Writeup for ECE239 Project

Yifan Shu, Ruoyu Peng, Haiqi Xiao, Yifeng Zhang
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095
evanshu@g.ucla.edu, jimmypry@g.ucla.edu
haiqixiao@ucla.edu, tjyifengzhang@outlook.com

Abstract

We perform both Convolutional neural network (CNN) and Recurrent neural network (RNN) on the Electroencephalography (EEG) dataset. The CNN model achieves training accuracy of around 83% and test accuracy of 67%. While for RNN model, we get training accuracy of around 96% and test accuracy of 51%.

1. Introduction

We approach the problem with two algorithms, Convolutional neural network and Recurrent neural network. The insight is to compare two algorithms, where CNN performs better on image, and RNN performs better on time-series signal.

1.1 Convolutional neural network

Our CNN model is based on the Shallow ConvNet model proposed in the paper [1]. We have tried four models in total, but here we only give motivations of the architecture for the best CNN model (Model 1).

We initialize the model with a convolutional layer along the temporal dimension, followed by another convolutional layer dealing with 22 electrodes. Then we introduce the nonlinearity into the model by making squaring, mean pooling, and log transformations. Afterwards, we implement a dropout layer to alleviate overfitting problem and make the model generalize better. Finally, we perform a fully-connected layer and softmax function to classify input into 4 movement classes.

Specifically, we first use 40 convolution filters of kernel size (1, 40) with stride (1, 2) and no padding to focus on the temporal dimension. Compared with the original architecture, we increase the kernel size from 25 to 40 as well as enhance the stride from 1 to 2 to drastically reduce the activation map size. The reason for this is that we have more data along the temporal dimension than what's in the paper and we want to reduce parameters in the following fully-connected layer and alleviate overfitting problem. Besides, by using larger kernel size, we find patterns in a larger patch by allowing a larger range of transformations. In addition, we choose the second convolutional layer to have 40 filters of kernel size (22, 1) with stride (1, 1) and no padding to deal with electrode dimension. Unlike the original Shallow ConvNet architecture, we utilize kernel size of 22 to make it consistent with the number of electrodes in our dataset. Moreover, we follow each convolution operation with a Batch Normalization layer to avoid the influence of input scale from previous layers.

Furthermore, we keep the implementation of square, mean pooling and log layer the same as in the paper. After the log transformation, we introduce a dropout layer with a dropping rate 0.3 to regularize the CNN model, which is not included in the original model. We tune the hyperparameter of dropout rate and find that 0.3 performs best. The reason why higher dropout rates don't work well may be because it drops too much nodes and loses a lot of information. The details of this architecture are shown in the Appendix II.

Besides for this architecture, we also test some other architecture, including adding more convolutional layer or adding some features based on the original dataset using Discrete Fourier transform (DFT) [2]. The performances and the details of these architectures can be further shown in the Appendix I and Appendix. And they are not performing well comparing to the architecture we show previously in this part.

1.2 Recurrent neural network

First, we try applying LSTM model on dataset with brute force by feeding the raw data into LSTM layer. After that we employ one fully connected layer and one non-linear activation layer as a transition stage between the final output layer and LSTM. The transition stage's functionality is to reduce the feature dimension and grab the most effective information. The raw data means our input shape to LSTM is (1000,22) so that we have 1000-time steps in LSTM layer and in every time step we feed (22,1) data into the model. As a result, the model always overfits the dataset after 1 epoch or 2 but the validation accuracy wanders around 30%. We conjecture that the training

dataset is too small to play with and because of the small scale and noisy dataset, we're kind of overfitting a small dataset with low generality. For this reason, the testing accuracy sucks.

We find that 1000 timestamps are too many to fit into LSTM model so we apply a time window to the sequence of brain electrical wave in order to strengthen the relations between the wave intensity of nearby time window. The windows size d and the number of slides n are all parameters to change and test. Then for every trial we get a three-dimensional tensor of shape (n, d,22). Since LSTM only accepts data of shape (sample, timestamp, features), we reshape the tensor for every trial to be (n, d*22) and feed it into the LSTM model.

Furthermore, stacked LSTMs tend to be useful to improve accuracy in time-series classification. Each LSTM layer outputs a sequence of vectors which will be used as an input to a subsequent LSTM layer. This hierarchy of hidden layers enables more complex representation of our time-series data, capturing information at different scales.

2. Results

Overall, both CNN and RNN models achieve good performance for movement class classification. By comparison, CNN performs better than RNN.

2.1 Convolutional neural network

We have tried four different CNN models. The details of result for each model can be found in Appendix I. Only the training history and testing result of the best CNN model (Model 1) is shown below.

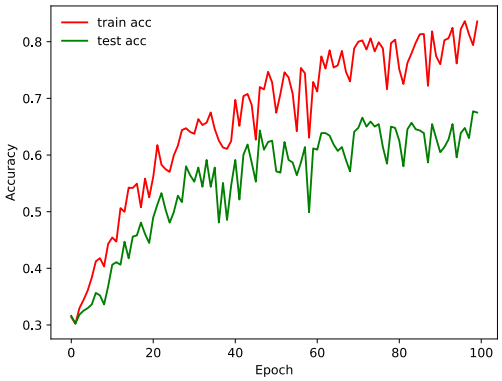

Figure 1: Training and Testing Accuracy of CNN Model 1

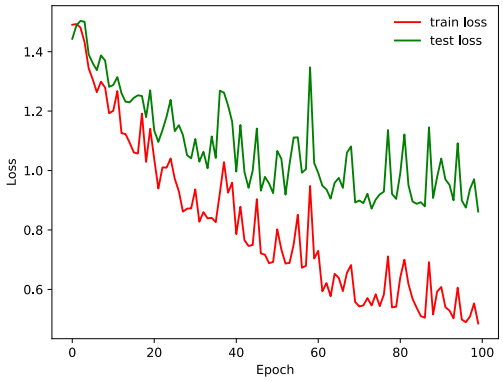

Figure 2: Training and Testing Loss of CNN Model 1

Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
83.59%	67.51%	0.4852	0.862

Table 1: Loss and Accuracy of CNN Model 1

2.2 Recurrent neural network

We have developed Simple RNN model and Stacked RNN model. The details of result for two models can be found in Appendix I. Only the training history and testing result of Stacked RNN on overall data set is shown below.

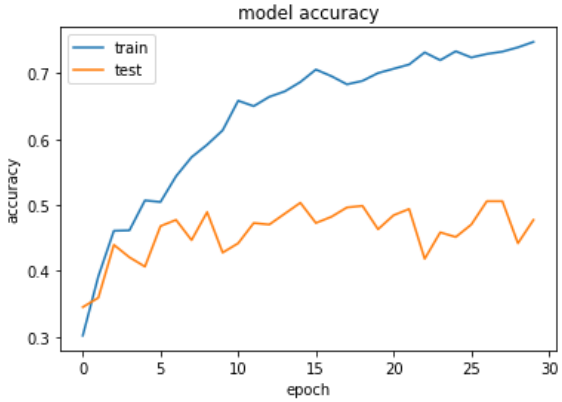

Figure 3: Training and Testing Accuracy of Stacked RNN

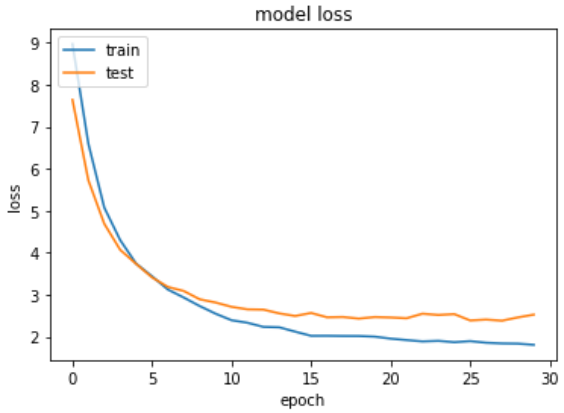

Figure 4: Training and Testing Loss of Stacked RNN

Training Accuracy	Testing Accuracy	Training Loss	Testing Loss
95.2%	50.4%	1.4952	4.8712

Table 2: Inter-Person Loss and Accuracy of Stacked RNN

2.3 Comparison between CNN and RNN

Generally, RNN is more suitable for dealing with time-series data where EEG dataset falls into, but from our results above, we can see that CNN performs better than RNN for the test dataset. It is very likely that the strong overfitting problem of RNN indicated by the big gap between training and testing accuracy lead to this unsatisfactory performance for EEG dataset. We try to alleviate the overfitting problem of RNN by adding a strong L2 regularization and several dropout layers, but the overfitting problem is still worrisome due to the size training dataset.

3. Discussion

3.1 Convolutional neural network

We have tried CNN models with more layers, but it performs worse than the shallow one. We think it might be because deep CNN introduces more parameters but our dataset does not have enough data to train it, which may lead to overfitting problem.

The result for the dataset using the frequency domain dataset also suggests the same reason, too many parameters can lead to poor performance. For the Model 3, we expand the number of channels from 22 to 44, indicating double number of parameters. In this case, the model can hardly perform well, which is due to insufficient training data.

In addition, we also try to reduce parameters for our shallow CNN by increasing the stride steps as well as enhancing the kernel size to reduce the activation map size. We find that having less parameters can efficiently improve our model performance. We believe that the gain is also due to alleviation of overfitting, which makes model generalize better.

3.2 Recurrent neural network

First, we train neural nets on the first subject and test on the whole dataset. Training accuracy on the first person is 50% while the test accuracy is 38%. This result is expected because electrode pattern differs for different subjects. We do figure out some shared features but these aren't enough to serve for all subjects.

We then try optimizing the network over all the training data and find that it is more difficult to fit the whole training dataset than inter-person data. In the inter-person training, we add considerable parameter penalties to prevent overfitting while in overall training, we have to remove those penalties to increase the training performance. This again proves our guess that 9 subjects possess different patterns. While optimizing hyperparameters, we find increasing window length will decrease training accuracy while increase testing accuracy. This does indicate that some useful features may be encoded in time series analysis and classical features like autocorrelation analysis may be helpful, but we don't have enough time to try these.

References

[1] 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.

[2] B. E. Oran. The fast Fourier transform and its applications. Englewood Cliffs, N.J.: Prentice Hall, 1988.

Appendix I: The Summary of the Performance of CNN and RNN models

1. Convolutional Neural Network

1.1 Summary of Performance of all the CNN Models We Test

CNN Model	**Training/Test Accuracy**	**Training/Test Loss**
CNN Model 1 (The best CNN model)	0.8359/0.6751	0.4852/0.8620
CNN Model 2	0.8411/0.6591	0.4452/10.949
CNN Model 3	0.8823/0.6298	0.3693/10.983
CNN Model 4	0.7144/0.4786	1.0234/1.2495

* This table just show the result for four kinds of CNN model we tested, where the detailed architectures for these models can be found in Appendix II.

1.2 Inter-person Performance of the Best CNN Model (Model 1)

Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Average Inter-person Test Accuracy**	**Test Accuracy Using the Whole Dataset**
CNN Model 1 (The best CNN model)	0.66	0.46	0.82	0.64	0.5957	0.5714	0.68	0.74	0.78	0.6616	0.6751

* In the inter-person test, for each person, we use its corresponding data to train our best CNN model (Model 1) and test accuracy for each people, then calculate the average inter-person accuracy.

2. Recurrent neural network

2.1 Summary of Performance of all the RNN Models We Test

RNN Model	**Training/Test Accuracy**	**Training/Test Loss**
RNN Mode1 1 (Simple RNN)	Inter-person: 0.9712/0.3450 All: 0.6202/0.3340	Inter-person: 3.7660/4.8712 All: 2.4711/3.0640
RNN Model 2 (Stacked RNN)	Inter-person: 0.9520/0.5040 All: 0.7476/0.4605	Inter-person: 1.4952/1.5211 All: 1.8052/2.5248

* This table shows the result for two kinds of RNN models we test, where the detailed architectures for two models can be found in Appendix II. In the inter-person test, for each person, we use its corresponding data to train the RNN model and test accuracy for each people, then calculate the average inter-person accuracy.

2.2 Inter-person Performance of the RNN Models

Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Average Inter-person Test Accuracy**	**Test Accuracy Using the Whole Dataset**
RNN Mode1 1 (Simple RNN)	0.38	0.32	0.30	0.48	0.319	0.347	0.42	0.26	0.277	0.345	0.334
RNN Model 2 (Stacked RNN)	0.50	0.40	0.52	0.44	0.523	0.51	0.52	0.54	0.574	0.504	0.46

* In the inter-person test, for each person, we use its corresponding data to train the RNN model and test accuracy for each people, then calculate the average inter-person accuracy.

Appendix II: The Architecture of CNN and RNN Models

1. Convolutional Neural Network

CNN Model	Training Parameter	Architecture
CNN Mode1 1 (The best shallow CNN model we get)	Xavier Initializer Batch Size: 211 Learning rate: 0.005 AdamOptimizer Decay rate: 0.975 (after epoch > 40) Epoch: 100	Input: 2115 * 22 * 1000 * 1 **Conv (stride: (1, 2), kernel: (1, 40), filter size: 40)** Activation Map: 2115 * 22 * 481 * 40 **Batch Normalization (momentum: 0.1)** **Conv (stride: (1, 1), kernel: (22, 1), filter size: 40)** Activation Map: 2115 * 1 * 481 * 40 **Batch Normalization (momentum: 0.1)** **Square Activation** **Average Pool (stride: (1, 15), kernel: (1, 75))** Activation Map: 2115 * 1 * 28 * 40 **Dropout (p: 0.3)** **Fully Connect** Output: 2115 * 4 Softmax
CNN Model 2 (The original shallow CNN model proposed in [1])	Xavier Initializer Batch Size: 211 Learning rate: 0.005 AdamOptimizer No decay rate Epoch: 100	Input: 2115 * 22 * 1000 * 1 **Conv (stride: (1, 1), kernel: (1, 40), filter size: 40)** Activation Map: 2115 * 22 * 961 * 40 **Batch Normalization (momentum: 0.1)** **Conv (stride: (1, 1), kernel: (22, 1) filter size: 40)** Activation Map: 2115 * 1 * 961 * 40 **Batch Normalization (momentum: 0.1)** **Square Activation** **Average Pool (stride: (1, 15), kernel: (1, 75))** Activation Map: 2115 * 1 * 61 * 40 **Dropout (p: 0.3)** **Fully Connect** Output: 2115 * 4 Softmax
CNN Mode1 3 (The shallow model with extra frequency domain transferred channel)	Xavier Initializer Batch Size: 105 Learning rate: 0.005 AdamOptimizer Decay rate: 0.975 (after epoch > 40) Epoch: 100	Input: 2115 * 44 * 1000 * 1 (for each channel, we used DFT to obtain the signal in frequency domain, and used it as extra features) **Conv (stride: (1, 1), kernel: (1, 40), filter size: 40)** **Batch Normalization (momentum: 0.1)** Activation Map: 2115 * 44 * 961 * 40 **Conv (stride: (1, 1), kernel: (44, 1) filter size: 40)** **Batch Normalization (momentum: 0.1)** Activation Map: 2115 * 1 * 961 * 40 **Square Activation** **Average Pool (stride: (1, 15), kernel: (1, 75))** **Dropout (p: 0.3)** Activation Map: 2115 * 1 * 61 * 40 **Fully Connect** Output: 2115 * 4 Softmax
CNN Mode1 4 (The CNN with one more convolute layer comparing to Model 1)	Xavier Initializer Batch Size: 211 Learning rate: 0.005 AdamOptimizer Decay rate: 0.975 (after epoch > 40) Epoch: 200	Input: 2115 * 22 * 1000 * 1 **Conv (stride: (1, 1), kernel: (1, 25), filter size: 40)** Activation Map: 2115 * 22 * 976 * 40 **Batch Normalization (momentum: 0.1)** **Conv (stride: (1, 1), kernel: (22, 1) filter size: 40)** Activation Map: 2115 * 1 * 976 * 40 **Batch Normalization (momentum: 0.1)** **Max Pool (stride: (1, 6), kernel: (1, 6))** Activation Map: 2115 * 1 * 162 * 40 **Conv (stride: (1, 1), kernel: (1, 23) filter size: 40)** Activation Map: 2115 * 1 * 140 * 40 **Batch Normalization (momentum: 0.1)** **Square Activation** **Max Pool (stride: (1, 5), kernel: (1, 5))** Activation Map: 2115 * 1 * 28 * 40 **Dropout (p: 0.6)** **Fully Connect** Output: 2115 * 4 Softmax

* The architectures in the table represent the best architectures and the best training parameters among each category architectures (1. The shallow convolutional net with significantly less parameters in the last fully connected layer; 2. The original shallow net proposed in the previous study; 3. The shallow net with frequency domain channels; 4. A deep convolutional net.)

* For Model 3, we use DFT to extend from 22 channels to 44 channels, where 1 – 22 channels are original signals in the time domain, and 23 – 44 channels are the corresponding signals in the frequency domain.

2. Recurrent Neural Network

RNN Model	Hyper Parameter	Architecture
RNN Mode1 1 (Simple RNN)	Batch Size: 50 Learning Rate: 0.001 adamOptimizer epoch: 30	**LSTM, units: 64** **Dropout (p: 0.5)** **Fully Connected, Output dimension: 4** Softmax
RNN Model 2 (Stacked RNN)	Batch Size: 50 Learning Rate: 0.001 adamOptimizer epoch: 30	**Bidirectional LSTM, units: 64** **Dropout (p: 0.5)** **Bidirectional LSTM, units: 64** **Flatten** **Dropout (p: 0.5)** **Fully Connected, Output dimension: 32** **Dropout (p: 0.5)** **Fully Connected, Output dimension: 4** Softmax