Motor Task Classification Using EEG Data

Srishti Majumdar
Computer Science
University of California, Los Angeles
srishtimajumdar@g.ucla.edu

Abstract

In this project, the classification of EEG data into 4 motor tasks is explored. The data provided is originally provided by the Brain-Computer Interface Competition [1] and has been processed to include only complete trials. As a part of the project, I describe the various shallow convolutional (CNNs) and recurrent networks (RNNs) explored across various kinds of data. I also explored the combination of CNN-RNN architectures for the EEG Classification task. The result from the architectures are compared and the maximum accuracy on the test set obtained is 65.6%.

1. Introduction

The aim of this project is to explore and optimise the classification of EEG data into four motor tasks. The design choices for architectures are made based on performance of each base model architecture on the data sets. The datasets, architectural designs and training strategies have been discussed in this section.

1.1. Dataset

The EEG data given consists of 2115 trials. Each trial contains data from 22 electrodes over a 1000 time steps. These 2115 trials belong to 9 subjects numbered from 0 to 8. The target labels are given by 769, 770, 771 and 772 corresponding to movement of left hand, right hand, foot and tongue respectively.

1.1.1 Data Variations

This data used for training and testing models has been described in this sub-section.

Main data: This is the data provided as part of the default project.

Person/Subject wise data: We know that the 2115 trials belong to 9 person/subjects. Hence, I use the person information to divide the main dataset into 9 smaller datasets.

Dataset		Train Shape	Test Shape
Main		1672, 22, 1000	443, 22, 1000
9*Person	Person 0	237, 22, 1000	50, 22, 1000
	Person 1	236, 22, 1000	50, 22, 1000
	Person 2	236, 22, 1000	50, 22, 1000
	Person 3	234, 22, 1000	50, 22, 1000
	Person 4	235, 22, 1000	47, 22, 1000
	Person 5	236, 22, 1000	49, 22, 1000
	Person 6	238, 22, 1000	50, 22, 1000
	Person 7	232, 22, 1000	50, 22, 1000
	Person 8	231, 22, 1000	47, 22, 1000
Cropped		14805, 22, 400	3101, 22, 400

Table 1. Summary of Data Variations Used (Note: Standardised data has the same shape as the original datasets)

Time wise sliced data: I wanted to observe the performance of models over time. Hence, I divided the dataset to reflect trial information over the first 200, 400, 600, 800 time steps and compared the performance to the entire data.

Cropped data: Crop training is often used in neural networks for performance improvement. On observing the higher performance of of data over 400 times steps, I decided to use a sliding window of 400 time steps with stride 100 for the creation of this dataset.

Standardizing data: It has been noticed in many cases that standardizing leads to better generalization in models. Hence, an attempt was made to normalize the data and evaluate model performance.

A summary of dimensions of the data mentioned above is given in Table 1.

1.2. Architectures

In this project, I have explored the implementation of various architectures for EEG Classification. Since I only had access to CPUs, I have explored the use of relatively smaller architectures in detail.

1.2.1 Convolutional Neural Networks

CNNs were the first choice for the given dataset as they would provide a baseline for the future models as well as give a clear idea for various parameters and pre-processing choices.

Shallow Architecture The first model was directly influenced from the paper[2]. The shallow architecture had two convolution layers followed by an average pooling layer and a fully connected layer. The model parameters had to be changed to account for less features in our data as opposed to their model. The choice of hyper parameters had a huge impact on the performance of CNN and a lot of tuning had to be done to get a good test accuracy. I also tried to change the kernel size ranging from 20 to 50. The best performance was achieved at 40. The pool size was average pooling was intentionally kept high to reduce the parameters in the fully connected model. I also found that reducing the pool size reduced the accuracy due to the kind of signals we are dealing with.

Deep Architecture For the deep architecture, the model in the paper did not achieve good accuracy. The model was overfitting due to the deeper layers and even with the cropped data the model did not perform well. I varied the convoluted layers from 4 to 6 and FC layers from 1 to 2.

1.2.2 Recurrent Neural Networks

RNNs are a class of neural networks that have the capability to handle temporal sequences. Given that our data is divided over time steps, RNNs seemed to be a natural choice of architecture.

In this project, I tried two type of RNN architectures: Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU).

LSTM: For LSTMs, I first tried a two layer network with 100 units in the LSTM layer and then 4 units in the dense layer, 1 unit per output label. However, it seemed to be a lot of parameters compared to the performance benefit. Hence, to reduce overfitting, I reduced the number of nodes to 64 which performed worse than the initial model. Hence, to make increase model performance and robustness, I added another LSTM layer with 64 units. This model consistently performed better than the previous two models. I experimented with the use of LSTM for observing data over time, subject wise data as well as standardised and crop data.

GRU: I explored GRU architectures while I was designing combined CNN-RNN architectures. For GRU, I first explored the use of a two layer network with 100 GRU units in first layer and 4 dense units in second. I experimented with dropout, addition of batch normalisation and even early stopping. Only GRU architectures were explored with less rigour as compared to LSTM and combined models due to time and resource constraints.

1.2.3 Combined Model (CNN-RNN)

The next model I tried was a combination of CNN and RNN discussed above. I tried to minimize the number of layers as increasing the layers only resulted in overfitting. I tried the shallow CNN network along with both LSTM and GRU.

1.3. Training Strategies

Tuning of Hyper-parameters: Various hyper parameters were tuned to improve validation accuracy like learning rate, dropout rate, decay and convolution sizes.

Batch Normalization: Batch normalization is often used in training neural networks and increasing performance of the network. I have explored networks with and without batch normalization and usually networks with batch normalization performed better and converged faster.

Dropout: Dropout helps approximates the performance of multiple models and helps regularize the model. Hence, I tried varying the dropout rates for the implemented architectures.

Varying Learning Rates: I tried multiple learning rates to check the performance of models.

Early Stopping: The models tend to overfit the data, especially in RNNs. Hence as a strategy to minimise that I have explored the use of early stopping by monitoring validation loss.

2. Results

I checked model performance over the entire dataset as well over varying time steps. I also tried improving classification accuracy for subject 0, and observing the affects over training across all subjects.

The results of several of the configurations tried are mentioned in the Supplementary Page. The summary of the best results obtained by the architectures is listed in Table 2. On considering **varying time-steps**, all models perform best when trained and tested on the first 400 time steps.

I tried **cropping data** using sliding window of size 400. The performance of models usually decreased as compared

Architecture	Best Testing Performance
CNN	65.6%
RNN (LSTM)	54.5%
RNN (GRU)	50.02%
CNN - RNN	62.5%

Table 2. Summary of Best Performance by Architectures

to performance when they were trained and tested on original data.

Standardisation of data tends to give mixed performance results as compared to original. On average, the performance of standardized data is similar to the original dataset as the data is highly co-related.

In **subject wise training**, I trained for Person 0 and tried modifying the architecture to provide better test accuracy for the same subject. With each modification, classification accuracy across all subjects was also checked. It is noticed, that improving classification for one class doesn't necessarily mean that the classification accuracy over all other subjects. The accuracies obtained are quite unstable and are susceptible to change over multiple runs.This may be due to the fact that per-person data is less.

In **CNN architectures**, the shallow architecture outperformed the deep architecture by 10 to 13% on an average. The deep architecture tends to over fit the training data even with very less epochs.

For the **shallow RNN architectures**, I noticed these tended to over fit the training data as compared to shallow CNN architectures.

The **CNN-RNN** architecture implemented performs better than the RNN architectures tried by a margin of at least 10%. Between LSTM and GRU, the results show that GRU performed better than LSTM.

3. Discussion

In this section, I will discuss some of the results observed.

3.1. Time Taken and Number of Parameters

In general, the time taken to train and test the CNN-RNN architecture was greater than the RNN architecture which was greater than that of training and testing a CNN architecture. This can be attributed to the fact that our shallow CNN architecture has less number of parameters than both RNN and CNN-RNN architectures. Additionally, the RNN architectures have lesser parameters that the CNN-RNN architecture.

3.2. Performance

From our results we see that CNNs performed better than all other architectures. Even when the same CNN was used for combining it with LSTM or GRU, the training improved whereas the testing accuracies reduced. The model overfit even with the usage of dropouts and cropped data.

In general, it can be observed that RNN architectures defined perform worse than CNN architectures. RNNs are more prone to overfitting the training data as compared to CNNs. This may be due to the fact that RNNs have a lot more parameters than CNN.

3.3. Training over Time

For all models, the best performance is noticed when training and testing for 400 time-steps. The increased time-steps may lead to less generalisation of the model.

3.4. Training over Cropped and Standardised Data

The cropped data was created by using a sliding window of size 400 and stride 100. The performance of all models seemed to worsen with this data. A more aggressive cropping strategy may help. However, the EEG data is known to be highly correlated which may cause the reduced performance. This may also be the reason as to why standardising data doesn't lead to guaranteed improvements.

3.5. Improvements

3.5.1 Miscellaneous

The introduction of early stopping in RNN architectures is effective in reducing the training accuracy and hence prevents over-fitting. It also helps reduce time in training the model.

3.5.2 Dropout

The introduction of dropout in models helps in regularization but dropout rates vary with architecture. For example, the LSTM model worked quite well with no dropout and lesser number of units, but the GRU seemed to perform better with some dropout.

3.5.3 Batch normalization

I tried the models with and without batch normalization. The GRU architecture showed an improvement of 3% on average with the inclusion of batch normalization. This makes sense as batch normalization allows for better generalisation of models and leads to more stable models.

3.5.4 Optimizer

In general, 'Adam' is used as an optimizer for all models. RMSProp tends to give similar results and hence has not been reported. The performance of SGD was very slow and needed more epochs, due to which it was quickly disregarded as a possible optimizer.

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] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for eeg decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, Aug 2017.

Methods : Architecture and Performance

Architecture	Data	Best Performance
Conv2D(40, kernel-size=(25, 1)) Conv2D(40, kernel-size=(1, 22)) BatchNormalization Activation : Elu Dropout : 0.5 AveragePooling2D FC Layer Activation : Softmax	Normal	Train: 88.3% Validation: 65.7% Test: 65.6%
	Time wise: Intervals of 200 Best at 400	Train: 85.9% Validation: 64.7% Test: 63.6%
	Cropped:	Train: 54.5% Validation: 39.9% Test: 48.8%
	Standardized	Train: 92.9% Validation: 52.7% Test: 51.9%

Table a. CNN Shallow

Architecture	Data	Best Performance
LSTM(64) LSTM(64) FC Layer	Normal	Train: 99.9% Validate: 51.2% Test: 54.5%
	Time Wise	Train: 99.8% Validate:57.9% Test: 57.7%
	Cropped	Train: 99.8% Validate: 35.1% Test: 33.7%
	Standardized	Train:99.8% Validate: 48.5% Test: 50.3%

Table b. RNN: LSTM

Architecture	Data	Best Performance
Conv2D(32, kernel-size=(10, 1)) Conv2D(32, kernel-size=(1, 22)) BatchNormalization Activation : Elu Dropout : 0.5 MaxPooling2D Conv2D(64) - Others same as prev layer Conv2D(128) - Others same as prev layer Conv2D(256) FC Layer Activation : Softmax	Normal:	Train: 72.4% Validation: 32.8% Test: 34.3%

Table c. CNN Deep

Architecture	Data	Best Performance
GRU(128) GRU(128) BatchNormalization FC Layer	Normal	Train: 94.3% Validate: 41.3% Test: 43.3%
	Time Wise	Train: 68.1% Validate: 52.7% Test: 50.2%
	Cropped	Train: 61.4% Validate: 46.8% Test: 42.8%
	Standardized	Train: 71.9% Validate: 43.7% Test: 34.5%

Table d. RNN: GRU

Architecture	Data	Best Performance
Conv2D(32, kernel-size=(10, 1)) Conv2D(32, kernel-size=(1, 22)) BatchNormalization Activation : Elu Dropout : 0.5 MaxPooling2D LSTM/GRU(128) LSTM/GRU(128) Conv2D(256) FC Layer Activation : Softmax	. Normal: .	Train: 91.3% Validation: 66.7% Test: 62.5%
	. Standardized .	Train: 92.1% Validation: 63.5% Test: 61.6%

Table e. CNN-RNN. Architectures with CNN and then LSTM or GRU were tried and have been reported in the same table.

Note: Only some of the architectures with the results mentioned. All different architectures are present in code. Best time step for 'Time Wise' model was observed at 400 for all our models.

Methods : Architecture and Performance

Subject ID	Test Performance 1	Test Performance 2
0	46.1%	47.9%
1	56.2%	47.1%
2	51.4%	47.3%
3	46.5%	41.3%
4	61.2%	57.8%
5	40.0%	32.1%
6	56.4%	57.2%
7	40.1%	41.5%
8	48.3%	44.9%

Table f: Subject wise accuracy for RNN (LSTM) models over varying hyperparameters. As mentioned in the report the accuracies are not stable across subjects and hence I have mainly focussed on overall architectures.

Model Accuracy and Loss Plots

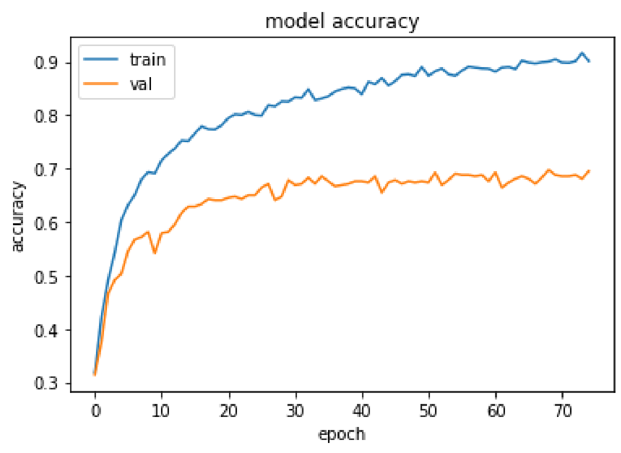

Figure 1 a

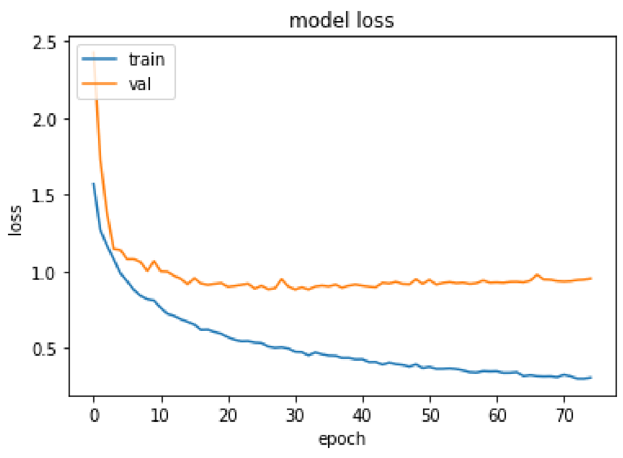

Figure 1 b

Figure 1: a. Model Accuracy b. Model Loss for Shallow CNN architecture

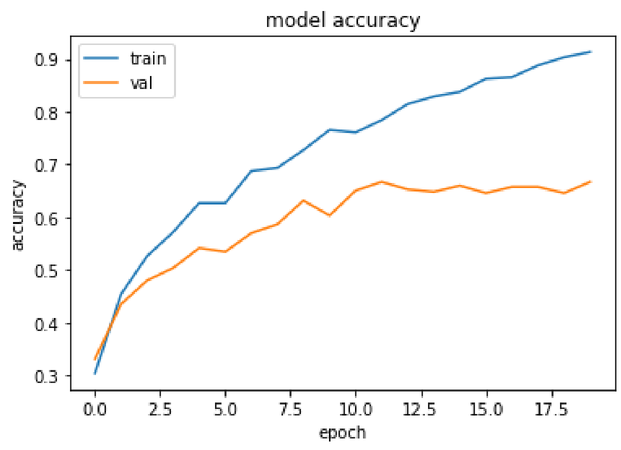

Figure 2 a

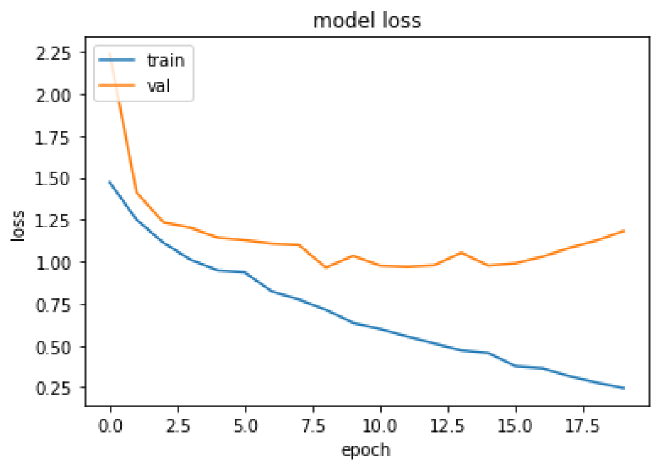

Figure 2 b

Figure 2: a Model Accuracy b. Model Loss for CNN-RNN architecture using GRU.