Classification of Motor EEG Signals using Deep Learning

Vaishnavi Manthena UID: 306075648
Dhakshina Ilango UID: 306299832
Kajal Sharma UID: 906297199
Vani Agarawal UID: 706302253

A. Abstract

In this project, we considered the EEG dataset and applied 4 different models on it, namely- CNN, CNN+LSTM, LSTM and vanilla RNN. Our aim is to compare how the results fluctuate in case of application of the different models. Out of all the models, CNN outperformed with an accuracy of 70.65% followed by CNN+LSTM with an accuracy of 69.99%, then, RNN with an accuracy around 53.04% and the least performing model was LSTM.

B. Introduction

In this report, we aim to decode brain signals from the open-source BCI Competition data using deep learning techniques.

This paper aims to answer 3 basic questions : (Q1) Test accuracy on subject 1 while training on subject 1 only, (Q2) Test accuracy on subject 1 while training on all data points and (Q3) Evaluate the classification accuracy as a function of time.

B.1. Motivation behind Architectures

The Convolutional Neural Networks (CNNs) model has made significant strides in the field of Computer Vision (He et al., 2015; Krizhevsky et al., 2012). It can extract low-level features from raw data by using convolutional operations and with the increase in the number of layers it can capture complex high-level features. Thus CNNs have been used extensively in the field of neuroscience for brain-signal decoding (Antoniades et al., 2016; Bashivan et al., 2016; Cecotti and Graser, 2011; Hajinoroozi et al., 2016; Lawhern et al., 2016;). CNNs along with other machine learning techniques, especially recurrent neural networks (RNNs) have also done well in this domain (Li and Wu, 2015; Sainath et al., 2015b; Saket al., 2015). In this paper, we experiment with various deep learning architectures RNN, Long Short-Term Memory (LSTM), CNN and CNN-LSTM models. We start with simpler models and evaluate their performance on the EEG signals present in the BCI dataset. Finally, we combine CNN and LSTM architectures to evaluate the performance of a more complex model on the dataset. We resorted to simpler architectures due to small datasets as well as computational constraints such as GPU and memory.

C. Results

C.1. Optimizing Models

Designing a CNN for classifying EEG data first involved using a baseline model, as provided in discussion 6, that seemed like a simplified version of the CNN architecture in the paper by Schirrmeister et al. that also classifies EEG data. This model uses 4 convolutional blocks (including convolutional, pooling, batch normalization, and dropout layers) followed by a dense layer fed into softmax activation. The model was optimized to reduce overfitting by using 3 convolutional blocks instead of 4. Smaller Kernel sizes were used for the initial layers. To avoid underfitting we increased the number of filters in the 3 convolutional blocks. Intuition for the last 2 updates comes from observations in the ZFNet case study where smaller filters with smaller strides in earlier layers and more filters in deeper layers seemed to help CNN performance. The dropout rate was increased to try and improve generalization. Figures 1 and 2 illustrate training with this new architecture using all training and validation data. The testing accuracy on all data now crosses 70%.

For the RNN model, we used 4 layers of SimpleRNN in the architecture and along with it, we leveraged gradient clipping and Prascanu's regularisation. The problem is to optimise the model to increase its accuracy. To do so, we added a regularisation term and took gradient clipping to tackle the issue of vanishing and exploding gradients. The main challenge we had during this implementation was the restriction on GPU usage. The model is overfitting while training on subject 1 data and testing on subject 1 only plus the technique of training on all subjects and testing on subject 1. For the model, we used an L1 kernel regulariser with a value 0.01, dropout of 0.25 and gradient clipping value of 0.2. The final accuracy while we trained on all data points and tested on all data points was around 53%. This can be seen in Figures 3 and 4. It can further be increased by hyperparameter tuning essentially because we got a chance to play with only a few numbers like 0.01 for regularisation, 0.25 and 0.5 for dropout, 32 as the batch size, 0.2 as the gradient clipping value. All these values when considered different can lead to different accuracy results which is a future aspect of testing.

For the LSTM model, initially, a naive model was implemented. To improve the performance of the model additional layers were added. The intuition behind increasing the number of layers of the model was so that the model could capture more patterns and high-level features accurately. The optimized LSTM model consists of 4 layers excluding the output layer with 512, 256, 128 and 64 units respectively. Since the model overfitted on the train data

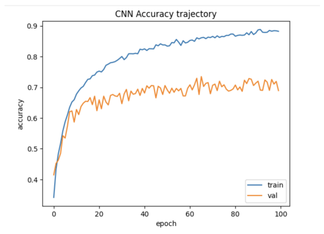

Figure 1. Optimized CNN accuracy history

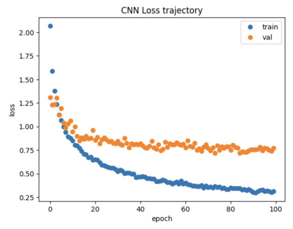

Figure 2. Optimized CNN loss history

different regularization techniques were experimented with such as Batch Normalization, L1 and L2 kernel regularizer, and dropout and the learning rate of the optimizer was adjusted as well. The dropout rate was an empirical choice and was optimized to tackle the overfitting problem. The value of dropout that gave the best accuracy was 0.01. The learning rate was also adjusted from 0.001 to 0.01 which resulted in smoother convergence of the model. The accuracy of the model was 27.08% (This is under the setting that the entire dataset is split into train, validation and test splits). This is explained in Figures 5 and 6.

For the CNN+LSTM hybrid model, initially, a very simple model was implemented with 1 convolutional block layer and 1 LSTM layer. However, this model performed very poorly on the training and validation set. To increase the complexity of the model, 4 convolutions layer blocks with 2 LSTM layers were added, however, this caused the model to highly overfit to the training data. To optimize the model and reduce overfitting, the model was reduced to 3 convolutional layer blocks. Each block consisted of a Conv2D layer followed by Batch Normalization, MaxPooling2D, and Dropout Layers with 16, 32, and 64 filters respectively. To further reduce overfitting, L2 regularization with a coefficient of 0.001 was added and dropout rates were reduced from 0.7 to 0.5 to produce the best accuracy. This can be observed in Figures 7 and 8.

Refer to the 'Algorithm Performance and Model Architecture' section for the exact final architectures and results of each optimized model.

C.2. Model Evaluation as a function of EEG time

To better understand the given data, we also trained our optimized models on different time crops to understand how

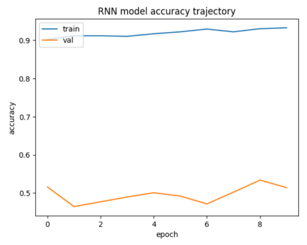

Figure 3. Optimized RNN accuracy history

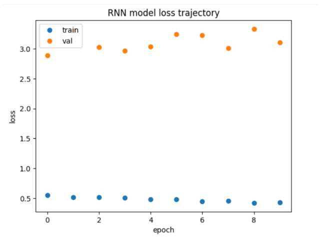

Figure 4. Optimized RNN loss history

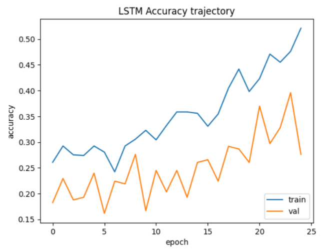

Figure 5. Optimized LSTM accuracy history

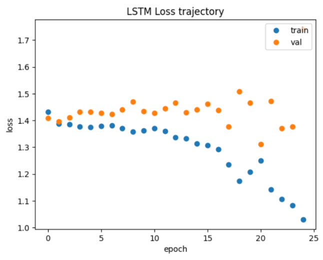

Figure 6. Optimized LSTM loss history

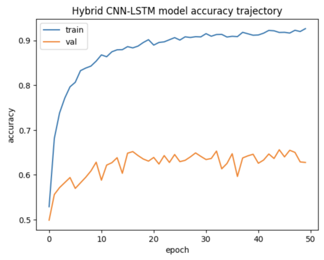

Figure 7. Optimized CNN-LSTM accuracy history

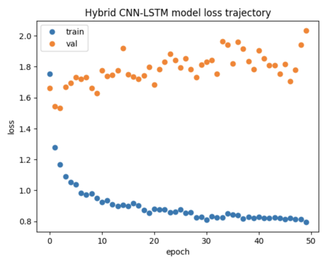

Figure 8. Optimized CNN-LSTM loss history

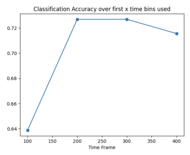

Figure 9. Optimized CNN test accuracy as a function of time frame

much time (out of the 1000 time bins in the original data) is really significant. Since the processed data has 400 time bins, we train and evaluate our model on each of the following time frames: 0-100, 0-200, 0-300, and 0-400.

Using the CNN model gave the result in Figure 9. The test time classification accuracy is pretty good even after just considering the first 200 time bins in the processed data. These 200 time bins roughly correspond to the first 400 out of 1000 time bins in the original data. This is consistent with our finding in data analysis where the electrode signals seem to become noise after first 500 out of 1000 time bins. For the last three time frames the final test accuracies cross 70% and also some of the epochs cross 70% validation accuracy. Although the model trained on the time frame 0-100 is not as good, it still gives a reasonable validation accuracy of a little over 60% during the final epochs and a final test accuracy of 64%. CNN-LSTM model performed similar results as the CNN model as the time bins increased as seen in Figure 7.

For the LSTM model with the increase in time bins, the accuracy decreased from 25% to below 22% which can be observed in Figure 10. The increase in time bins could be attributed to an increase in the dimensionality of the input data which exacerbates the issue of overfitting. Therefore since LSTMs are used for temporal analysis as the temporal context expands it might be difficult for the model to capture meaningful patterns. This potentially could cause overfitting which results in high train accuracy and low validation accuracy of the model.

Using the RNN model gave the result in figure 12. The accuracy for the first 200 bins was quite low (close to 32%). The increase in time bins led to a better accuracy (approx 45%) till 300 bins and it dropped again afterwards to 30%.

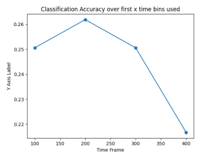

Figure 10. Optimized LSTM test accuracy as a function of time frame

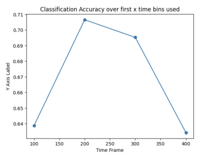

Figure 11. Optimized CNN-LSTM test accuracy as a function of time frame

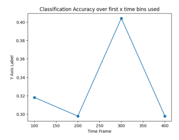

Figure 12. Optimized RNN test accuracy as a function of time frame

D. Discussion

As can be seen through the table of final performance comparisons, for CNN, the subject 1 test accuracy turns out to be higher when the model is trained on all data versus just subject 1 data. This is potentially because multitasking regularization (trying to make the model good at classifying for any subject) helps the model generalize better for the particular task (classifying subject 1's data).

For RNN and LSTM the subject 1 test accuracy remains the same when the model is trained on a subset of subject 1 data as well as when it is trained on a subset of all the data. This could be because the subset of all data used for training may contain a sufficient representation of subject 1 data along with data from other subjects.

Testing accuracy for CNN model and CNN-LSTM model is the highest when we train on all data and test on all data. These two models also give best performances in the other test accuracies. This is potentially because although each input datapoint involves a time series, the datapoints are not inter-related through time. Hence, considering each data as an independent gray-scale images (CNN) and/or using hybrid models yields better results than simple RNN's.

References

[1] R. Schirrmeister et al., "Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG,"

IEEE, 2017.

[2] D. Balderas, P. Ponce, and A. Molina, ”Convolutional long short term memory deep neural networks for image sequence prediction,” Expert Systems with Applications, vol. 122, pp. 152-162, May 2019.

[3] T. Najafi, R. Jaafar, R. Remli, and W.A. Wan Zaidi, ”A Classification Model of EEG Signals Based on RNN-LSTM for Diagnosing Focal and Generalized Epilepsy,” Sensors (Basel), vol. 22, no. 19, pp. 7269, Sep. 2022.

[4] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. Salakhutdinov, and Y. Bengio. Architectural complexity measures of recurrent neural networks, 2016.

A. Algorithm Performance and Model Architecture

Model	Train on all and Test on all	Train on all and Test on Subject 1	Train on Subject 1 and Test on Subject 1
CNN	70.65%	68%	56%
RNN	51.99%	51.99%	53.04%
LSTM	27.08%	30.00%	30.00%
CNN + LSTM	67.72%	69.99%	68%

Table 1. Comparison of Test Accuracies

Model	Complexity (Trainable parameters)	Time to Train	GPU
CNN	567,954	2.3 minutes	with GPU
RNN	400,304	-	with GPU
LSTM	2,129,924	-	with GPU
CNN + LSTM	459,796	-	with GPU

Table 2. Comparison of Deep Learning Models

Data Processing

We first split the train data into train and validation data so that we have separate train, validation, and test sets. Then we do preprocessing:
Train and Validation Data Processing & Augmentation

- Adjust labels so that they are in range 0 to 4 instead of 769 to 772.
- Remove that last 200 time bins.
- Apply max pool low pass filter across time dimension and then apply an average low pass filter across time dimension and add noise.
- Augment average filtered plus noise added data to max pool filtered data.
- Apply subsampling in time dimension and add noise. Augment this to above to form final training or validation data.

Note about noise: This is gaussian noise with mean 0 and standard deviation 0.05.

Test Data Processing

- Adjust labels so that they are in range 0 to 4 instead of 769 to 772 and remove that last 200 time bins.
- Apply max pool low pass filter across time dimension.

The shape of each EEG trial finally turns out to be 22*400. We have used same data pre-processing steps for all 4 models.

Convolutional Neural Network (CNN) architecture

The architecture involves 3 convolutional blocks followed by a fully connected layer whose output (of 4 units) is fed into softmax activation. For this architecture the labels are formatted into a categorical one-hot encoding and the data is shaped in the dimension: (400, 1, 22). Description of a convolutional block:

- 2D convolution layer:
  - 'elu' activation function
  - Stride of 1
  - Padding setting such that output_shape = input_shape
  - First convolutional block uses 50 kernels of size of (3,3) here.
  - Second convolutional block uses 100 kernels of size of (3,3) here.
  - Third convolutional block uses 200 kernels of size of (5,5) here.
- 2D max pooling layer where filter size and stride is '(3, 1)'. Padding setting such that: output_shape = floor $\left(\frac{\text{input\_shape}-1}{\text{stride}}\right)+1$.
- Batch Normalization
- Dropout with rate 0.6

Categorical cross entropy loss function, adam optimizer with learning rate of $1e-3$, 100 epochs, and 64 batch size were used for training.

Long Short Term Memory (LSTM)

The architecture consists of 4 LSTM layers stacked on top of each other with varying numbers of units (512, 256, 128 and 64) respectively and dropout regularization of 0.01 is applied to prevent overfitting. The output layer consists of a fully connected layer whose output (of 4 units) is fed into the softmax activation function. The labels are formatted into a categorical one-hot encoding and the data is shaped in the dimensions: (400, 22). Categorical cross entropy function loss function, Adam optimizer with learning rate 0.01, 25 epochs and 32 batch size was used for training.

CNN + LSTM Hybrid Architecture

The architecture consists of 3 convolutional blocks each consisting of a Conv2D layers followed by BatchNormalization, MaxPooling2D, and Dropout. Each CNN layer uses ELU activation and L2 regularization with a coefficient of 0.001. The Dropout layers have a dropout rate of 0.25 to prevent overfitting. The CNN blocks are then flattened and fed into a fully connected Dense layer with 128 units and ELU activation. The output is then reshaped to a 3D tensor for input to the LSTM layer. The LSTM layer consists of 64 units with a dropout rate of 0.5. After the LSTM layer, there is a Flatten layer followed by a Dense layer of 4 unts with softmax activation. This is the output layer for the classification.

RNN Architecture

The architecture consists of 4 Simple RNN layers each consisting of a dropout of 0.25, l2-kernel regularisation as 0.01, recurrent initializer as glorot-uniform. The dropout was added to prevent overfitting, which did help a bit. The RNN layers are then flattened and fed into a fully connected Dense layer with different number of units(25, 50, 100, and 200) and ELU activation. The labels are formatted into a categorical one-hot encoding and the data is shaped in the dimensions: (400, 22). Categorical cross entropy function loss function, Adam optimizer with learning rate 1e-3, 32 batch size and different number of epoch were used for 3 different cases (i.e., train on subject 1 only and test on 1, train on all v/s test on 1, and train on all v/s test on all) were used for training. For 1st case, we used 20 epochs for training and 10 epochs for the last 2 cases because of GPU shortage.