Deep Learning for Classifying EEG Recordings with CNN and RNN

Dong Wang
UCLA
Wangdong90@ucla.edu

Juan Rebanal
UCLA
jrebanal@g.ucla.edu

Haitao Chen
UCLA
harrychen@ucla.edu

Abstract

This paper presents the design and best practices of classifying electroencephalography (EEG) recordings using deep learning techniques. The task is to classify various body movements that the subject is thinking about without actually moving (motor imagery), using only the recordings of the electrical activities outside the scalp. Deep learning techniques have been applied in this filed through end-to-end learning, i.e. learning from the raw data. And the performance can be improved a lot after incorporating some cutting-edge tools.

1. Introduction

The data set consists of EEG data from 9 who are completing four different tasks, including the imagination of movement of the left hand, right hand, both feet and tongue.

Firstly, standalone convolutional neural network models are proposed and two architectures which have the best performance are chosen. Both of them yield a satisfying accuracy, over 70%. One of them is the traditional CNN model, named as CNN++[1]., extracting the hidden features with convolution layers on the original "image", composed by the different electrodes as the height and the time stepping of each recording as the width. The other architecture called Deep ConvNet [2] can handle the spatial and temporal characteristic of EEG input, with a special designed block.

Of course, recurrent neural network should be a good candidate due to the temporal nature of these recordings, where at each time step the network retains information from previous time steps. Here we try both LSTM and GRU, which can solve the short-term memory problem caused by vanishing gradient of vanilla RNN. The result of them are very close, and GRU will run faster due to the replacement of cell state compared with LSTM.

Then a combination of CNN and RNN across multiple input modalities has been exploited to improve the results. The classification accuracy is not as good as the CNN++ and Deep Convnet, but it can achieve better result than RNN models with very few epochs of training.

2. Approach

The data contains recordings from 9 human subjects, each of them has the same time length of 4 seconds with 250hz sampling rate. With about 530 samples per, the full size of the training data is (2115 samples ×25 electrodes × 1000 time points). The last three electrodes are monopolar EOG channels and are contained in the CNN models and removed in the RNN models. In addition, the first second of the recording which is the first 250 time points is also removed in the CNN models. And we down sample the recordings to reduce sample rate from 250Hz to 125 Hz in CNN++ and 25 Hz in Deep ConvNet. We also try different down sample rate in the RNN models. We don't apply data normalization since no improvement is observed. After pre-processing, the final shape of the training data before it gets to the classifier is: CNN++ & CNN+GRU (4230×25×375), Deep ConvNet (21150×22×100), LSTM (4230×22×375), and GRU (4230×22×375). Each model will be briefly introduced, and the detailed description will be included in the appendix part.

2.1. CNN++

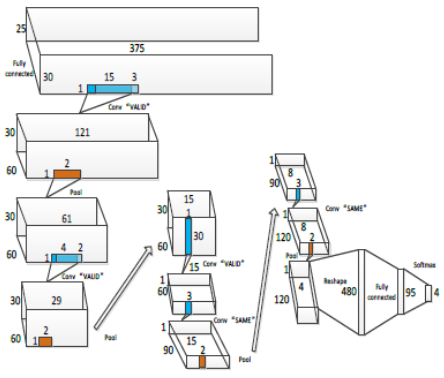

Figure 1: CNN++ Architecture

As shown in Figure1, CNN++ has an initial projection that use a fully connected layer that change the input channels from 25 to 30. Therefore, the projected input of each trial has size 30×375, which is then fed to the

convolutional layers. Finally a softmax layer is added. All convolutional layers have dropout with probability 0.5 and also batch normalization. All activation function is Exponential Linear Units (ELU). In CNN architecture, I follow this sequence: Conv → Batch Normalization → Activation→ MaxPooling → Dropout

2.2. Deep ConvNet

The deep ConvNet has four convolutional-max-pooling blocks. The first block includes two convolution layers which deal with the time domain and spatial domain separately. Since there is no activation function in between the two convolutions, they could in principle be combined into one layer, which is the same as CNN++. Using two layers however implicitly regularizes the overall convolution by forcing a separation of the linear transformation into a combination of two convolutions. Here we also use batch normalization and ELU as activation in all the blocks.

2.3. LSTM

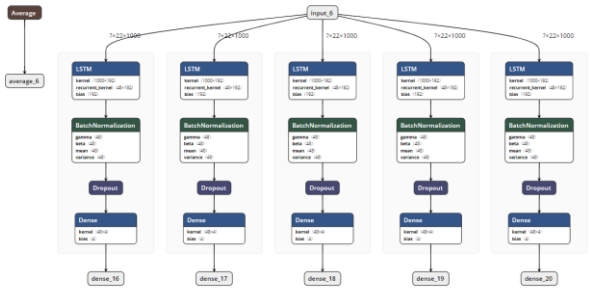

Figure 2: LSTM Architecture

Figure 2 shows the architecture used in creating the model. We use a LSTM layer with 48 units, which was chosen among numbers of units ranging from 8 to 512 in steps of powers of 2 at first before adjusting more finely. The LSTM layer also makes use of dropout of p = 0.5 at the affine calculation of the next cell state as well as at the input to each of the gates to improve generalization of the model. Dropout in the context of being within a LSTM layer refers to dropping a certain proportion of the LSTM cells. Regularization was used with a regularization factor of 0.001. The non-recurrent and recurrent weights were initialized to be random-seeded orthogonal matrices as this showed empirically better results in terms of higher test accuracy and less epochs needed to converge than Xavier weight initialization.

Following the LSTM, we use a batch normalization layer to normalize the statistics of the LSTM's output, improving performance. As in course homework, we initialize the gamma values as a vector of ones and the beta values as a vector of zeros. After the batch normalization layer, a regular dropout layer is used with p = 0.5 to approximate bagging and make the model more robust to variations in data, again improving generalization.

Finally, a fully connected layer with 4 outputs (the number of classes to classify) is used with a softmax layer afterwards.The Adam optimizer was used to speed up the convergence of the network's loss function, and after much experimentation we arrived at a learning rate of 0.000.

2.4. GRU

Several structures with GRU are tried and the one that performs best is as follows: We first permute the input data into (22, 375) and then use a fully connected layer (Dense layer), which contains 64 units. This Dense layer is meant to reduce the time features from 375 to 64. Regularization using the L2 norm was used for the kernel weights with a regularization factor of 0.05 to improve generalization. A dropout layer with p = 0.5 was added to further improve generalization. Then, the data is permuted back to the size of (64, 22). After that, there is a simple one-dimensional convolutional layer with 32 kernels(filters). After batch normalization and activation, a regular dropout layer is used with p = 0.5 to approximate bagging and make the model more robust to variations in data, again improving generalization.

After the above layers, the data was fed into two GRU layers, each with 32 units. For each GRU layer, regularization using the L2 norm was used for the kernel and recurrent weights with a regularization factor of 0.01 to improve generalization. Also, a batch normalization layer was used to normalize the statistics of each GRU's output. A regular dropout layer is used with p = 0.5 after each batch normalization layer.

Finally, a fully connected layer with 4 outputs (the number of classes to classify) is used with a softmax layer afterwards to convert the findings into a probability distribution.

2.5. CNN+GRU

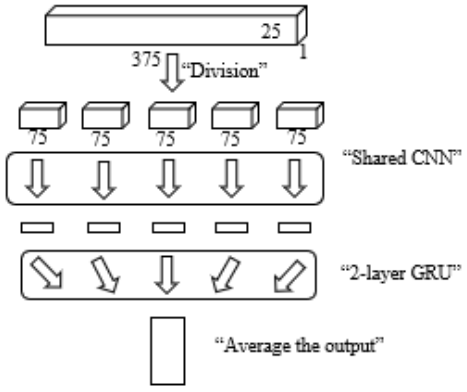

Figure 3: CNN+GRU Architecture

The combination of CNN and RNN is exploited by a hierarchical neural network. Firstly, the input trial consisting of 375 time steps is divided into 5 frames, each containing 75 samples. We apply the shared CNN model which has 5 layers of convolution and pooling. The output of the 5 frames are fed to a 2-layer GRUs with hidden dimension 80. And then the average of the hidden states from 5 frames are followed by a softmax layer.

3. Experiment

We use the different training configuration for each approach to try getting the best result. We use softmax cross entropy as loss function and L2 loss is applied for regularization. By default, we are using Adam gradient optimizer to update weights. But SGD with Momentum is a better choice in some case. The testing results are shown in Table 1.

Table 1
Classification Accuracy on Test Data

Model	Accuracy
CNN++	75%
Deep ConvNet	70%
LSTM	46%
GRU	41%
CNN+GRU	55%

From Table 1 we can see that the overall performance of each model. CNN++ achieves the highest accuracy. And LSTM and GRU have almost the same result. And CNN plus RNN yield less accuracy than pure CNN models

Based on the performances of different models of GRU, a simple one-dimensional convolutional layer can help improve the performance a little, which implies that the CNN explained before may be a better choice for this dataset. Also, models with simple structures tend to outperform those with complex structures.

Table 2
Testing Accuracy for Each Subject

Subject	CNN++	ConvNet	LSTM	GRU	CNN+GRU
A01T	68%	62%	38%	44%	39%
A02T	66%	52%	46%	34%	57%
A03T	78%	86%	26%	46%	48%
A04T	64%	72%	26%	50%	53%
A05T	81%	79%	36%	45%	61%
A06T	63%	61%	49%	41%	42%
A07T	86%	72%	50%	40%	78%
A08T	82%	72%	26%	36%	69%
A09T	79%	72%	43%	32%	41%

Table 3
Testing Accuracy for Each Task

Task	**CNN++**	**ConVNet**	**LSTM**	**GRU**	**CNN+GRU**
Left Hand	72%	74%	32%	24%	40%
Right Hand	77%	65%	17%	33%	80%
Feet	76%	67%	60%	55%	35%
Tongue	71%	74%	45%	54%	65%

The table 2 shows that the accuracy of different subjects vary a lot. This may be due to the complexity of human brain, EEG can't fully handle the characteristic of those activities. And table 3 shows that different model maybe good at different task. We can use the multi-model ensemble method to deal with the similar tasks in the future.

4. Conclusion

Through various deep neural network architectures, the paper has shown that such networks can have various degrees of success when classifying EEG activity. While LSTM and GRU networks have shown some potential to be good tools for classifying EEG activity, having a limited amount of training data coupled with attempts at data augmentation not improving performance holds them back in this project. The CNN-based networks such as CNN++ and Deep ConvNet perform the best out of the architectures investigated, with CNN + GRU following behind them.

The observation that RNN can't get good result may be due to the short duration of our dataset. The trials are only three or four seconds long, so there is really not much "long term" temporal relations to learn. But convolutional networks, on the other hand, can learn the local time and frequency features while at the same time learn more abstract features in higher layers which can also span more time steps. We believe that for some complicated tasks in reality, which will last for a long time, RNN performance will definitely be improved a lot.

By the way, across all architectures, we found that proper preprocessing of the data and exploiting knowledge about the dataset were helpful in improving performance. This demonstrates the importance of understanding how the data was acquired and more generally how features other than those defined in the dataset can prove helpful. Also, balance between complexity of structure and proper regularization is very important.

References

[1] Y. Zhao *et al*., "On the improvement of classifying EEG recordings using neural networks," *2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA, 2017, pp. 1709-1711.

[2] Schirrmeister, R.T., Springenberg, J.T., Fiederer, L.D., Glasstetter, M., Eggensperger, K., Tangermann, M.W., Hutter, F., Burgard, W., & Ball, T. (2017). Deep learning with convolutional neural networks for EEG decoding and visualization. *Human brain mapping*.

[3] Roy, S. , Kiralkornek, I. , & Harrer, S. . (2018). Chrononet: a deep recurrent neural network for abnormal EEG identification. *International Joint Conferences on Artificial Intelligence 2018.*

[4] Cho, Kyunghyun; van Merrienboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation"

Appendix

1. CNN++

Layer	In_channels	Out_channels	Kernel	Padding
Projection: (25,375,1) →(30,375,1)				
Conv2D	1	60	(1,15)	valid
MaxPooling2D	60	60	(1,2)	same
Conv2D	60	60	(1,4)	valid
MaxPooling2D	60	60	(1,2)	valid
Conv2D	60	60	(30,1)	valid
Conv2D	60	90	(1,3)	same
MaxPooling2D	90	90	(1,2)	same
Conv2D	90	120	(1,3)	same
MaxPooling2D	120	120	(1,2)	same
Dense :480→ 95				
Dense :95→ 4				

2. Deep ConvNet

Layer	In_channels	Out_channels	Kernel	Padding
Convolution in Time Domain (Conv2D)	1	25	(10,1)	valid
Convolution in Spatial Domain (Conv2D)	25	25	(1,22)	valid
MaxPooling2D	25	25	(3,1)	valid
Reshape: (30,1,25) → (30,25)				
Conv1D	25	50	3	same
MaxPooling1D	50	50	3	valid
Conv1D	50	100	3	same
MaxPooling1D	100	100	3	valid
Conv1D	100	200	3	same
MaxPooling1D	200	200	3	valid
Dense :400→ 4				

3. LSTM

Layer	Hyperparameters
LSTM	Units: 48, p for recurrent and non-recurrent dropout: 0.5, batch size: 16, epochs: 200, optimizer: Adam, learning rate: 0.0001, L2 reg. with factor 0.001, orthogonal weight initialization
Batch Norm	L2 reg. with factor 0.001, gamma initialized w/ones, beta initialized w/zeros
Dropout	P = 0.5
Dense	Number of outputs: 4, classifier: softmax, Xavier weight initialization, L2 reg. with factor 0.001

4. GRU

Layer (type)	Output Shape	Param #
permute_5 (Permute)	(None, 22, 375)	0
dense_5 (Dense)	(None, 22, 64)	24064
dropout_9 (Dropout)	(None, 22, 64)	0
permute_6 (Permute)	(None, 64, 22)	0
conv1d_3 (Conv1D)	(None, 31, 32)	2144
batch_normalization_7 (Batch	(None, 31, 32)	128
activation_3 (Activation)	(None, 31, 32)	0
dropout_10 (Dropout)	(None, 31, 32)	0
gru_5 (GRU)	(None, 31, 32)	6240
batch_normalization_8 (Batch	(None, 31, 32)	128
dropout_11 (Dropout)	(None, 31, 32)	0
gru_6 (GRU)	(None, 32)	6240
batch_normalization_9 (Batch	(None, 32)	128
dropout_12 (Dropout)	(None, 32)	0
dense_6 (Dense)	(None, 4)	132

Total params: 39,204
Trainable params: 39,012
Non-trainable params: 192

GRU Architecture 1

Layer (type)	Output Shape	Param #
permute_1 (Permute)	(None, 22, 375)	0
dense_1 (Dense)	(None, 22, 64)	24064
dropout_1 (Dropout)	(None, 22, 64)	0
permute_2 (Permute)	(None, 64, 22)	0
gru_1 (GRU)	(None, 64, 32)	5280
batch_normalization_1 (Batch	(None, 64, 32)	128
dropout_2 (Dropout)	(None, 64, 32)	0
gru_2 (GRU)	(None, 32)	6240
batch_normalization_2 (Batch	(None, 32)	128
dropout_3 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 4)	132

Total params: 35,972
Trainable params: 35,844
Non-trainable params: 128

GRU Architecture 2

Layer (type)	Output Shape	Param #
permute_5 (Permute)	(None, 22, 375)	0
dense_5 (Dense)	(None, 22, 64)	24064
dropout_5 (Dropout)	(None, 22, 64)	0
permute_6 (Permute)	(None, 64, 22)	0
gru_3 (GRU)	(None, 32)	5280
batch_normalization_3 (Batch	(None, 32)	128
dropout_6 (Dropout)	(None, 32)	0
dense_6 (Dense)	(None, 4)	132

Total params: 29,604
Trainable params: 29,540
Non-trainable params: 64

GRU Architecture 3

GRU Classification Accuracy on Test Data

Architecture	Accuracy
GRU 1	41%
GRU 2	34%
GRU 3	38%