ECE C147/C247 Project – CNN and RNN for EEG data

Viacheslav Inderiakin (505430686),
Venkata Sai Sriram Sonti (904599241),
Atharv Sakhala (105191937),
Danny Depe (004146367)

March 2020

Abstract

Electroencephalography (EEG) is a method of monitoring brain activity by recording electric potentials in various points of human scalp. Compared to other human bio-signals, EEG is presumed to be more noisy and have worse resolution, which obstructs its classification. Machine Learning algorithms, particularly Neural Networks (NN), allow us to overcome this limitation. In this project, we construct 4 NN models and optimize them on the task of classifying 22-channels EEG signals [1]. We report, that regardless of architecture, models trained on 1 subject do not generalize well. We investigate dependency between signal duration and classification accuracy, and show that the best accuracy can be achieved by combining CNN and RNN architectures.

1. Introduction

Over the course of this project, our group has designed and optimized 4 different networks: Shallow CNN, Deep CNN, RNN and CRNN, and compared their accuracy on the task of personalized/general EEG classification. Out of these networks, Shallow CNN and Deep CNN are commonly used solutions for classification, and their performance is used as a baseline to evaluate RNN and CRNN.

1.1. Shallow CNN

Shallow CNN architecture, proposed in [3], consists of a single convolution-average pooling block, followed by a dense layer. It is worth mentioning that convolution across time and space are performed in separate layers to separate time and space features, which helps to regularize the model. Shallow CNN uses square and log activations to produce FBCSP-like features. The architecture of this network can be seen in figure 5 (note that square activation is applied to the output of the second convolution layer and log activation is applied to the output of maxpool layer).

1.2. Deep CNN

Deep CNN architecture, also proposed in [3], consists of a sequence of 4 convolution-max pooling blocks, followed by a dense layer. Similarly to Shallow CNN, it performs convolutions across time and space separately. However, in the case of Deep CNN it does not help significantly, and it remains prone to over-fitting. To fix this, we added dropout layers to blocks 2-4. To further improve accuracy, we also performed batch normalization after each convolution. Deep CNN uses ELU activations, which prevent dying out of neurons. Architecture of this network can be seen in figure 6 (note that batchnorm / dropout / activation / rehsape / permute layers are not shown on the graph).

1.3. RNN

The first RNN architecture was a simple 2 LSTM layer RNN. This mirrored what we had done previously in the class with first testing 2-layer fully-connected nets. This architecture also included a fully-connected layer with an ELU activation. This kept the architecture similar to our CNNs and more comparable.

Next, we increased the number of LSTM layers to 3, as well as added another fully-connected layer to gauge the impact of deeper RNNs. However, training showed that RNNs did not significantly benefit from more layers. Howver, the accuracy can be increased by applying dropout for LSTM and regularizing its weights with L2 norm. Optimal architecture is shown in figure 7 (note that dropout/activation/reshape layers are not shown on the graph).

1.4. CRNN

Several more recent works combine CNN and RNN architecture to classify EEG signals, for instance [2]. Unfortunately, its architecture can not be directly applied to our

task, because this work deals with samples 3 times longer than those we have. This is why we had to reduce the number of convolution-max pooling blocks to 3, decrease the kernel size to 4 and use fewer LSTM layers (3). We also appended the last LSTM block with a dense ELU layer followed by dropout to make features more distinguishable, which had a great impact on accuracy. Optimized architecture can be seen in figure 8 (note that dropout / activation / reshape / permute layers are not shown on the graph).

2. Results

2.1. Shallow CNN

Accuracy of Shallow CNN model for single subject and for the whole dataset is shown in tables 1 and 3. It can be seen that, while testing on a specific subject, the model achieves higher accuracy if trained on just that subject as opposed to training on the whole dataset, but this result does not generalize well to testing on the entire dataset. Dependency between accuracy and sample duration is shown in Figure 1.

2.2. Deep CNN

Accuracy of Deep CNN model for single subject and for the whole dataset is shown in tables 2 and 3. It can be seen that this models is approximately as precise as shallow CNN. However, accuracy of classification for single person is much lower for Deep CNN. As in case of Shallow CNN, subject-specific model does not generalize well. Dependency between accuracy and sample duration is shown in Figure 2.

2.3. RNN

While initially RNN performance was poor, by applying regularization and modifications to the initial architecture, we were able to improve its performance. Most significantly accuracy was affected by dataset augmentation - after we normalized inputs (using only training set mean and variation), we were able to match RNN and baseline (Deep/Shallow CNN) performance (see table 3). Strong dependency on sample duration was observed for RNN similarly to CNNs and can be seen in figure 3. The top 5 hyperparameter optimized configurations are shown in figure 4.

2.4. CRNN

Accuracy of CRNN model is shown in table 3. Dependency between sample duration and accuracy can be seen in figure 4. Out of all our models, this achieved the best accuracy. The top 5 hyperparameter optimized configurations are shown in figure 4

3. Discussion

In this section, we will review our project as a whole and discuss insights we had gained from the varying performances of our models.

3.1. Choice of hyperparameters

For Shallow CNN, the only hyperparameter we optimized was the length of the input. We also tried to normalize input data, but this did not affected performance.

For Deep CNN, we optimized the length of the input and dropout.

For RNN, we optimized the length of the input, dropout, regularization and number of units in LSTM layers. However, the most important effect had input normalization.

For CRNN, we optimized parameters of convolution/max pooling blocks and LSTM layers separately. For RNN, we varied number of units in the last layer, regularization and dropout. For CNN part, we varied kernel size and dropout. We also optimized number of units in the last dense layer and number of timestamps per sample.

For all of the networks, Adam optimizer was used.

3.2. Subject-specific vs general performance

As seen in table 1, Shallow CNN can achieve better results being trained for single subject specifically. This is most probably due to the nature of EEG signal, which characteristics strongly depend on one's physiology. Being optimized for one person, Shallow CNN memorizes person-specific attributes of the signal and can predict class better. This hypothesis is supported by the fact that high accuracy does not generalize on the whole test set.

Unlike Shallow CNN model, Deep CNN achieves poor results being trained for one subject (see table 2). This is because per-person number of training examples is low and number of parameters of Deep CNN is high which creates conditions for overfit. Similarly to Shallow model, per-subject accuracies of Deep model do not generalize well.

3.3. Comparison between networks

Table 3 shows that baseline models (Shallow/Deep CNN) achieve almost the same test accuracy (61-62%) on the whole test dataset. Recurrent neural network are well suited for classifying time sequences, so we expected RNN model to perform better than baseline. However, this proved wrong: RNN performance was about 56%. We attribute it to the large number of timestamps in a sample: it seems that state of LSTM can not "memorize" all information presented in an example. In contrast, convolution layers allow CRNN to "pre-fetch" important features from the series of timestamps and to greatly reduce their number, which makes following LSTM layers perform more efficiently. This results in almost 5% gain in test accuracy over the CNNs.

Out of the 4 models, Simple RNN is the longest to train. The reason again is the high number of timestamps per example. Other three models (Shallow/Deep CNNs and CRNN) take approximately same time per epoch.

3.4. Performance with different time windows

Performance of all of the models strongly depended on the time window we chose. Graphs of Validation accuracy vs number of time stamps are given below. As you can see from figures 1, 2 and 3, for Shallow CNN, Deep CNN and RNN optimal length of the training example was less than the maximal length of 1000 timestamps. This result seems counter-intuitive because the more information a model has the more features it has to distinguish and the better classify. We attribute this result to poor time-domain capacity of these models: as we discussed below, RNN model can not efficiently store information from all 1000 examples in its state, and the capacity of CNNs is restricted by the receptive field each neuron of the last convolution layer. In contrast, CRNN uses both convolutional-max pooling blocks and LSTM layers, which allows it to have higher time-domain capacity. This hypothesis is supported by the fact that for CRNN optimal length of a training example is the maximal possible length.

References

[1] Bci competition iv. *BCI Competition IV*. http://www.bbci.de/competition/iv/.

[2] E. Bresch. Recurrent deep neural networks for real-time sleep stage classification from single channel eeg. 2018.

[3] R. T. Schirrmeister. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human eeg. 2018.

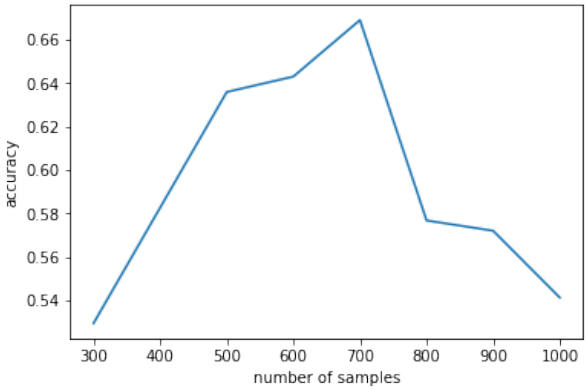

Figure 1. Shallow CNN Architecture: Number of samples refers to the number of timestamps used in the input data

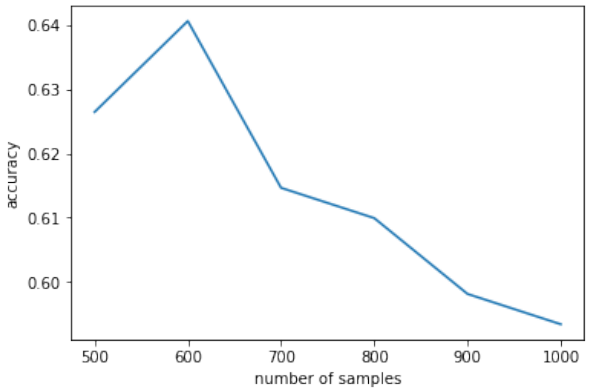

Figure 2. Deep CNN Architecture: Number of samples refers to the number of timestamps used in the input data

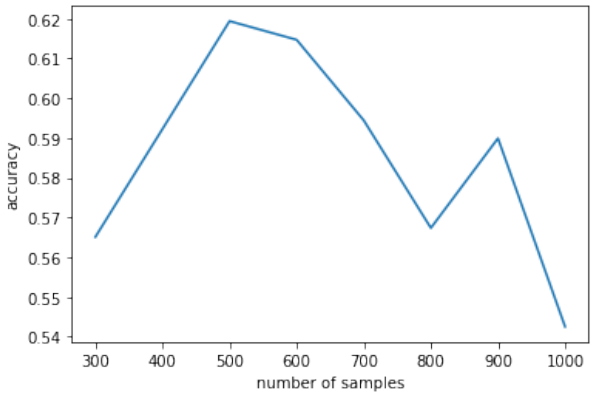

Figure 3. RNN architecture: Number of samples refers to the number of timestamps used in the input data

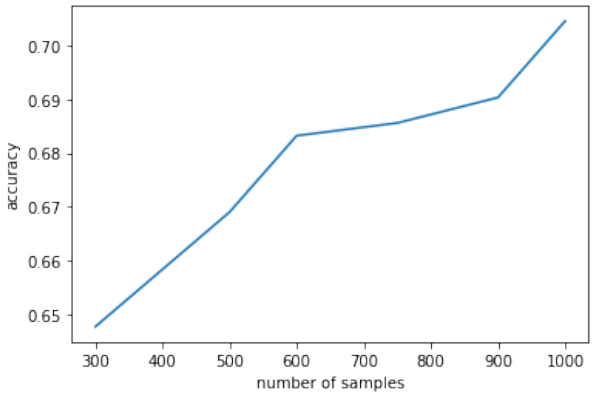

Figure 4. CRNN architecture: Number of samples refers to the number of timestamps used in the input data

Person Number	SinglePersonAcc	EntireDatasetAcc
0	0.5800	0.3370
1	0.4200	0.3454
2	0.8400	0.3747
3	0.4200	0.3409

Table 1. Shallow CNN trained on different subjects. Accuracies for test on single person's dataset and for test on entire dataset

Person Number	SinglePersonAcc	EntireDatasetAcc
0	0.5200	0.3138
1	0.3600	0.3521
2	0.4400	0.3725
3	0.4200	0.3476

Table 2. Deep CNN trained on different subjects. Accuracies for test on single person's dataset and for test on entire dataset

Model	TrainAcc	ValAcc	TestAcc
ShallowCNN	0.8853	0.6609	0.6208
DeepCNN	0.8300	0.6407	0.6140
RNN	0.7482	0.6194	0.5621
CRNN	0.7701	0.7045	0.6704

Table 3. Optimal Training, Validation, and Testing accuracies for our different models. We tested with the configurations with the best validation accuracies

Time Stamps	Regula -rizer	Hidden Units	Dropout Rate	Val Acc
500	0.005	128	0.3	0.6194
500	0.001	128	0.3	0.6147
600	0.005	128	0.3	0.6147
600	0.005	128	0.2	0.6028
600	0.001	128	0.3	0.6028

Table 4. RNN Validation accuracies with various top configurations. Time stamps refers to the time window or number of samples; regularizer is the factor for L2 regularization; hidden units is the number of hidden units in the LSTM layers, dropout rate is the factor for every dropout layer

Time Stamps	Last LSTM	Last Hidden	Dropout Rate	Val Acc
1000	64	1000	0.3	0.7045
1000	128	1000	0.2	0.6927
750	64	500	0.3	0.6856
750	64	1000	0.2	0.6832
750	128	500	0.3	0.6832

Table 5. CRNN Validation accuracies with various top configurations. Time stamps refers to the time window or number of samples; last LSTM is the size of the last LSTM layer; last hidden is the number of hidden units in the last layer, dropout rate is the factor for the last dropout layer

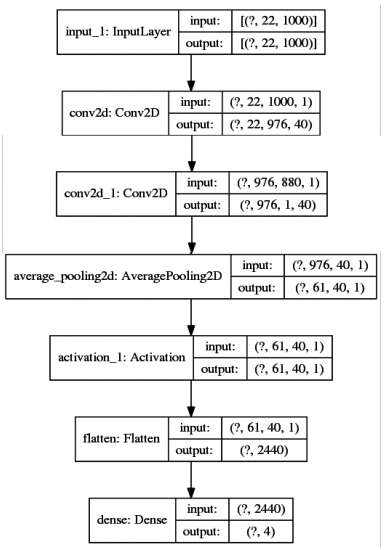

Figure 5. Shallow CNN Architecture: The order of filters used is convolution of 1x25 stride 1, convolution of 1x880 stride 1, and averagepool of 75x1 stride (15, 1)

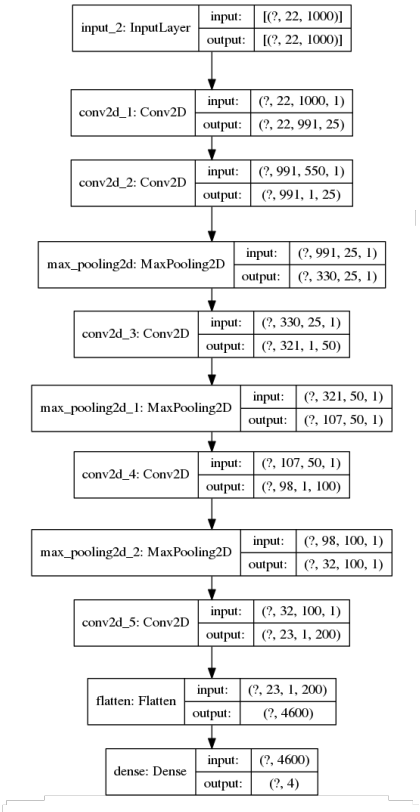

Figure 6. Deep CNN Architecture: The order of filters used is convolution of 1x10 stride 1, convolution of 1x550 stride 1, maxpool of 3x1 stride (3,1), convolution of 10x25 stride 1, maxpool of 3x1 stride (3, 1), convolution of 10x50 stride 1, maxpool of 3x1 stride (3,1), convolution of 10x100 stride 1, maxpool of 3x1 stride (3,1)

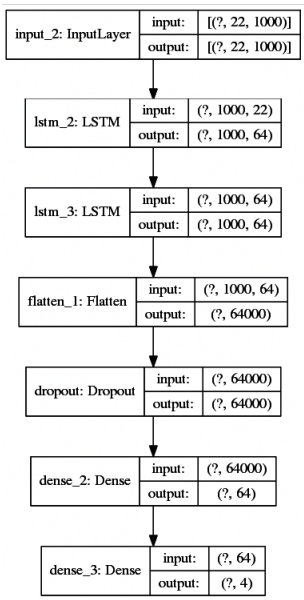

Figure 7. RNN architecture

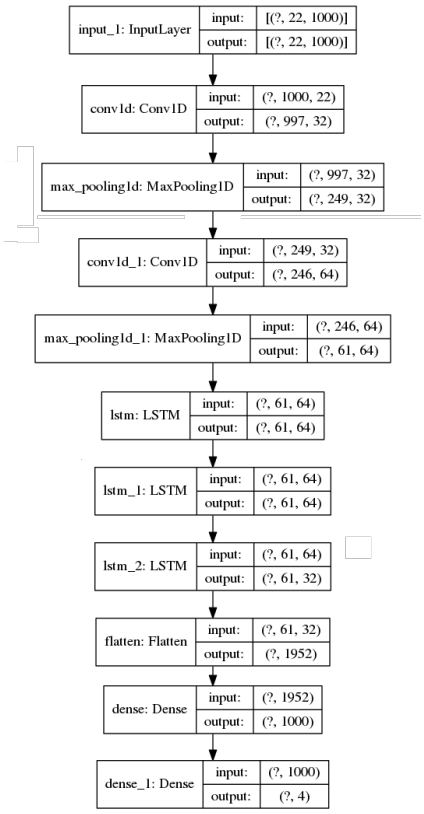

Figure 8. CRNN architecture: A filter size of 4 was used for all the convolution and maxpool layers