EEG classification with Deep Learning Methods

Yining Wang
504983099
UCLA
Los Angeles, CA
wangyining@g.ucla.edu

Yiheng Jian
705625602
UCLA
Los Angeles, CA
jianyiheng@ucla.edu

Abstract

This project aims at doing a classification task on the EEG [1] dataset. We employed deep learning architectures including Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and CNN with RNN. We also experimented with training using data of 1 subject vs. all subjects, training with different time window sizes, and data augmentation.

1. Introduction

We chose to implement three architectures: CNN, LSTM, and CNN with LSTM. The details of each architecture could be found in Appendix B.

1.1. CNN

We first chose to employ CNN to do the classification task. Because EEG data are in the shape of (batch, electrode, time), for each sample the points on the surface (electrode, time) are relevant to the nearby points. We can thus use CNN to extract such spatial information.

We noticed that CNN comes in different flavors. We chose to implement one shallow CNN and one deep CNN, both from the Schirrmeister paper "Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG". [2]

1.1.1. Shallow CNN

We first implemented the shallow version of CNN. It consists of a convolution layer with ELU, followed by a fully connected layer with ELU, a square function, an average pool layer, a log function, and a SoftMax layer. The total number of convolution layer is 1.

1.1.2. Deep CNN

The deep CNN consists of one convolutions layer with ELU, then followed by four consecutive combos of a convolution layer with ELU followed by a max pooling layer, and then followed by a SoftMax layer. The total number of convolution layer is 5.

1.2. RNN

A recurrent neural network (RNN) is a class of artificial neural networks. It is very powerful because it combines two properties: (1) distributed hidden state that allows it to store a lot of information about the past efficiently. (2) Non-linear dynamics that allows it to update its hidden state in complicated ways. Since EEG data has the time dimension, we think it might be a good idea to use RNN models on it.

1.2.1 Simple RNN

Simple RNN network in this project is composed of 2 RNN layers and 1 fully connected layer.

1.2.2 LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture. A common LSTM unit is composed of a cell, and input gate, an output gate and a forget gate. It is developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.

LSTM network in this project contains 2 LSTM layers and 1 fully connected layer.

1.2.3 GRU

Gated recurrent units (GRUs) are a gating mechanism in RNN. It has higher performance on certain smaller and less frequent datasets.

GRU network in this project contains 2 GRU layers and 1 fully connected layer.

1.3. Combined Network

Since it makes sense to use both CNN and RNN model on EEG data, naturally we think it might be a good idea to combine the two. We decoded to implement CNN-RNN

models to combine the power of two and see if it gives a better outcome.

1.3.1 CNN-Simple RNN

This network is composed of 1 CNN layer, 1 GRU layer and 1 fully connected layer. The CNN layer contains a convolution layer followed by a batch normalization layer and 1 ReLU layer.

1.3.2 CNN_LSTM

This network is similar to CNN-Simple RNN network. It is composed of 1 CNN layer and 1 LSTM layer.

1.3.3 CNN GRU*:*

This network contains 1 CNN layer and 1 GRU layer.

2. Results

Detailed results could be found in Appendix A.

2.1 Comparison of Architectures

We trained different flavors of CNNs, RNNs, and CNN with RNNs and then compared their test accuracy.

Within the CNNs, Shallow CNN has a better performance than deep CNN.

All the RNN models have terrible performance Among them, LSTM network achieves the highest accuracy while RNN is the worst.

As for the CNN with RNN combined models, the CNN-RNN network has the best performance and CNN-GRU gets the worst.

If we compare all the models together, shallow CNN and CNN with RNN models have the best performance.

2.2 Single Subject vs. All Subjects

We compared the performance of models trained with subject 1 data to those trained with all data, with the same goal of classifying subject 1 data.

When using shallow CNN, training with all data turns out to perform better.

Also, when training shallow CNN with all data, the accuracy of classifying subject 1 data is higher than that of classifying all data.

RNN, LSTM and CNN with RNN combined networks have better performance on dataset with all subjects. In contrast, GRU network on single subject outperforms that on all subjects.

2.3 Different Time Windows

We used a sliding window technique to transform the original data of length 1000 into datasets with length 500 and 300, and then compared the performance of models trained with those different length of dataset.

According to the result of experiments, models with larger window size outperform those with smaller window size.

2.4 Data Augmentation

We applied data augmentation to the data by trimming, max pooling, adding noise to, and subsampling the data. We then trained a CNN with LSTM model with the augmented data. It has a better accuracy than CNN with LSTM trained on dataset without preprocess.

3. Discussion

This section analyses the possible reasoning behind the results presented in section 2. Each subsection corresponds to the subsection with the same index in section 2.

3.1 Comparison of architectures

Within the CNNs, shallow CNN works better because deeper CNNs would have problems with overfitting and vanishing gradients.

Among the RNNs, because LSTM and GRU have ability to deal with the vanishing gradient problem, therefore they could be trained for more epochs to achieve higher performance. On the other hand, vanilla RNN has a bad performance because it suffers from the vanishing gradient problem cause by the large time span of the dataset.

3.2 Single Subject vs. All Subjects

When using shallow CNN, classifying on subject 1 data with a model trained with all the data performs better than one trained with subject 1 data only.

One possible reason is that the distribution of the data from all subjects is similar. Therefore, training with all data not only gives a reasonable model still, but also reduces the problem of overfitting.

3.3 Different Time Windows

For both tested models, performance is significantly worse when the model is trained with a shorter time window. It makes sense because data with a shorter time window contains less consequent information, which a convolution layer would fail to extract.

3.4 Data Augmentation

Our models perform better when the data used to train it is augmented. The reason is that since the number of samples is low, overfitting could easily happen. Training with augmented dataset reduces the chance of overfitting significantly.

References

[1] Brunner, C; Leeb, R; Muller-Putz, R; Schlogl, A; Pfurtscheller1, G. “BCI Competition 2008 – Graz data set A”

[2] Schirrmeister, Robin Tibor; etc. “Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG”. arXiv:1703.05051v5 [cs.LG]. Jun 8th, 2018.

Appendix A: Model Performance

1. Test Accuracy of Models

Shallow CNN	65.5%
Deep CNN	49.2%
Simple RNN	24.3%
LSTM	28.4%
GRU	26.4%
CNN with RNN	66.8%
CNN with LSTM	68.3%
CNN with GRU	52.7%

2. Test Accuracy of Models Trained with Subject 1 Data vs. All Data

Trained with	Shallow CNN	LSTM	RNN	GRU	CNN with LSTM	CNN with RNN	CNN with GRU
Subject 1 Data	64%	22%	22%	32%	62.4%	63.2%	57%
All Data	78%	28.4%	24.3%	26.4%	68.3%	66.8%	52.7%

3. Test Accuracy of Models Trained with Dataset of Different Sizes

Trained with Data of Time Length	Shallow CNN	CNN with LSTM	CNN with GRU	CNN with RNN
1000	65.2%	42%	52.7%	66.8%
500	48.2%	39%	43.5%	34.6%
300	40.1%	24%	42.2%	33.8%

4. Test Accuracy of a CNN with LSTM model trained with Normal Dataset vs. Augmented Dataset

model	CNN with LSTM	CNN with GRU	CNN with RNN
Normal Dataset	42%	52.7%	66.8%
Augmented Dataset	55.7%%	60.6%	49%

Appendix B: Model Architecture

1. Shallow CNN

```
model parameters amount 46204
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 40, 22, 976]           1,040
               ELU-2          [-1, 40, 22, 976]               0
       BatchNorm2d-3          [-1, 40, 22, 976]              80
            Linear-4              [-1, 976, 40]          35,240
               ELU-5              [-1, 976, 40]               0
         AvgPool1d-6               [-1, 40, 61]               0
       BatchNorm1d-7               [-1, 40, 61]              80
            Linear-8                    [-1, 4]           9,764
           Softmax-9                    [-1, 4]               0
================================================================
Total params: 46,204
Trainable params: 46,204
Non-trainable params: 0
```

Optimizer: Adam
Dataset Aumentation: No
Activition Function: ELU

2. Deep CNN

```
model parameters amount 283304
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 25, 22, 991]             275
               ELU-2          [-1, 25, 22, 991]               0
       BatchNorm2d-3          [-1, 25, 22, 991]              50
            Conv2d-4           [-1, 25, 1, 991]          13,775
               ELU-5           [-1, 25, 1, 991]               0
       BatchNorm1d-6              [-1, 25, 991]              50
         MaxPool1d-7              [-1, 25, 330]               0
            Conv1d-8              [-1, 50, 321]          12,550
               ELU-9              [-1, 50, 321]               0
      BatchNorm1d-10              [-1, 50, 321]             100
        MaxPool1d-11              [-1, 50, 107]               0
           Conv1d-12              [-1, 100, 98]          50,100
              ELU-13              [-1, 100, 98]               0
      BatchNorm1d-14              [-1, 100, 98]             200
        MaxPool1d-15              [-1, 100, 32]               0
           Conv1d-16              [-1, 200, 23]         200,200
              ELU-17              [-1, 200, 23]               0
      BatchNorm1d-18              [-1, 200, 23]             400
        MaxPool1d-19               [-1, 200, 7]               0
           Linear-20                    [-1, 4]           5,604
================================================================
Total params: 283,304
Trainable params: 283,304
Non-trainable params: 0
```

Optimizer: Adam
Dataset Aumentation: No
Activition Function: ELU

3. RNN network

All RNN networks use Adam as Optimizer, ReLu as activision function, and does not use data augmentation.

3.1 Simple RNN

```
RNN_RNN(
  (rnn): rnn(22, 64, num_layers=2, dropout=0.6)
  (fc): Squential(
    (0): Linear(in_features=64, out_features=54,
    bias=True)
    (1): BatchNorm1d(54, eps=1e-05, momentum=0.2,
    affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=54, out_features=44,
    bias=True)
    (5): BatchNorm1d(44, eps=1e-5, momentum=0.2,
    affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Linear(in_features=44, out_features=4,
    bias=True)
  )
)
```

3.2 LSTM

```
RNN_LSTM(
  (lstm): lstm(22, 64, num_layers=2, dropout=0.6)
  (fc): Squential(
    (0): Linear(in_features=64, out_features=54,
    bias=True)
    (1): BatchNorm1d(54, eps=1e-05, momentum=0.2,
    affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=54, out_features=44,
    bias=True)
    (5): BatchNorm1d(44, eps=1e-5, momentum=0.2,
    affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Linear(in_features=44, out_features=4,
    bias=True)
  )
 )
```

3.3 GRU

```
RNN_GRU(
  (gru): gru(22, 64, num_layers=2, dropout=0.6)
  (fc): Squential(
    (0): Linear(in_features=64, out_features=54,
    bias=True)
    (1): BatchNorm1d(54, eps=1e-05, momentum=0.2,
    affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
```

```
    (4): Linear(in_features=54, out_features=44,
    bias=True)
    (5): BatchNorm1d(44, eps=1e-5, momentum=0.2,
    affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Linear(in_features=44, out_features=4,
    bias=True)
  )
)
```

4. CNN with RNN

All CNN with RNN networks use Adam as Optimizer and ReLu as activision function. Among them, CNN with LSTM was trained with augmented once and normal data another time.

4.1 CNN+ Vanilla RNN

```
RNN_RNN(
  (cnn): Sequential(
    (0): Conv1d(22, 25, kernel_size=(10,), stride=(2,))
    (1): BatchNorm1d(25, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
    (2): ReLU()
)
  (rnn): rnn(22, 64, num_layers=2, dropout=0.6)
  (fc): Squential(
    (0): Linear(in_features=64, out_features=54,
    bias=True)
    (1): BatchNorm1d(54, eps=1e-05, momentum=0.2,
    affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=54, out_features=44,
    bias=True)
    (5): BatchNorm1d(44, eps=1e-5, momentum=0.2,
    affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Linear(in_features=44, out_features=4,
    bias=True)
  )
)
```

4.2 CNN+LSTM

```
RNN_LSTM(
  (cnn): Sequential(
    (0): Conv1d(22, 25, kernel_size=(10,), stride=(2,))
    (1): BatchNorm1d(25, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
    (2): ReLU()
)
  (lstm): lstm(22, 64, num_layers=2, dropout=0.6)
  (fc): Squential(
    (0): Linear(in_features=64, out_features=54,
    bias=True)
    (1): BatchNorm1d(54, eps=1e-05, momentum=0.2,
    affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=54, out_features=44,
    bias=True)
    (5): BatchNorm1d(44, eps=1e-5, momentum=0.2,
    affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Linear(in_features=44, out_features=4,
    bias=True)
  )
)
```

4.3 CNN+GRU

```
RNN_GRU(
  (cnn): Sequential(
    (0): Conv1d(22, 25, kernel_size=(10,), stride=(2,))
    (1): BatchNorm1d(25, eps=1e-05, momentum=0.1,
    affine=True, track_running_stats=True)
    (2): ReLU()
)

  (gru): gru(22, 64, num_layers=2, dropout=0.6)
  (fc): Squential(
    (0): Linear(in_features=64, out_features=54,
    bias=True)
    (1): BatchNorm1d(54, eps=1e-05, momentum=0.2,
    affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=54, out_features=44,
    bias=True)
    (5): BatchNorm1d(44, eps=1e-5, momentum=0.2,
    affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): Linear(in_features=44, out_features=4,
    bias=True)
  )
)
```