to [2] which uses CNNs as filters in time domain, and RNN to extract the information.

The model we use in this project is shown in Fig. 1, four parallel branches are used, each branch has two serial CNNs and a GRU layer (Fig. 2). The intuition of this model is using the first CNN as a time-domain filter, second CNN for spatial filter and combining the result, and GRU to extract the high level time-dependency features. The Details of the model will be discussed in later sections.

1.1. Preprocessing

The training data is augmented by duplicating one class with gaussian noise, more details are in Sec. 3.1.1.

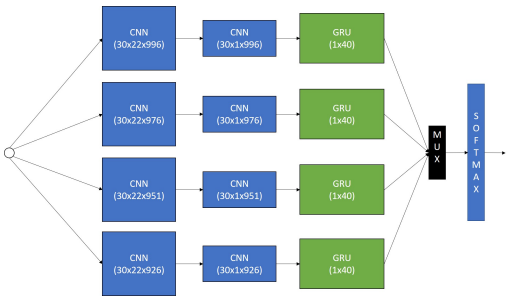

Figure 1. Structure of the model

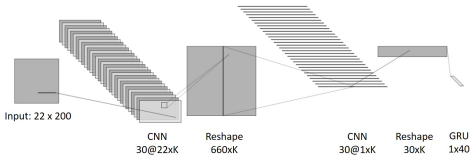

Figure 2. Operation in Single Branch (K will be different in each branch due to different filter size)

1.2. Convolutional Neural Network

Each input datum is of dimension 22×1000. The first CNN uses 30 filters with size $(1 \times n_t)$, where n_t is aligned with the time axis (for each of the four parallel serial, $n_t = 5, 25, 50, 75$ respectively), and 1 is aligned with individual EEG sensors. This filter functions as a temporal filter, which we believe can extract the temporal feature along the time axis (axis $= 3$) of one sensor signal.

Taking $n_t = 25$ as an example, the output of the first Conv2D layer in this branch will be of size $30 \times 22 \times 976$ and then reshaped into $1 \times 660 \times 976$, so that in the second layer of CNN, we can use 30 2D filters with size 660×1 to extract the spatial features contained in 22 sensors.

With these two layers of CNN, we think both temporal

and spatial features of EEG signal will be obtained and then fed into GRU in the next layer. Moreover, by using CNN we also reduce the model complexity and save running time. Batch normalization is performed after each Conv2D layer to ensure proper activation in the next layer, and average pooling is added after the second CNN before GRU.

It is worth mentioning that the axis of doing batch normalization is different for each layer. For the first layer, the batch normalization is applied on the time axis, while the later one is applied on the channel axis. Further details will be explained in Sec. 3.2.

1.3. Recurrent Neural Network

RNN is designed to utilize the sequential information with a cyclic structure. However, traditional RNN will always faces the problems of exploding and vanishing gradient. LSTM (long short-term memory) as well as GRU are designed to solve this problem.

We use orthogonal initialization and choose 1-layer GRU network with 40 hidden states since GRU will have 25% less parameters than LSTM. In order to avoid model over-fitting, L_1 regularization and dropout (0.5) are added in GRU.

The output of GRU layer will of size 40. Since there are four parallel serials of CNN and GRU, after concatenation, the output size will be of 160.

1.4. Fully Connected Neural Network

In our architecture we add two layers of FC network after concatenated GRU before softmax. The reason behind this is that after CNN and GRU, multiple temporal and spatical and dependency features have been extracted, now a dense layer is needed to fully learn those features as much as it can. To avoid over-fitting, the out put of the first FC layer is set to 40, same as the output size of concatenated GRU. The output size of the second FC layer is 4, classifying 4 subjects.

2. Results

In training we use Adam as optimizer and cross-entropy loss as cost function. The training, validation and test split in each experiment is stated in the appendix. We use Keras with Tensorflow backend as the deep learning framework to implement our models.

The result we get is validation accuracy 65.96% while the training accuracy is 91% after 200 epochs (Fig. 3). We called the training due to potential over-fitting. The output of the last dense layer during training and validation are shown in Fig. 4 and Fig. 5. Ideally, the scatter plots of the outputs should be similar, yet, by comparing Fig. 4 and Fig. 5 we can see the the *class 1* data is not classified properly as the distribution is different between training and validation. The accuracy of the model performed on individual class in test data set proves our concern (Table 1). Data augmentation is applied to address this issue, however, comes with price, which will be discussed in Sec. 3.1.1.

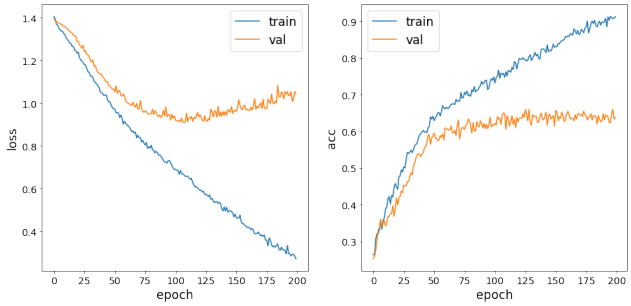

Figure 3. Loss and accuracy over epochs for training and validation

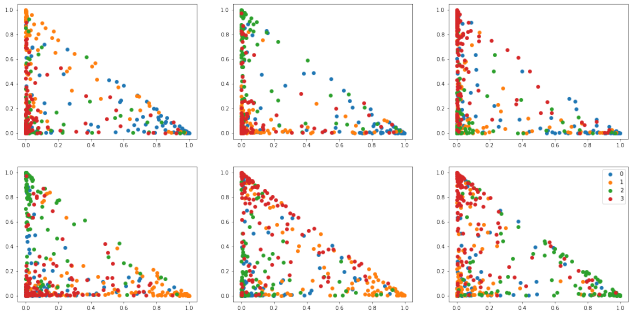

Figure 4. Output of the last layer during validation: numerical values of the 1×4 dense layer output, the color represent the class of each validation data, to have better visual representation we plots all combinations of two neurons.

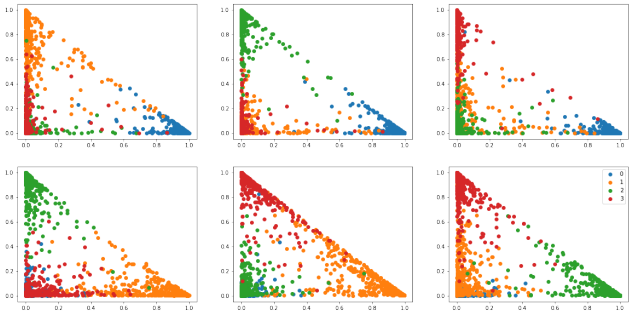

Figure 5. Output of the last layer during training: numerical values of the 1×4 dense layer output, the color represent the class of each validation data, to have better visual representation we plots all combinations of two neurons.

Test Accuracy in Different Class		
Data Class	Original	Aug. Class 1
Class 0	62.16%	56.76%
Class 1	50.39%	59.84%
Class 2	62.5%	61.46%
Class 3	68.81%	71.56 %
All	60.5%	62.3%

Table 1. Accuracy on Test Data

3. Discussion

3.1. Insights from Preprocessing

3.1.1 Data Augmentation

One of the difficulties that we found during training is classifying *class 1* during training. As we report in Table 1, the accuracy of the *class 1* data is significantly lower than the others. In order to have the model learn more about its features, *class 1* data are augmented by duplicating and adding Gaussian noise with standard deviation 1 and 2. Although this augmentation helps us to increase the accuracy of *class 1* data from 50.39% to 59.84%, the accuracy on validation data drops to 65.96% and 64.54%. Such waterbed effect makes us believe our model may have difficulties learning the class 1 data. We also consider build a dedicated model to classify *class 1* data and combine its result with the original model but result in low accuracy (A.4).

3.1.2 Signal Filtering/ Data Cropping

While some paper suggested signal filtering may help increasing the model's accuracy, in this project we did not find a digital filter that is able to pre-process without losing information. We discover although applying an low-pass filter (cutoff frequency: 0.25/0.45 sampling frequency) both expedite the learning process, yet, the model will saturate around 50%, which may suggest EEG data will have information in high frequency data.

Since while recording the original EEG data, the test subjects waited for 2 second before they decided to move, it is possible that the first two seconds of data are only background noise. Thus, data cropping is also tested, still, does not benefit the training. We assume such background noise is required for our filter-like CNN layers to have effect.

3.2. Insights from CNN

As for the architecture of the CNN, firstly use two Conv layers as we only have limited training data. Too many parameters will result in over fitting problem. We did try to use deeper CNN layers but found it hard to adjust the architecture or tune the hyper-parameters to gain a good performance.

In these two layers, data can be down-sampled to an acceptable length before it is given as input to RNNs. However, using fixed filter size in first layer of CNN means that networks will not be able to adapt to EEG patterns with different time duration. In order to mitigate these problems, we use 4 branches 2D convolution filters with step size $n_t = 5, 25, 50, 75$, enabling the network to learn laocal EGG patterns with different time length appropriately and reduce the input signal automatically.

Batch normalization and pooling are applied on the CNNs, yet, in a different way. We believe for each different individuals, their EEG signals may have different biases and variances. Thus, the first batch normalization is applied on the time axis, e.g., for input data $(1 \times 22 \times 1000)$, we apply batch normalization on axis with dimension 1000. Few may judge this breaks the correlation between different time points, yet, we assume the EEG data is noisy enough to prevent it. Our results shows less overfitting, when reaching 66% accuracy in validation, batch normalized along channels has training accuracy around 95%, compare to our method, it is around 80%.

3.3. Insights from RNN

There is an interesting phenomenon that with only RNN (LSTM or GRU), the performance is poor, no more than 30% in validation accuracy, which is only a bit better than random guess. While RNN is said to have huge advantage in dealing with sequential data, why the performance is overshadowed by CNN in this project?

We think the reason is that the goal of the network is to do classification, rather than text generating or predicting next pulse in EEG signal. With 2 layers of CNN learning features from temporal then spatial perspective, the CNN have good enough features to do classification as well as avoiding over-fitting. For RNN, by merely learning the dependency relation of EEG signal, it is not enough for the network to distinguish among left, right, foot, tongue movement.

References

[1] F. Lotte, L. Bougrain, A. Cichocki, M. Clerc, M. Congedo, A. Rakotomamonjy, and F. Yger. A review of classification algorithms for EEG-based brain-computer interfaces: A 10 year update. *Journal of Neural Engineering*, 15(3), 2018. 1

[2] 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. *Human Brain Mapping*, 38(11):5391–5420, 2017. 1

A. Hyper-parameters

A.1. Training on 1 subject v.s. Training on All

The result we get from training with only one subjects data over-fit easily. The training accuracy get to more than 97% in less than 10 epochs, while validation accuracy is struggling around 45%. We believe with merely around 200 data for training and validation is not enough to train our model. Data augmentation is applied by duplicating the whole training set plus white noise. The result is shown in Fig. 6. The model clearly over-fits, however, the validation accuracy reaches as high as 66.66%. It shows our model is able to detect EEG data as long as the number of training data is enough.

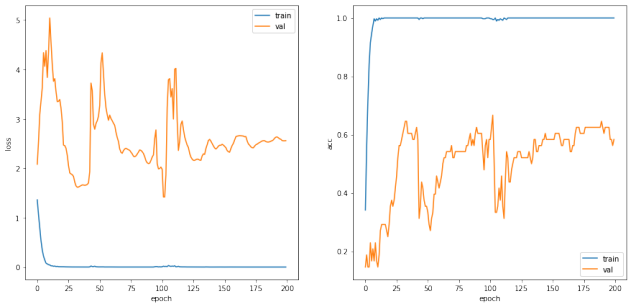

Figure 6. Loss and accuracy over epochs for training and validation with only 1 test subject's data

A.2. Data Augmentation

A.3. Batch normalization axis

As we mentioned in 3.2, the axis we choose to do batch normalization is different than usual cases. For instance, the output in one of branches with dimension $(channels \times EEG \times time)$, we choose to do normalization along time but not channels. Numerically, we found the performance of our model has similar accuracy (66.1%), yet it takes longer time to train (500 epochs) compare to the other (300 epochs), and with slightly higher training accuracy (93%). As a result, we choose to normalize along the time. Although by the due time of this project we are still unsure what causes this behavior, nevertheless, our current assumption for EEG signals, the bias and variance of the EEG signals may varies, and accumulated as time goes on, thus, each individual time may have different mean values and variance.

A.4. Separated-train Classifier

One of the problem we faced during training the model is classifying *class 1* data, as we see in Table 1, trying to amend this issue came with a separated model. Our assumption is duplicating the same model we have, but with *Class 1* data labeled as 1, and the rest data labeled as 0. Ideally, it should be able to return an (1×4) vector $(0, 1, 0, 0)$ for *class 1* data and $(0, 0, 0, 0)$ for the rest. By adding it back to the original with multiplying a weight (λ), we should get a model that have better classification on *class 1*. However, while the separated model has 78% accuracy with its separated validation data, the results show a much worse test accuracy (45%, $\lambda = 1$) in the original data, and gets optimal accuracy if $\lambda = 0$. We conclude that making our model output 0 on different classes may not be possible, unless we developed a special structure that focuses on the difference between *class 1* and *other classes*, such idea may not be realizable.

B. Architecture

	Layer	Detail	Input	Output
branch 1	**1**	30@ Conv2D ($1 \times$ **25**)	$1 \times 22 \times 1000$	$30 \times 22 \times 976$
	2	Batchnorm (axis = 3)	$30 \times 22 \times 976$	$30 \times 22 \times 976$
	3	30@ Conv2D (660×1)	$1 \times 660 \times 976$	$30 \times 1 \times 976$
	4	Batchnorm (axis = 1)	$30 \times 1 \times 976$	$30 \times 1 \times 976$
	5	AveragePool(1×76, stride $(1, 15)$)	$30 \times 1 \times 976$	$30 \times 1 \times 61$
	6	GRU (hidden units = 40)	30×61	40
branch 2	**7**	30@ Conv2D ($1 \times$ **50**)	$1 \times 22 \times 1000$	$30 \times 22 \times 951$
	8	Batchnorm (axis = 3)	$30 \times 22 \times 951$	$30 \times 22 \times 951$
	9	30@ Conv2D (660×1)	$1 \times 660 \times 951$	$30 \times 1 \times 951$
	10	Batchnorm (axis = 1)	$30 \times 1 \times 951$	$30 \times 1 \times 951$
	11	AveragePool(1×51, stride $(1, 15)$)	$30 \times 1 \times 951$	$30 \times 1 \times 61$
	12	GRU (hidden units = 40)	30×61	40
branch 3	**13**	30@ Conv2D ($1 \times$ **75**)	$1 \times 22 \times 1000$	$30 \times 22 \times 926$
	14	Batchnorm (axis = 3)	$30 \times 22 \times 926$	$30 \times 22 \times 926$
	15	30@ Conv2D (660×1)	$1 \times 660 \times 926$	$30 \times 1 \times 926$
	16	Batchnorm (axis = 1)	$30 \times 1 \times 926$	$30 \times 1 \times 926$
	17	AveragePool(1×26, stride $(1, 15)$)	$30 \times 1 \times 926$	$30 \times 1 \times 61$
	18	GRU (hidden units = 40)	30×61	40
branch 4	**19**	30@ Conv2D ($1 \times$ **5**)	$1 \times 22 \times 1000$	$30 \times 22 \times 996$
	20	Batchnorm (axis = 3)	$30 \times 22 \times 996$	$30 \times 22 \times 996$
	21	30@ Conv2D (660×1)	$1 \times 660 \times 996$	$30 \times 1 \times 996$
	22	Batchnorm (axis = 1)	$30 \times 1 \times 996$	$30 \times 1 \times 996$
	23	AveragePool(1×96, stride $(1, 15)$)	$30 \times 1 \times 996$	$30 \times 1 \times 61$
	24	GRU (hidden units = 40)	30×61	40
Cat 1 – 4	**25**	Concatenate layer 6, 12, 18, 24	4@40	160
	26	Fully connected network	160	4

Table 2. Network Structure

- All CNN layers are activated with ReLU.
- Use Adam as optimizer with $lr = 3e - 5$.
- Data augmentation is discussed under section 3.1.1.