Classification on EEG Dataset using CNN and RNN

Meng Chai , Yinxue Xiao , Ruomei Ye
805034623, 904581627, 005030209
University of California, Los Angeles
joychaimeng@gmail.com, yolandaxiao7@gmail.com, ruomei@ucla.edu

Abstract

Deep learning becomes one of the most popular topics and deep learning algorithms use the neural network, which is modeled loosely after the human brain, to find associations between a set of inputs and outputs. The input data could varies from image, signal, sound and test and Neural networks help cluster and classify input data. In this paper, the team intended to classify four different patterns from electroencephalography(EEG) dataset, which is obtained from human brain signals. In order to discover the relationship between EEG dataset and patterns, the team designed different neural network models based on Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN). Three different architectures were used, including RNN, CNN and Convolutional Recurent Neural Network (CRNN). All three network based models achieved a testing accuracy of 67%.

1. Introduction

The EEG dataset consists of time series data inputs based on 22 electrodes and 1000 time steps, which spans 4 seconds. Since it recorded 22 neural signals within a specific duration, the team came up with an idea to use an RNN structure to implement the classifier. Considering vanilla RNN model is potentially gradient exploded or vanishing, the team replaced it with LSTM and GRU units.

The team also used CNN to perform the classification task since the EEG data could be visualized as 2D images of size 22 x 1000. The AlexNet [2] architecture was used at first but failed to achieve a high accuracy. The team then modified the structure to fit the EEG data better, resulting in two architectures including a 2D convolution and 1D convolution model.

After training both RNN and CNN models, it was natural for the team to integrate them into a combined network and benefit from the advantages of both. The team therefore developed a Convolutional Recurrent Network (CRNN). This CRNN model was loosely based on the previous designed 1D CNN model and RNN model with modifications.

2. Results

2.1. RNN

The RNN architecture with double-layer LSTM model resulted in a 62.5% testing accuracy for entire time steps and a 65% testing accuracy for first 500 time steps out of 1000. For the first 250 time steps , the model resulted in a 60% testing accuracy. When training the double-layer LSTM model with one subjects data, the model achieved the testing accuracy at 55.7%.

In the training based on different periods of time steps, the RNN model based on GRU network achieved the testing accuracy of 63.8%, 67.2% and 60% corresponding to the first 250 time steps, 500 time steps and 1000 time steps of the input data. The model trained with one subject input data achieved 40% testing accuracy.

2.2. CNN

In the case of training a classifier for all subjects, the CNN architecture with 2D convolutions resulted in a 64.56% testing accuracy, while the other with 1D convolutions resulted in a 67.27% testing accuracy.

In the case of training a classifier for only the first subject, the 1D CNN architecture was able to achieve a 52% testing accuracy, while the 2D one could only achieve a 26% testing accuracy.

The team also trained the CNN model based on a variety of input time lengths. The classification accuracy vs. the corresponding time sequence is plotted in Figure 6, with highest accuracy 67% being the data that consists of entire 1000 time steps.

2.3. CNN + RNN

The CRNN model was implemented with 1D CNN model and 3-layer LSTM/GRU model. For all subjects with entire 1000 time steps, the testing accuracy of CNN_3-layer LSTM model achieved 67%, and CNN_3-layer GRU model achieved 65%. When developing the model, the team also tried CNN_1-layer LSTM and CNN_2-layer LSTM models, which yielded testing accuracy 60.09% and 64.3%, slightly lower than the 3-layer CRNN model.

The team trained the model with only the first subject input data, and found that the testing accuracy could only reach about 50% for both CNN-LSTM/GRU models.

The team also recorded the testing accuracy vs. data over different periods of time. The table is data is shown in Figure 6. The results will be analyzed in Section 3.

3. Discussion

3.1. RNN

Considering EEG data was collected in a duration of a particular event, the team first came up with RNN model, which is a deep learning model based on time sequence. The team first built a model with double-layer LSTM network since standard RNN network has problems that the gradient of the loss function decays exponentially with time. LSTM network uses the 'memory cell' and effectively avoids the vanishing gradient problem. Another reason why double-layer LSTM network was picked was that the result of one layer network was not complex enough whereas the deep LSTM network was unable to learn from the lost function since the input data was not sufficient enough. Another problem during the training was that the model tended to over-fitting after the third epoch of training and in order to avoid the over-fitting problem, L2 Regularization was added to each of the layer and drop out over 80% of output from the previous layer. A fully connected layer was followed LSTM layer with a Softmax activation. For loss function, both Cross Entropy and Mean Square Error were used, and they performed almost the same for the testing accuracy result. Moreover, in order to balance the learning process, Aadm optimizer was used and the learning rate was 0.0005 with decay equaled to 0.001. This setting helped to speed up the learning at the beginning but decrease the learning step as epoch increased.

The double-layer LSTM model trained with the first subject and entire time steps achieved a relatively low accuracy of 55.7% since there was not enough input data. The model trained with all subject and entire time steps input achieved maximumly 62.5%. Since the 1000 time steps input data includes Cue and Motor Imaginary in the experiment, the team divided the 1000 time steps into 4 groups to discover how input training data influence the model result, each containing 250 time steps. The testing accuracy corresponding to each group were 59.8%, 46.0%, 30.5%, 27.5%. It's thus shown that the cue process was the major part to classify the EEG data compared to that of the motor imaginary process. Furthermore, the team trained the model with 500 time steps out of 1000 and the model achieved a testing accuracy of 65.2%, which was slightly higher than the that of 1000 time steps. The result also satisfied the experiment procedure, since the cue happened in the first half of the procedure.

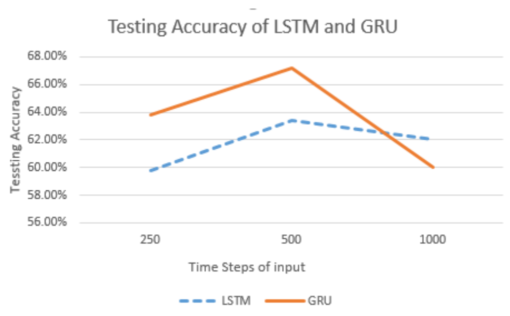


Figure 1. 2-layer LSTM Architecture Overview

The testing accuracy result of both LSTM network and GRU network corresponding to 250 time steps, 500 time steps and 1000 time steps are plotted in figure 1. According to the plot, the model based on GRU had better performance, and less overfitting than LSTM. This is due to the complexity of the inner architecture between LSTM and GRU. The EEG data provided in this project was not extremely sophisticated. GRU in this case had less parameters to train and performs better than LSTM in the double-layer RNN model.

According to the testing accuracy result, the models trained with fist 500 time steps and 1000 time steps input data achieved a relatively high accuracy whereas the latter 500 time steps input data, which was connected with the Motor Imaginary in the experiment would not help much in classification.

3.2. CNN

While the idea of using RNN comes from the fact of time series data, the thought of using CNN comes from visualizing the input data. The team plotted out the value of electrodes in respect to time, as shown in Figure 2,3. It can be seen that the values fluctuate in a certain pattern through time, and all electrodes behave comparatively similarly. The end goal is to classify the 22x1000 inputs into one of the four tasks. Since the inputs can be visualized into 2D images that has different features, convolutional neural network can thus be applied here.

The team first applied the AlexNet [2] architecture to the EEG inputs since it yielded high accuracy for ImageNet classification. However, the performance turned out to be comparatively low, which is around 24% for the testing accuracy.

It's thus important to modify the layers and hyperparameters so that it adapts to the EGG data better. While the setup for both the 2D and 1D convolutions are the same, it'll be discussed together in the following paragraph. The

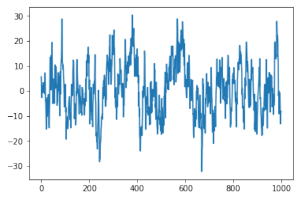


Figure 2. Visualization of 1st Electrode in 1st Example of the 22x1000 Input

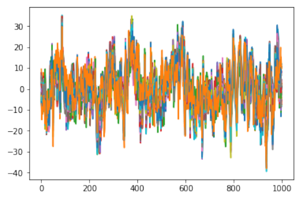


Figure 3. Visualization of All 22 Electrodes in 1st Example of the 22x1000 Input

difference is that the inputs are treated as images in 2D convolutions, so it's essentially doing an image classification task. It makes sense since all the information are encoded in the 22*1000 input itself. The idea of using 1D convolution comes from comparing the visualization of time series data for 1 electrode versus all electrodes. In Figure 2 and 3, the trend for the electrodes are comparatively similar. Thus, most of the features are encoded in the rows instead of the columns. Instead of using 2D convolutions, 1D convolution should lead to similar results, as shown in the section 2.

In the original Alex-Net architecture, there were 2 groups of convolution, max pooling, batch normalization, along with two consecutive convolutions. The team found that changing the two individual convolutions into grouped ones has improved the performance. Max pooling made the detection of features invariant to scale or orientation changes. In this case, it helped to stabilize the result. Batch normalization helped with centering the data. The filter sizes were also increased so that the features could be encoded better.

The number of filters was set to 50. It's tested to be a reasonable value since setting it too high tends to overfit the data to the model, and setting it too low was not sufficient to encode features well enough for a good performance. The number of epochs was set to be 100 so that the model was trained optimally and the validation loss decreased to almost flat line. Since the model had overfitting in the beginning, dropout and regularization was added to prevent it. The training and validation accuracy are thus able to stay relatively close.

3.3. CNN + RNN

In order to leverage the advantages of both RNN and CNN models, the team designed the CRNN model. It made sense as the ultimate goal was to classify the input signals into four given tasks; therefore the convolutional layers could extract the hidden features from electrode signals. Meanwhile, since the time sequence of the electrode signals matters for classification as well, applying the recurrent layers to features which were extracted from convolutional layers was reasonable.

The CRNN model first extracted features using 1D convolutional layers, where the result was good enough for filtering the features as discussed in 3.2. The team then added recurrent layers after it, developing both CNN-LSTM and CNN-GRU models. As learned in lecture, the performance of LSTM and GRU are almost on the same level when the architecture is deep. It's illustrated by the testing accuracy of 67% and 65% respectively for all subjects. During the development of this CNN-3-layer RNN model, the team first tried 1-layer and 2-layer LSTM. It turned out the 3-layer model performed better. This result was expected because the stacked RNN model was deeper and more complicated. To tune the hyper-parameters, the team first directly applied the parameters from RNN and CNN models, and the testing accuracy was only about 50%.

As shown in Figure 6, the accuracy was relatively high when using the first 250, 500, 750 and 1000 time steps that includes the "Cue" part. The accuracy was lower when using later time steps that only includes the "motor imaginary" part. The team believed that first 250 time steps, corresponding to "cue" part, is most important for motor imaginary tasks classification [1]. On the other hand, the second half of time period seemed not so useful for classification. In the case for CNN and CRNN specifically, the accuracy increased as longer periods of data got included, as shown in Figure 5. The data including the entire 1000 time steps tend to be better while that with only 250 time steps were slightly lower.

In general, the Model performance based on RNN, CNN and CRNN could achieve the same level testing accuracy in the end. The input time steps both influenced RNN and CNN models as discussed above, even though the CNN model considered the input signals as a 2D visualization graph. This result indicated that the cue process in the experiment mainly influence the result of classification both in RNN and CNN. Apart from training data would influence the model testing accuracy, the architecture was also important to testing accuracy result. Regularization and drop out would effectively prevent over-fitting and proper learning rate and decay parameter would increase the learning process and testing result.

References

[1] C. Brunner, R. Leeb, G. Müller-Putz, A. Schlögl, and G. Pfurtscheller. Bci competition 2008–graz data set a. *Institute for Knowledge Discovery (Laboratory of Brain-Computer Interfaces), Graz University of Technology*, 16, 2008.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

4. Performance Summary

Architectures	All Subjects			First Subject
	First 250 Time Steps	First 500 Time Steps	All 1000 Time Steps	All 1000 Time Steps
2 Layer LSTM	59.80%	63.43%	62.00%	43.70%
2 Layer GRU	**63.80%**	**67.20%**	60.00%	40.00%
1D CNN	56.43%	63.88%	**67.27%**	**52.00%**
2D CNN	58.69%	61.62%	64.56%	26.00%
1D CNN +1 layer LSTM	N/A	N/A	61.90%	50.00%
1D CNN +2 layer LSTM	N/A	N/A	64.30%	50.00%
1D CNN +3 layer LSTM	61.85%	62.50%	**67.30%**	58.00%
1D CNN +3 layer GRU	56.88%	62.75%	65.46%	50.00%

Figure 4. Testing Accuracy of Different Architectures

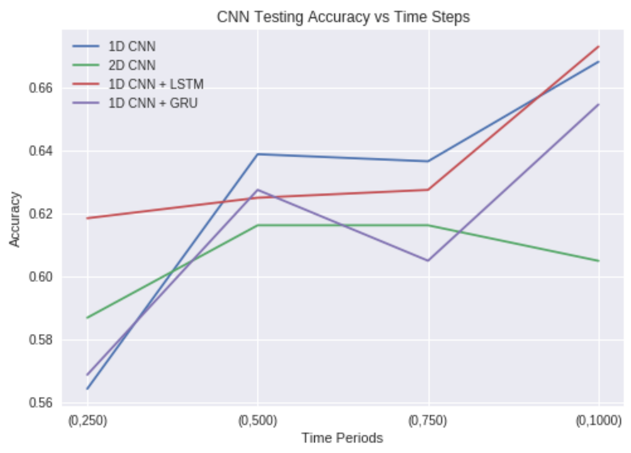


Figure 5. Testing Accuracy vs. Timestep Periods

Architectures\Testing Accuracies	All Subjects(Time Steps)					
	0-250	0-500	0-750	0-1000	250-100	750-1000
1D CNN	56.43%	63.88%	63.66%	**66.82%**	51.24%	28.44%
2D CNN	58.69%	61.62%	61.62%	60.49%	47.62%	28.21%
1D CNN +3 layer LSTM	61.85%	62.50%	62.75%	**67.30%**	52.14%	35.44%
1D CNN +3 layer GRU	56.88%	62.75%	60.50%	65.46%	49.40%	37.00%

Figure 6. Testing Accuracy vs. Timestep Periods

5. Architecture Summary

5.1. RNN

The double layer LSTM/GRU model is shown in figure 7, 100 memory cells were used for both layers. The input size is (batch size * time steps * number of electrodes), which is (50 x 1000 x 22) in this case. The L2 regularization is set to 0.01 and the batch size is 50. Adam is chosen as the optimizer, with learning rate as 0.0005 and decay as 0.001.

---

1. [50x1000x22x100] **LSTM**/ **GRU:** 100 memory cell with input size as 22 x 1000
2. [50x1000x22x100] **DROP OUT:** drop out equals to 0.8

---

3. [50x1000x22x100] **LSTM/GRU:** 100 memory cell with input size as 22 x 1000
4. [50x1000x22x100] **DROP OUT:** drop out equals to 0.8

---

5. [50x2200000] **FLATTEN:** flatten from multi-dimension into 1 dimension
6. [50x4] **FC:** Fully connected into 4 unit
7. [50x4] **OUT:** Softmax activation

Figure 7. 2-layer LSTM Architecture Overview

5.2. CNN

The CNN architectures are modified based on AlexNet [2].

The 1D architecture is shown Figure 8. The input size is (batch size * time steps * number of electrodes), which is (50 x 1000 x 22) in this case. The axis for time steps and number of electrodes are switched to fit the keras function specification for CONV1D. The L2 regularization all have lambda=0.01, and the batch size is 50. The padding is set so that the input size is equal to the output size. Adam is used as the optimizer, with learning rate = 0.001.

---

1. [50x248x50] **CONV 1D**: 50 filters of size 11 with stride 4 with RELU and l2 regularization
2. [50x123x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
3. [50x123x50] **BATCH NORM**: normalization layer
4. [50x123x50] **DROP OUT**: drop out equals to 0.5

---

5. [50x123x50] **CONV 1D**: 50 filters of size 9 with stride 1 with RELU and l2 regularization
6. [50x61x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
7. [50x61x50] **BATCH NORM**: normalization layer
8. [50x61x50] **DROP OUT**: drop out equals to 0.5

---

9. [50x61x50] **CONV 1D**: 50 filters of size 7 with stride 1 with RELU and l2 regularization
10. [50x30x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
11. [50x30x50] **BATCH NORM**: normalization layer
12. [50x30x50] **DROP OUT**: drop out equals to 0.5

---

13. [50x30x50] **CONV 1D**: 50 filters of size 5 with stride 1 with RELU and l2 regularization
14. [50x14x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
15. [50x14x50] **BATCH NORM**: normalization layer
16. [50x14x50] **DROP OUT**: drop out equals to 0.5

---

17. [50x14x50] **CONV 1D**: 50 filters of size 3 with stride 1 with RELU and l2 regularization
18. [50x6x50] **MAX POOLING 1D**: filters of size 3 with stride 2.

---

19. [50x300] **FLATTEN**: flatten out the dimensions into 1 dimensional
20. [50x4] **FC**: Fully connected layer with 4 units (class scores).
21. [50x4] **OUT**: Softmax layer

Figure 8. 1D CNN Architecture Overview

The 2D architecture has similar layers and hyper-parameters to that of 1D, as shown in Figure 9. One difference is that the convolutions are 2D on the entire image. In order to perform 2D convolution, the input size is extended to 4 dimensions into (batch size * number of electrodes * time steps * depth), which is (50 x 22 x 1000 x 1) in this case.

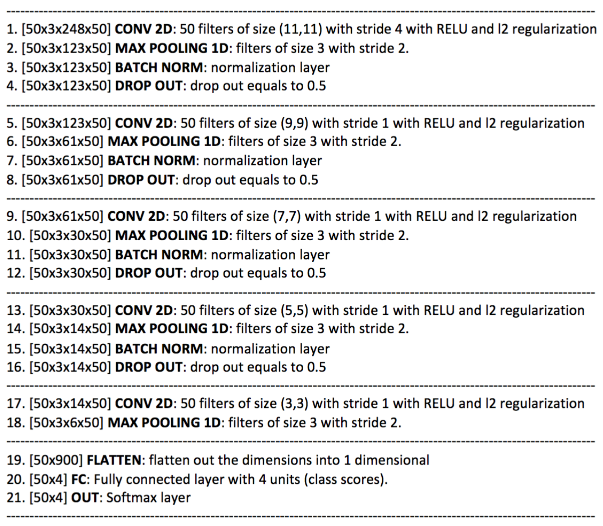
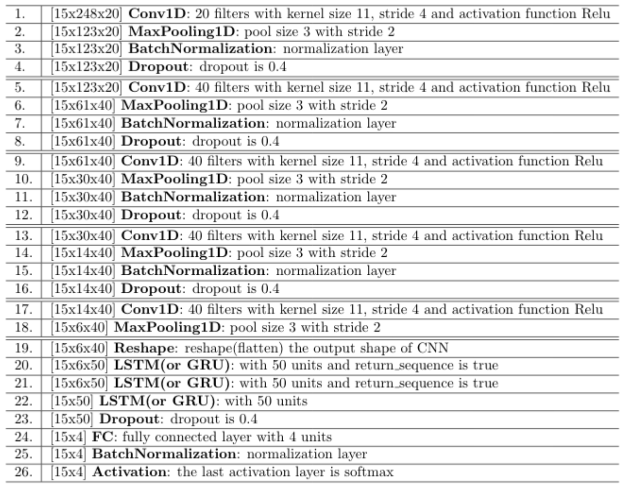

1. [50x3x248x50] **CONV 2D**: 50 filters of size (11,11) with stride 4 with RELU and l2 regularization
2. [50x3x123x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
3. [50x3x123x50] **BATCH NORM**: normalization layer
4. [50x3x123x50] **DROP OUT**: drop out equals to 0.5
5. [50x3x123x50] **CONV 2D**: 50 filters of size (9,9) with stride 1 with RELU and l2 regularization
6. [50x3x61x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
7. [50x3x61x50] **BATCH NORM**: normalization layer
8. [50x3x61x50] **DROP OUT**: drop out equals to 0.5
9. [50x3x61x50] **CONV 2D**: 50 filters of size (7,7) with stride 1 with RELU and l2 regularization
10. [50x3x30x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
11. [50x3x30x50] **BATCH NORM**: normalization layer
12. [50x3x30x50] **DROP OUT**: drop out equals to 0.5
13. [50x3x30x50] **CONV 2D**: 50 filters of size (5,5) with stride 1 with RELU and l2 regularization
14. [50x3x14x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
15. [50x3x14x50] **BATCH NORM**: normalization layer
16. [50x3x14x50] **DROP OUT**: drop out equals to 0.5
17. [50x3x14x50] **CONV 2D**: 50 filters of size (3,3) with stride 1 with RELU and l2 regularization
18. [50x3x6x50] **MAX POOLING 1D**: filters of size 3 with stride 2.
19. [50x900] **FLATTEN**: flatten out the dimensions into 1 dimensional
20. [50x4] **FC**: Fully connected layer with 4 units (class scores).
21. [50x4] **OUT**: Softmax layer

Figure 9. 2D CNN Architecture Overview

5.3. CNN + RNN

The CRNN architecture is inspired by the 1D CNN trained by the team. Similar to CNN model, the input size is batch_size*time_steps*electrode_numbers (which is 15x1000x22 here). The optimizer is Adam, with default learning rate 0.001. Here the CRNN model does not apply any regularizer as in CNN model. The team also chose categorical_crossentropy to calculate loss.

Notice that in the stacked LSTM, the LSTM layer needs to output all its output sequence to the next LSTM layer (except the last LSTM), and therefore return_sequence is set to true for first two layers.



1.	[15x248x20] **Conv1D**: 20 filters with kernel size 11, stride 4 and activation function Relu
2.	[15x123x20] **MaxPooling1D**: pool size 3 with stride 2
3.	[15x123x20] **BatchNormalization**: normalization layer
4.	[15x123x20] **Dropout**: dropout is 0.4
5.	[15x123x20] **Conv1D**: 40 filters with kernel size 11, stride 4 and activation function Relu
6.	[15x61x40] **MaxPooling1D**: pool size 3 with stride 2
7.	[15x61x40] **BatchNormalization**: normalization layer
8.	[15x61x40] **Dropout**: dropout is 0.4
9.	[15x61x40] **Conv1D**: 40 filters with kernel size 11, stride 4 and activation function Relu
10.	[15x30x40] **MaxPooling1D**: pool size 3 with stride 2
11.	[15x30x40] **BatchNormalization**: normalization layer
12.	[15x30x40] **Dropout**: dropout is 0.4
13.	[15x30x40] **Conv1D**: 40 filters with kernel size 11, stride 4 and activation function Relu
14.	[15x14x40] **MaxPooling1D**: pool size 3 with stride 2
15.	[15x14x40] **BatchNormalization**: normalization layer
16.	[15x14x40] **Dropout**: dropout is 0.4
17.	[15x14x40] **Conv1D**: 40 filters with kernel size 11, stride 4 and activation function Relu
18.	[15x6x40] **MaxPooling1D**: pool size 3 with stride 2
19.	[15x6x40] **Reshape**: reshape(flatten) the output shape of CNN
20.	[15x6x50] **LSTM(or GRU)**: with 50 units and return_sequence is true
21.	[15x6x50] **LSTM(or GRU)**: with 50 units and return_sequence is true
22.	[15x50] **LSTM(or GRU)**: with 50 units
23.	[15x50] **Dropout**: dropout is 0.4
24.	[15x4] **FC**: fully connected layer with 4 units
25.	[15x4] **BatchNormalization**: normalization layer
26.	[15x4] **Activation**: the last activation layer is softmax

Figure 10. CRNN with 1D convolutional layers and three-layer LSTM/GRU