Convolutional and Recurrent Neural Networks for EEG classfication

Yifan Zhang
UCLA
ECE
yifanz114@ucla.edu

Edward Nguyen
UCLA
ECE
edwardnguyen65@g.ucla.edu

Abstract

In this project, we used convolutional neural networks (CNN), recurrent neural networks (RNN), and recurrent neural networks with convolutional layers (CNN + RNN) to classify the EEG data. For one person, the average test accuracy achieved using a CNN was 62.7 percent. For all people, the best test accuracy achieved was 69 percent while using a CNN, 43 percent while using a RNN, and 62 percent while using a CNN + RNN.

1. Introduction

1.1. Convolutional Neural Network Architectures

The architectures for the CNN were based off of the Shallow ConvNet architecture presented in the paper [1]. We specifically chose this model over the Deep ConvNet architecture because of limited computing capacity.

Our first model (CNN model 1) begins with a convolutional layer that acts along the temporal dimension. Following this convolutional layer is another convolutional layer that instead acts along the spatial dimension (the dimension of the electrodes). We use an exponential linear unit as our activation function for the outputs of both convolutional layers. Average pooling and dropout are applied afterwards. Finally, there is a fully connected layer and softmax function to convert the output of the previous layers into a prediction of the four different possible classes.

The first convolutional layer is composed of 50 filters with a kernel size (1, 50), stride (1, 3) and no padding. In the paper, they had used a temporal convolutional filter of kernel size (1, 25) with stride (1, 1). We increased our kernel size and stride not only to save memory but also because our data had a temporal dimension of 1000 instead of 534. Thus, we needed to cover more "distance". Initially, we wanted to have a stride (1, 1) but the device used to train lacked sufficient memory to hold the increased number of parameters. In addition, by using a larger kernel size, our filter can observe patterns over a greater area. The second convolutional layer had 50 filters with a kernel size (22, 1), stride (1, 1) and no padding. The modification here was to change the kernel size from 44 to 22 as we had 22 electrodes in our data compared to the 44 in the paper[1]. To avoid the exploding and vanishing gradient problem, we initialized the weights of our convolutional layers using the Xavier uniform initialization and applied batch normalization after each layer. The exponential linear unit activation function and average pooling was kept the same as in the paper. Following the average pooling layer is a dropout layer with a drop rate of 0.3 to introduce some regularization into the model. A lower dropout rate was used because in training higher dropout probabilities performed poorly. We reasoned that with a higher dropout, too much of the network is deactivated so the model lacked sufficient complexity.

Our CNN model 2 and 3 were the same but with some key modifications. In model 2, we increased the number of parameters per filter for the first convolutional layer but shrunk the number of filters. The temporal convolutional layer was changed to have 40 filters with a kernel size (1, 40), stride (1, 2) and no padding. We wanted to observe whether our previous model's filters were overlooking some more local correlations along the time dimension. However, to compensate for this increase in parameters, we needed to decrease the total number of filters. In model 3, we reversed the order of the temporal and spatial layers of model 2. We wanted to observe whether the ordering was important or negligible.

CNN model 4 was exactly the same as CNN model 2. However, model 4 consists of nine different trained models for the nine different subjects. Then we tested the nine models both on the whole test dataset and on the specific subjects corresponding to the subject. The second approach means that each model is meant to predict for only one specific subject in comparison to before where we had one general model to handle all subjects. CNN Model 5 was trained to predict the subject instead of the four different movements.

1.2. Recurrent Neural Network Architectures

The RNN architecture for the RNN were based off of the LSTM models. We experimented with stacked LSTM and bidirectional LSTM models on the EEG dataset. Also, we experimented with the effect of the sliding window. Sliding windows technique is a kind of processing method of concept drift in data streams from paper [2], which has many applications in intrusion detection. The essence of this technique is data update mechanism. The data stream (x) is divided into several parts of data blocks. When the sliding window moves to the next block, a new block is added to the window at intervals, and the oldest block is deleted. Through this dynamic sample selection method, the sample for modeling is updated.

Our first model (RNN Model 1) focused on one layer bidirectional LSTM model. We passed input to the first layer of LSTM together with one batch normalization and one dropout layer. Then we used an exponential linear unit as our activation function and softmax layer for the outputs, which keeps it the same as in CNN models, helping us compare the results.

Our second model (RNN Model 2) is the same as the first model, but the input passed to the LSTM layer is reshaped using window sliding.

Our third model (RNN Model 3) is a stacked bidirectional LSTM model. The difference between the first and the third model is that the third model added one more LSTM layer to the input, while keeping all the other the same.

Our fourth model (RNN Model 4) is exactly the same as the third model, but the input passed to the LSTM layer is reshaped using window sliding.

1.3. Convolutional with Recurrent Neural Network Architectures (CNN + RNN)

We wanted to combine our models from CNN and RNN models, so we designed a hybrid model with a frontend CNN and a backend LSTM. The CNN is used to interpret subsequences of input that together are provided as a sequence to an LSTM model to interpret. We designed one architecture for convolutional with Recurrent Neural Network, but the main difference is the stack number of convolution layers and LSTM layers. Our model 1 has 3 convolution layers and 3 bidirectional LSTM layers. Model 2 increased that number to 3 and 4, and model 3 increased number to 4 and 5.

2. Results

Each epoch is defined as a pass through the data. For each epoch we use k-fold validation with 4 splits. For each train/validation split, we randomly sample a batch of 500 from the current training split to train on. Thus, each epoch consists of 4 training iterations with each iteration being a batch sample of size 500. The training and validation accuracy for an epoch is the average of the training and validation accuracy from the 4 different training/validation splits.

2.1. Convolutional Neural Network

The results from model 1, 2, and 3 are shown in the following table.

Model	1	2	3
Test(%)	62	64	69.1

Table 1. Test Accuracy for CNN Model 1, 2, 3

In our model 4, we trained nine models, one for each subject, and got different two results by using two different approaches. The first was, after training, to test on the data from all subjects. The second was, after training, to test on their own individual data.

Subject	1	2	3	4	5	6	7	8	9
Acc(%)	39	45	41	43	41	46	45	46	47

Table 2. Result of the first approach to test CNN Model 4

Subject	1	2	3	4	5	6	7	8	9
Acc(%)	54	48	74	54	64	53	68	72	77

Table 3. Result of the second approach to test CNN Model 4

In our model 5, the prediction of the subject got test accuracy more than 99%, which means the differences between people is extreme.

2.2. Recurrent Neural Network

We developed two RNN architectures with two different inputs, which are our four RNN models. Those RNN models from 1 to 4 got 32%, 39%, 30%, and 42% for test accuracy. The following figures are from model 4, which got best result.

2.3. Convolutional with Recurrent Neural Network

We developed one architecture with 3 models, and they got 57% 62%, and 41% for those models. The following figures are from model 2, which got best result.

3. Discussion

3.1. Convolutional Neural Network

The motivation for most of our architectures came from the fact that we lacked sufficient domain knowledge. As a result, we could do little pre-processing of the data and had to use our models to gain better insight into the data.

Increasing the number of parameters per filter by changing the temporal layer (going from CNN model 1 to 2) gave

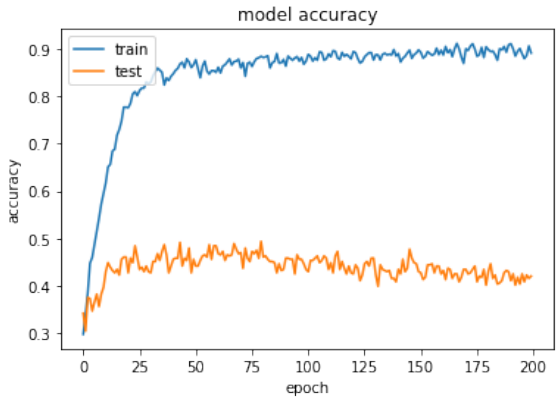

Figure 1. Accuracy of RNN model 4; Note test here means validation accuracy.

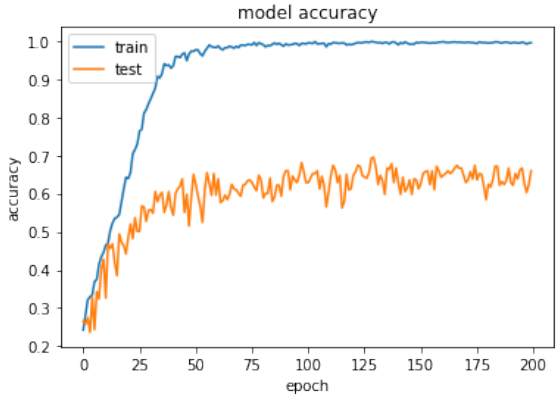

Figure 2. Accuracy of CNN + RNN model 2; Note test here means validation accuracy.

us a 2 percent increase in performance. This negligible increase in performance suggests that simply tuning parameters was not going to give us any significant gains. As a result, we decided to change the order of the convolutional layers i.e place the spatial layer before the temporal layer. This gave us a 5 percent increase in performance. Such a significant gain seems to imply that there is a dependency of the temporal dimension on the spatial dimension. In other words, acquiring information and extracting features of the spatial dimension helps the temporal layer extract more significant features along the time dimension.

We chose to use CNN model 2 as the framework for CNN model 4 and 5 because came up with CNN model 3 much later. We did not have time or computational power to retrain these models. With more time we would have switched to CNN model.

The two testing approaches in our CNN model 4 have average accuracies 43.5% and 62.7%. This result shows that specializing a model for one subject doesn't provide us with a better model. The other CNN models are based on the whole dataset and give us a better result. When we specialize the model for one subject, the accuracy overall does not consistently increase for each subject and even decreases when testing on some subjects. Thus, we find that larger data size gives better result by reducing the effect of noisy data. Also, this seems to imply that for specific subjects there is a larger difference between their movements than other subjects which may explain why for some subjects the test accuracy increases significantly compared to others.

The CNN model 5 which predicts the person has an accuracy over 99%. This tells us that there exists a difference between the EEG signals for each person. However, the information we gained from CNN model 4 suggests that the difference between subjects cannot overcome the lack of data or that for certain subjects there may not be a larger difference in between movements.

3.2. Recurrent Neural Network

From our models, we find that the results from the models with window sliding input is much higher than those without. EEG is a time series data, which is sensitive to unscaled data, and therefore we normalize every minibatch. After several experiments on the hyperparameters, we decide to use 10 as window length and 5 as hop length, because it provides the best value and minimizes data reduction.

3.3. Convolutional with Recurrent Neural Network

In our hybrid models, we find that accuracies from our models firstly increases, and then decrease with the increasing number of convolutional and LSTM layers. This result makes sense that the training begins at underfitting when the number of layers are lower than four, and then goes to overfitting when the number of layers are higher than four.

3.4. Comparing CNN, RNN, and CNN + RNN

For our best models, CNN, RNN, and hybrid models have 67.4%, 42%, and 62% accuracies. It seems that the RNN is disadvantaged to CNN for EEG data. In general, CNN works better for spatial dataset like computer vision, and RNN works better for time series data like speech, because a CNN saves a set of weights and applies them spatially, and RNN has a set of weights applied temporally through time. So for this EEG dataset, we expected RNN to perform better. However, the final result from experiment is different from what we expected. We think the potential reason might be that the data size of EEG dataset is too small for RNN, as RNN requires more inner layers. It's easy for RNN to become overfitting and reduce accuracy.

References

[1] Robin Tibor Schirrmeister, Jost Tobias Springenbergc, Lukas Dominique Josef Fiederera, Martin Glasstettera, Katharina Eggenspergere, Michael Tangermannf, Frank Huttere, Wolfram Burgardg, and Tonio Bal *Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG, June 11, 2018*

[2] Putu Sugiartawan, Reza Pulungan, Anny Kartika Sari *Prediction by a Hybrid of Wavelet Transform and Long-Short-Term-Memory Neural Network, International Journal of Advanced Computer Science and Applications 8(2), February 2017*

Note: N is the number of data points. These are not all the architectures we tested but are rather ones with presentable results.

Model	Training Parameters	Architecture
CNN Model 1	Xavier Uniform Initialization Number Epochs: 1000 K-Folds: 4 Splits Batch Size: 500 Adam Optimizer Learning Rate: 0.005 Criterion (Loss Function): Cross Entropy Loss	Input: N * 1 * 22 * 1000 Convolutional Layer: stride: (1,3), kernel: (1, 50), filter number: 50 Batch Normalization: momentum 0.1 Exponential Linear Units (ELU) with activation map N*22*317*50 Convolutional Layer: stride: (1, 1), kernel: (22, 1), filter number: 50 Batch Normalization: momentum 0.1 Exponential Linear Units (ELU) with activation map N*1*317*50 Average Pooling: stride: (1, 15), kernel: (1, 75) Dropout (p = 0.3) Fully Connected Layer: (50 * 17, 4) Softmax Output: N * 4
CNN Model 2	Xavier Uniform Initialization Number Epochs: 1000 K-Folds: 4 Splits Batch Size: 500 Adam Optimizer Learning Rate: 0.005 Criterion (Loss Function): Cross Entropy Loss	Input: N * 1 * 22 * 1000 Convolutional Layer: stride: (1, 2), kernel: (1, 40), filter number: 40 Batch Normalization: momentum 0.1 Exponential Linear Units (ELU) with activation map N*22*481*40 Convolutional Layer: stride: (1, 1), kernel: (22, 1), filter number: 40 Batch Normalization: momentum 0.1 Exponential Linear Units (ELU) with activation map N*1*481*40 Average Pooling: stride: (1, 15), kernel: (1, 75) Dropout (p = 0.3) Fully Connected Layer: (40 * 28, 4) Softmax Output: N * 4
CNN Model 3	Xavier Uniform Initialization Number Epochs: 1000 K-Folds: 4 Splits Batch Size: 500 Adam Optimizer Learning Rate: 0.005 Criterion (Loss Function): Cross Entropy Loss	Input: N * 1 * 22 * 1000 Convolutional Layer: stride: (1, 1), kernel: (22, 1), filter number: 40 Batch Normalization: momentum 0.1 Exponential Linear Units (ELU) with activation map N*22*481*40 Convolutional Layer: stride: (1, 2), kernel: (1, 40), filter number: 40 Batch Normalization: momentum 0.1 Exponential Linear Units (ELU) with activation map N*1*481*40 Average Pooling: stride: (1, 15), kernel: (1, 75) Dropout (p = 0.3) Fully Connected Layer: (40 * 28, 4) Softmax; Output: N * 4
CNN Model 4	Same as CNN Model 2 but Number Epochs is 50 for each model, which has 9 models for each subject	Same as CNN model 2
CNN Model 5	Same as CNN Model 2	Same as CNN model 2, but Fully Connected Layer becomes (40 * 28, 9), and the softmax outputs (N * 9)
RNN Model 1	Number Epochs: 200 K-Folds: 5 Splits Batch Size: 50 Adam Optimizer Learning Rate: 0.001 Criterion (Loss Function): Categorical Cross Entropy Loss	Input: N * 1 * 22 * 1000 Bidirectional LSTM Layer: a visible layer with (22, 1000) input, and a hidden layer with 128 LSTM neurons, with return sequence, 0.01 l2 kernel, bias, and recurrent regularizer. Batch Normalization: momentum 0.99 Dropout (p = 0.5) Flatten layer Dense with ELU Activation layer(32) Dropout (p = 0.5) Output: N * 4, using Adam optimizer with 0.002 kernel regularizer
RNN Model 2	Same as RNN Model 1	Same as RNN Model 1, except for the input with size (N * 1 * 199 * 220)
RNN Model 3	Number Epochs: 200 K-Folds: 5 Splits Batch Size: 50 Adam Optimizer Learning Rate: 0.001 Criterion (Loss Function): Categorical Cross Entropy Loss	Input: N * 1 * 22 * 1000 Bidirectional LSTM Layer: a visible layer with (22, 1000) input, and a hidden layer with 128 LSTM neurons, with return sequence, 0.01 l2 kernel, bias, and recurrent regularizer. Batch Normalization: momentum 0.99 Dropout (p = 0.5) Bidirectional LSTM Layer: a visible layer with (22, 1000) input, and a hidden layer with 64 LSTM neurons, with return sequence, 0.01 l2 kernel, bias, and recurrent regularizer. Flatten layer Dense with ELU Activation layer(32) Dropout (p = 0.5) Output: N * 4, using Adam optimizer with 0.002 kernel regularizer
RNN Model 4	Same as RNN Model 3	Same as RNN Model 3, except for the input with size (N * 1 * 199 * 220)
CRNN Model 1	Number Epochs: 200	Part 1: Input: N * 1 * 199 * 220

	K-Folds: 5 Splits Batch Size: 50 Adam Optimizer Learning Rate: 0.005 Criterion (Loss Function): Categorical Cross Entropy Loss	Convolutional Layer: stride: (1, 1), kernel: (1, 10), filter number: 16 Exponential Linear Units (ELU) with activation map N*22*211*16 Batch Normalization: momentum 0.99 Convolutional Layer: stride: (1, 1), kernel: (21, 1), filter number: 32 Exponential Linear Units (ELU) with activation map N*1*211*16 Batch Normalization: momentum 0.99 MaxPool: (1, 4) Convolutional Layer: stride: (1, 1), kernel: (1, 10), filter number: 64 Exponential Linear Units (ELU) with activation map N*10*202*64 Batch Normalization: momentum 0.99 MaxPool: (1, 4) --- Part 2: Dropout (p = 0.4) Permute(2, 3, 1) TimeDistributed Flatten Layer Bidirectional LSTM Layer: 128 LSTM neurons with return sequence Bidirectional LSTM Layer: 64 LSTM neurons with return sequence Bidirectional LSTM Layer: 32 LSTM neurons Dropout (p = 0.4) Output: N * 4, using Adam optimizer with 0.01 kernel regularizer
CRNN Model 2	Same as CRNN Model 1	Part 1 from CRNN Model 1 Convolutional Layer: stride: (1, 1), kernel: (1, 10), filter number: 128 Exponential Linear Units (ELU) Batch Normalization: momentum 0.99 MaxPool: (1, 4) Part 2 from CRNN Model 1
CRNN Model 3	Same as CRNN Model 1	Part 1 from CRNN Model 1 Convolutional Layer: stride: (1, 1), kernel: (1, 10), filter number: 256 Exponential Linear Units (ELU) with activation map N*1*193*256 Batch Normalization: momentum 0.99 Part 2 from CRNN Model 1, but add 'Bidirectional LSTM Layer: 128 LSTM neurons with return sequence' after 'TimeDistributed Flatten Layer'

Models	Train/Test Accuracy (%)	Final Training Loss
CNN Model 1	1.0/.62	.77
CNN Model 2	1.0/.64	.79
CNN Model 3	1.0/.69	.77
CNN Model 4.1	1.0/.44 (Average)	.75
CNN Model 4.2	1.0/.63 (Average)	.78
CNN Model 5	1.0/.99	1.37
RNN Model 1	.91/.32	2.16
RNN Model 2	.84/.40	1.05
RNN Model 3	.96/.30	1.64
RNN Model 4	.96/.42	1.05
CRNN Model 1	1.0/.57	.08
CRNN Model 2	1.0/.63	.14
CRNN Model 3	.97/.41	.24

For CNN Model 4, all of the various test accuracies for each individual subject network can be found in the report.