Analyzing EEG Signals using Neural Networks and Deep Learning

University of California, Los Angeles, USA

Anirudh Krishna, Swagath Babu, Jacob Thomas

anirudhk@g.ucla.edu, swagathb18@g.ucla.edu, jacobthomas28@g.ucla.edu

Abstract

In this project, we compared several deep learning architectures using EEG signals recorded from nine different subjects. In particular, we trained five neural network models using individual and cascaded architectures consisting of Convolutional Neural Networks (CNNs), Long ShortTerm Memory units (LSTMs) and Gated Recurrant units (GRUs). For each of the nine subjects, in addition to all subjects at once, we compared each architecture's classification performance against an independent test set. After hyperparameter optimization and several stages of data preprocessing, we achieved a best decoding accuracy of 74% for a single subject and 54% for models trained on all subjects. In addition, the use of generative models such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) was explored for data augmentation. Stacked Autoencoders were considered for a compressed representation of the dataset. The process helped in fine-tuning the pipeline significantly.

Index Terms: Convolution, Long-short-time-memory, Gated Recurrent Units

1. Introduction

The first lecture for ECE C247 2023 has been instrumental in the selection of this project topic. Machine learning and Artificial Intelligence is playing an important role in revolutionizing the domain of Brain-Machine Interfaces. As part of the research for this project, several articles were studied. One study investigates deep learning methods for end-to-end electroencephalographic (EEG) analysis of movement-related information generated by clinical subjects. Beginning with raw EEG data [2], results from experiments using various machine-learning pipelines were compared by classification accuracy metrics. Before considering any pre-processing technique, the first challenge is to evaluate the quantity of data available, which in this case was less. To offset the potential for overfitting, necessary steps were taken in data regularization and augmentation. Noise reduction was an important aspect considered during pre-processing, and in order to facilitate that, fixed noise reduction techniques such as bandpass filtering and data smoothing using a Hanning window were used. The necessary functions were included in the utils.py Python file. To perform regularization, data clusters were eliminated based on parameters such as null values, insignificance in the classification etc. A choice that was considered in the initial stages were to create data samples using data augmentation techniques and using VAEs, GANs etc. The major work has been done for the classification task in which, after defining the architecture for the models, fine-tuning them and preparing them for the signal decoding tasks was given top priority. The models worked on include CNN, LSTM, CNN + LSTM, GRU and CNN + GRU. These models were used singularly and in combinations of two for evaluation. The analysis and results are discussed in the further sections of the report.

2. Pre-processing and Model Training

2.1. Pre-Processing the Dataset

The EEG dataset contains a small sample size of close to 2000 samples. One of the major reasons behind large data requirements is to avoid overfitting and to expose the model to a larger database. This helps in achieving generality among the data [4]. In addition to directly increasing the training set size via data augmentation, other methods of pre-processing the input EEG data were considered for the task. The pre-processing pipeline used in this study was comprised of four functions, the parameters of which are detailed in [Figure 1]. The band-pass filter and the smoothing function comprise the first two constituents of the pipeline. During training, the last two processes were applied every time a sample was taken from the training set. The DataLoader class defined in Pytorch is extremely important as it allows for the in-place transformation of the data, such that the training data is not permanently altered during training. The first in-place transformation is the addition of Gaussian random noise to the input sample. This applies zero-mean unit-variance Gaussian noise to the entire 22 x 1000 training sample. The noisy sample is then fed into a random erasing method, where a random rectangular mask of the input sample is set to zero.

Stages of the Pre-Processing Pipeline	Filter Parameter Values
Band-Pass Filter	Order of the Filter = 3
Smoothing	Hanning window used using signal.windows.hann command. window size = 5
Gaussian Noise	$\mu = 0$ $\sigma = 1$

Figure 1: *Pre-Processing Pipeline considered*

2.2. Training the Models

After the pre-processing step, it was important to set the model hyperparameters essential for training. The hyperparameters were initially decided based on intuition, however, after model training and cross-validation, the final set of hyperparameters was set based on empirical results and the values are shown in [Figure 2]. An important process during the training was to con-

vert the tensor to numpy array. The issue with this step is that numpy library is a CPU only library and hence, to facilitate the conversion, the tensor object was first detached from GPU using the detach() function and then converted to the numpy array using a command tensor.detach().cpu().numpy().

Parameters	Value
Learning Rate	0.0005
Betas	(0.9,0.999)
Epsilon	$1e^{-08}$
Decay	0.0005
Batch_Size	32
Epochs	100

Figure 2: *Hyperparameters and their values*

3. Model choice and architectures

After the pre-processing step, it was important to set the model hyperparameters essential for training. The hyperparameters were initially decided based on intuition, however, after model training and cross-validation the final set of hyperparameters were set based on empirical results. The hyperparameters that were finalised are shown in Figure 2. To have the ideal overview, the hyperparameters once fixed, were unchanged throughout the individual subject cases that were considered. The CNN architecture was modified as described in [3], as the exponential linear unit (ELU) activation functions were mainly used. ReLU activation functions were used at the output due to better performance based on empirical results and considering the nature of the data. It is important to consider that the decision to use a larger value for the learning rate was strictly due to the shortage of time

The model architectures are mentioned in Section 8 of the report. The design of the CNN architecture was largely inspired by [3]. Schirrmeister et al. took these factors into consideration in the design of their Deep ConvNet architecture. The main modifications to this architecture were in the first convolutional block, where single 25 x 44 convolutional kernel were replaced with two 3 x 3 kernels and a single 18 x 1 kernel. The intuition was that adding more convolutions with smaller strides would aid in feature extraction.Since this is a classification task, the outputs of all the models have been completed by using a fully connected neural network. Outputs from the convolution/pooling layers and RNN-specific architectures were passed to a fully connected neural network as previously mentioned to aid in the learning process using methods such as batch normalization and dropout layers. This in turn would decrease hyperparameter sensitivity.

4. Results

4.1. Subject 1 v/s All Subjects

[Figure 3] and [Figure 4] show the comparison in the plots for the single subject case i.e, Subject 1 and the All subjects case. The plots shown are for the training and validation loss/accuracy plots specifically obtained using the cascaded CNN-LSTM hybrid architecture. The plot was obtained by plotting Metrics v/s Number of Epochs.

Even though the hyperparameters were fixed for most cases, for CNN-LSTM to better the performance and to obtain higher accuracy, the model architecture was changed.

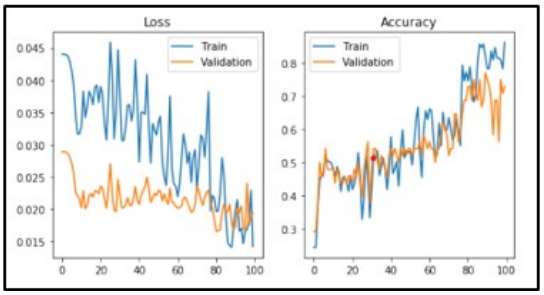

Figure 3: *Training and Validation plots for Subject 1*

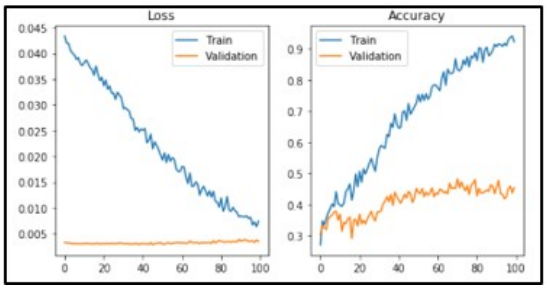

Figure 4: *Training and Validation plots for the All Subjects case*

Two separate notebook files have been submitted to verify the same. The best model for the classification task was CNN-LSTM, and a detailed description of the model will be explored in the following sections.

4.2. CNN+LSTM

One of the major models considered in the classification task was CNN-LSTM. This was one model for which all the experimentation was done with respect to the model architecture. Particularly one of the best-performing architectures contained 4 blocks of CNN and 1 LSTM followed by a leakyReLU activation, max pool, and dropout layers. The training was done for 80 epochs with the Adam optimizer, and a test accuracy of 71.5% was seen with a batch size of 64 and a learning rate of 1e-3. More epochs did not significantly cause a rise in the test accuracy however, a point to be noted was that the ReLU activation function when combined with this architecture, caused overfitting and couldn't beat the baseline model. Adding more CNN layers should give better performance for ReLU based on intuition, but it can lead to the vanishing gradient problem for LSTM, which is prone to overfit the model making it more complex. L1/L2 regularization can be considered to tackle the problem of overfitting.

The initial attempt started with a filter size of 16, and it was doubled on every CNN layer till 4 blocks, keeping other hyperparameters the same. The learning rate was varied to avoid overfitting, by decreasing from 1e-3 to 1e-4. The accuracy dropped by 25%. The filter size was then increased, which boosted the model accuracy by 4% from the previous value. To deal with the higher training accuracy, the number of epochs was reduced from 100 to 80, and that resulted in a test accuracy of 71.5%. The intuition behind this is that increasing the filter size in CNN, increases the receptive field of each neuron, allowing it to capture more global features, which in turn reduces the risk of overfitting local features of the data.More trainable parameters was increased which it turned helped.

	precision	recall	f1-score	support
class1	0.73	0.74	0.74	444
class2	0.67	0.75	0.71	508
class3	0.68	0.60	0.64	384
class4	0.69	0.67	0.68	436
accuracy			0.69	1772
macro avg	0.69	0.69	0.69	1772
weighted avg	0.69	0.69	0.69	1772

Figure 5: *Classification Report - Trained and tested on entire dataset*

Based on the classification report of [Figure 5], classifying class 3 was difficult for the model compared to classes 1, 2, and 4. Class 3's precision, recall, and f1-score values indicate that the model was not able to correctly classify this class. while the low recall value for class 3 implies that the model missed a large number of data that belonged to this class. The performance of the model may have been impacted by the imbalance in the number of samples per class or the unique features of the data for class 3.

4.3. Training all Subjects and testing 1 Subject

In this case, the accuracy attained was 50% after training on all 9 subjects for CNN+LSTM and testing on just 1 subject. This suggests that the trained model may be biased toward the data from the other 8 subjects and has an inherent difficulty classifying 1 specific subject resulting in no effective learning in the model. After training all 9 subjects, the model would have learned some patterns that are common to all subjects, but when testing a single subject, the model did not learn the pattern specific to that subject.

	precision	recall	f1-score	support
class1	0.59	0.50	0.54	48
class2	0.56	0.67	0.61	48
class3	0.48	0.38	0.42	56
class4	0.41	0.50	0.45	48
accuracy			0.51	200
macro avg	0.51	0.51	0.51	200
weighted avg	0.51	0.51	0.50	200

Figure 6: *Classification Report - Trained on the entire dataset, tested on subject 1's dataset*

Based on the classification report as shown in [Figure 6], classes 3 and 4 have poor F1 scores. It's likely that these classes have distinctive patterns that the model finds challenging to learn.

4.4. Training and testing 1 Subject

In this case, the model was trained and tested on a single subject and achieved training, validation, and testing accuracy values of 85%, 45.24%, and 66%, respectively, for CNN+LSTM architecture. The model overfits the data set it was trained on, resulting in better training accuracy but lower validation accuracy. However, the unexpectedly higher testing accuracy shows that either the validation set may not have been representative of the test data or that the test data may have been more similar to the training data than the validation data.

Based on the confusion matrix and classification report of [Figure 7] and [Figure 8] respectively, the model outperformed in classifying class 1 in classes 2, 3, and 4. Class 2's precision,

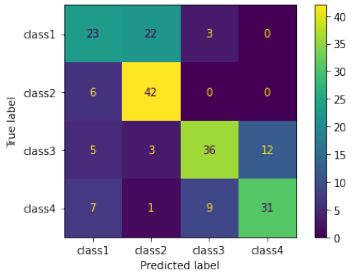

Figure 7: *Confusion matrix - Trained & tested on subject 1's dataset*

	precision	recall	f1-score	support
class1	0.56	0.48	0.52	48
class2	0.62	0.88	0.72	48
class3	0.75	0.64	0.69	56
class4	0.72	0.65	0.68	48
accuracy			0.66	200
macro avg	0.66	0.66	0.65	200
weighted avg	0.67	0.66	0.66	200

Figure 8: *:Classification Report- Trained & tested on subject 1's dataset.*

recall, and f1-score values indicate that the model was able to correctly classify this class, whereas class 1's results indicate that the model had difficulty classifying this class. The lower precision value suggests that the model misclassified some samples from other classes as class 1, while the low recall value for class 1 implies that the model missed a large number of data that belonged to this class. The performance of the model may have been impacted by the imbalance in the number of samples per class or the unique features of the data for class 1.

5. Discussion

Based on the results obtained, the cascaded CNN + LSTM architecture outperformed the other four methods that were considered. An important step before the training process is the pre-processing of the dataset, as without it, for a single subject using the best architecture, the test accuracy obtained was only close to 50%. But with the pipeline considered, the accuracy rose to 74 An additional observation was that the cascaded architectures (i.e., CNN+LSTM and CNN+GRU) typically performed better than individual methods (i.e., CNN, LSTM, and GRU) on this dataset. RNN's innate ability to capture the temporal structure found within the "downsampled" EEG signals via CNNs increases the overall classification performance. The CNN alone would not see this information, and individual RNNs may be overloaded by the size of the raw input data. Single-subject models were more performative than all-subject models, as expected, an above-average accuracy of 54% was still achieved for all-subject classification via cascaded architectures. While the cascaded CNN+LSTM typically performed better than the CNN+GRU architecture as expected, we observed that the CNN+GRU architecture trained much more efficiently as expected, given that the GRU contains fewer parameters than an LSTM. The answer for question 3 mentioned in the project guidelines PDF has been answered in the README regarding the relation between classification accuracy and time. An increasing trend was observed as time increases.

References

[1] Y. L. Qiqi Zhang. Improving brain-computer inter-face performance by data augmentation with conditional deep convolutional generative adversarial networks, 2018. https://arxiv.org/pdf/1806.07108.pdf.

[2] C. Brunner, R. Leeb, G. R. Muller-Putz, and A. Schlogl. BCI Competition 2008 – Graz data set A. page 6.

[3] R. Schirrmeister, J. Springenberg, L. Fiederer, M. Glasstet-ter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related informa-tion from the human eeg. 03 2017.

[4] D. Foster, S. Kakade, and R. Salakhutdinov. Domain adap-tation: Overfitting and small sample statistics, 2011.

6. Appendix

6.1. Dataset Statistics

Subject 1 Model: Data Statistics

```
Training/Valid data shape: (237, 22, 1000)
Test data shape: (50, 22, 1000)
```

Subject 2 Model: Data Statistics

```
Training/Valid data shape: (236, 22, 1000)
Test data shape: (50, 22, 1000)
```

Subject 3 Model: Data Statistics

```
Training/Valid data shape: (236, 22, 1000)
Test data shape: (50, 22, 1000)
```

Subject 4 Mode: Data Statistics

```
Training/Valid data shape: (234, 22, 1000)
Test data shape: (50, 22, 1000)
```

Subject 5 Mode: Data Statistics

```
Training/Valid data shape: (235, 22, 1000)
Test data shape: (47, 22, 1000)
```

Subject 6 Mode: Data Statistics

```
Training/Valid data shape: (236, 22, 1000)
Test data shape: (49, 22, 1000)
```

Subject 7 Mode: Data Statistics

```
Training/Valid data shape: (238, 22, 1000)
Test data shape: (50, 22, 1000)
```

Subject 8 Mode: Data Statistics

```
Training/Valid data shape: (232, 22, 1000)
Test data shape: (50, 22, 1000)
```

Subject 9 Mode: Data Statistics

```
Training/Valid data shape: (231, 22, 1000)
Test data shape: (47, 22, 1000)
```

All Subject Models: Data Statistics

```
Training/Valid data shape: (2115, 22, 1000)
Test data shape: (443, 22, 1000)
```

6.2. Model Comparison - Test Accuracy

Subject	CNN	LSTM	CNN + LSTM	GRU	CNN + GRU
1	0.51	0.40	**0.74**	0.40	**0.7**
2	0.34	0.36	0.46	0.42	0.69
3	0.58	0.36	**0.74**	0.42	**0.72**
4	0.24	0.4	0.48	0.32	0.38
5	0.38	0.34	0.38	0.31	0.40
6	0.47	0.31	0.36	0.35	0.39
7	0.50	0.40	0.42	0.28	0.38
8	0.48	0.38	0.62	0.38	0.42
9	0.43	0.34	0.64	0.36	0.66
All Subjects	0.37	0.29	0.71	0.28	0.54

6.3. CNN Architecture

Block	Type	Size	Value
Convolutional Pool 1	*Conv*	1 x 10	*stride = 1*
	BatchNorm	-	*momentum = 0.2*
	Conv	3 x 3	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	Conv	3 x 3	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	Conv	18 x 1	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 2	*Conv*	25 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 3	*Conv*	50 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 4	*Conv*	100 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
Fully Connected	*Linear*	200 x 54	-
	BatchNorm	-	*momentum = 0.2*
	ReLU	-	-
	Dropout	-	*p = 0.4*
	Linear	54 x 44	-
	BatchNorm	-	*momentum = 0.2*
	ReLU	-	-
	Linear	44 x 4	-

6.4. LSTM Architecture

Block	Type	Size	Value
LSTM	*LSTM*	22 x 64 x 3	*dropout = 0.4*
Fully Connected	*Linear*	64 x 54	-
	BatchNorm	-	*momentum = 0.2*
	ReLU	-	-
	Dropout	-	*p = 0.4*
	Linear	54 x 44	-
	BatchNorm	-	*momentum = 0.2*
	ReLU	-	-
	Linear	44 x 4	-

6.5. CNN + LSTM Architecture

Block	Type	Size	Value
Convolutional Pool 1	*Conv*	1 x 10	*stride = 1*
	BatchNorm	-	*momentum = 0.2*
	Conv	3 x 3	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	Conv	3 x 3	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	Conv	18 x 1	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 2	*Conv*	25 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 3	*Conv*	50 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 4	*Conv*	100 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
LSTM	*LSTM*	7 x 64 x 3	*dropout = 0.4*
Fully Connected	*Linear*	64 x 4	-

6.6. GRU Architecture

Block	Type	Size	Value
GRU	*GRU*	22 x 64 x 3	*dropout = 0.4*
Fully Connected	*Linear*	64 x 54	-
	BatchNorm	-	*momentum = 0.2*
	ReLU	-	-
	Dropout	-	*P = 0.4*
	Linear	54 x 44	-
	BatchNorm	-	*momentum = 0.2*
	ReLU	-	-
	Linear	44 x 4	-

6.7. CNN + GRU Architecture

Block	Type	Size	Value
Convolutional Pool 1	Conv	1 x 10	*stride = 1*
	BatchNorm	-	*momentum = 0.2*
	Conv	3 x 3	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	Conv	3 x 3	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	Conv	18 x 1	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 2	Conv	25 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 3	Conv	50 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
	Dropout	-	*p = 0.4*
Convolutional Pool 4	Conv	100 x 10	*stride = 1*
	ELU	-	*alpha = 0.9*
	BatchNorm	-	*momentum = 0.2*
	MaxPool	1 x 3	*stride = (1,3)*
GRU	GRU	7 x 64 x 3	*dropout = 0.4*
Fully Connected	Linear	64 x 4	-