Decoding EEG Signals with Deep Learning
ECE C247 Winter 2020 Project

Alexie Pogue
UID: 204517263
anpogue@ucla.edu

Amirali Omidfar
UID: 204869201
omidfar@ucla.edu

Eric Peltola
UID: 204590970
ericpeltola@ucla.edu

Kenny Chen
UID: 505219294
kennyjchen@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 Short-Term 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, we investigated the use of generative models such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) for data augmentation. The insights gained throughout this project provided a better understanding about the choices of various parameters and architectures on this classification task and in deep learning as a whole.

1. Introduction

Machine learning has proven to be an invaluable tool in brain-computer innovation to assist and rehabilitate individuals with brain-related impairments. To this end, this 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.

Our primary concern in data preprocessing was the limited quantity of training set examples. To offset the potential for overfitting, measures were taken in data regularization and augmentation. Fixed noise reduction techniques such as band pass filtering and data smoothing using a Hanning window were used. Among regularization approaches were Gaussian noise injection and random erasing of data clusters. Methods to increase the number of viable training examples were also considered using VAEs and GANs.

The focus of our work was on the construction and analysis of neural network architectures for signal decoding experiments. In total, a CNN, LSTM, GRU, CNN + LSTM and CNN + GRU were designed. Using these models we were interested in assessing different architectures individually and in combination for the evaluation of data with temporal structure. Specific methods, results, and analysis of dominant trends are given in the following sections.

2. Methods

2.1. Pre-Processing the Training Data

As the EEG dataset contained only 2115 total samples, regularization was an important factor to consider, as training deep networks on a relatively small training set is prone to overfitting [3]. In addition to directly increasing the training set size via data augmentation, other methods of preprocessing the input EEG data were considered for this study.

The preprocessing pipeline used in this study was comprised of four functions, the parameters of which are detailed in Table 1. The first two functions, a band-pass filter and a smoothing function, were performed before training. During training, the last two processes were applied every time a sample was taken from the training set. The $DataLoader$ class in Pytorch allows for 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, whereby a random rectangular mask of the input sample is set to zero.

Section 3.1 of this paper includes a study which compared the performance of deep learning architectures with and without input data preprocessing.

2.2. Details on Training Procedure

After preprocessing the EEG data as described in the section above, we annealed through different values of each hyperparameter and settled with a learning rate of 0.0005, $\epsilon = 1\text{e}{-08}$, $\beta = \{0.9, 0.999\}$, and a weight decay of 0.0005 for an Adam with cross-entropy loss optimizer, which was chosen empirically. For a fair comparison across all different architectures, we used the same hyperparameters during

Table 1. Pre-Processing Pipeline

Stage	Process	Parameters
1	Band-pass filter	$order = 3$ $f_{LP} = 56$ $f_{HP} = 240$
2	Smoothing	window = $'hanning'$ window size = 5
3	Gaussian Noise	$\mu = 0$ $\sigma = 1$

4	Random Erasing	aspect = 1:40
		scale = 2-8 %

the training of each model using a batch size of 32 for 100 total epochs. As we used a modified version of the CNN architecture as described in [5], we mainly used exponential linear unit (ELU) activation functions; however, we chose to use rectified linear units (ReLU) activation functions in the subsequent fully-connected layers as they are known to be empirically more performative. We trained all models using Google CoLaboratory's provided NVIDIA 1080Ti GPU. We note here that although a higher training accuracy could have been achieved with a lower learning rate and additional epochs, we chose to use a higher learning rate in the interest of time.

The model architectures are given in Tables 3 - 7. Data characteristics of EEG signals like low SNR and temporal structure make learning features more difficult than for static images CNNs are typically most successful on. The design of the CNN architecture was largely inspired by [5]. 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 we replaced the single 25 x 44 convolutional kernel with two 3 x 3 kernels and a single 18 x 1 kernel. It was thought that adding more convolutions with smaller strides would aid in feature extraction; specific layers and kernel sizes were determined empirically by trial and error. All architectures were cascaded with an FC to classify the EEG features into a gesture. Outputs from the convolution/pooling and RNN-specific architectures were input to a deep FC to improve learning using methods such as batch normalization and dropout layers in order to decrease hyperparameter sensitivity and further regularize data, respectively.

3. Results

3.1. Preprocessing Effects on Performance

The results reported in Table 2, with the exception of Subject 1, were generated using preprocessed training data according to the pipeline described in Section 2.1. The dataset corresponding to subject 1 was chosen for a comparison study assessing the effects of preprocessing. Parenthetical results in Table 2 (corresponding to label "1") indicate training data was input without using the preprocessing pipeline. It is clear from results that preprocessing has a significant positive impact on classification accuracy.

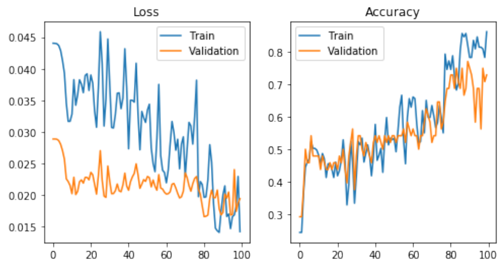

Figure 1. The training and validation loss/accuracy plots of Subject 1 using our cascaded CNN-LSTM architecture with preprocessing and data augmentation across 100 epochs.

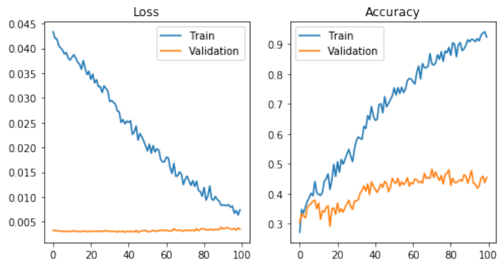

Figure 2. The training and validation loss/accuracy plots with all subject data using our cascaded CNN-LSTM architecture with preprocessing and data augmentation across 100 epochs.

3.2. Decoding Performance

In order to test the relative performance of the neural network architectures described in Section 2, all models were trained and tested on the EEG dataset with fixed hyperparameters for consistent evaluation. To evaluate performance as a function of architecture, we study test set accuracies between subjects and across all subjects. Each row of Table 2 (corresponding to labels "1-9") contains the results from a subject-specific subset of the EEG dataset. Data specific to a single subject was used to train each of the architectures in these experiments. The final row of Table 2 (corresponding to label "All") contains the testing accuracy for each architecture trained across all subjects. Results corresponding to best performance are highlighted with bold text.

3.2.1 Individual Architectures

Referring to individual architectures in Table 2, with the exception of subject 4, the CNN model performed the best for both the individual subject experiments and across all subjects. In experiments corresponding to subjects 6 and 7, the CNN model had the best performance over all architectures with testing accuracies of 47 % and 50 % respectively. While we expect the LSTM and GRU models are suitable for temporal data, the deep CNN architecture was still able to extract key features better. The batchnorm and dropout layers within the CNN architecture aided in model generalization, while it was observed that at approximately 60 epochs the LSTM and GRU training accuracies were converging to high values close to 1. Increasing batch size helped reduce the issue, however larger training sets would have also helped. Between the GRU and LSTM architectures, performance varied. This is expected given results from literature.

3.2.2 Cascaded Architectures

Referring to Table 2, we see that in general, the CNN+LSTM architecture is most optimized for EEG signal decoding. The CNN + LSTM model performed best in 6 of the 9 single-subject experiments. On the full dataset, the CNN+LSTM and CNN+GRU performed similarly, with testing accuracies of 53% and 54%, respectively.

An example experiment is shown in Fig. 1 for the CNN + LSTM architecture trained on subject 1. It is clear from the loss plot in this figure and the parenthetical results in Table 2 that data preprocessing was effective in improving model generalization. The corresponding validation curve in the accuracy plot further confirms this. The training and validation accuracies measured across all subjects is shown in Fig. 2. The large gap in loss and accuracy indicates that the network is overfitting the training data. We imagine this overfitting could be mitigated by more sophisticated data augmentation methods given by VAEs and GANs.

4. Discussion

In this project, we analyzed various deep learning architectures for a classification task using EEG signals and determined that, on average, our cascaded CNN + LSTM architecture outperformed the other four methods we compared against. We observed that data preprocessing and training data augmentation played a large role in the model performance, in that for a single subject using our best architecture, we could only achieve 48% accuracy without preprocessing, whereas with our entire pipeline we reached 74% for the same subject. Additionally, we observed that cascaded architectures (i.e., CNN+LSTM and CNN+GRU) typically performed better than individual methods (i.e., CNN, LSTM, and GRU) on this dataset. This may be explained by an RNN's ability to capture the temporal structure found within the "downsampled" EEG signals via a CNN that could increase the overall classification performance, whereas a singular CNN would not see this temporal information and individual RNN's may be overloaded by the datasize of the raw input data from not using a CNN.

While the cascaded CNN+LSTM typically performed better than the CNN+GRU architecture as expected, we observed that the CNN+GRU method trained and inferred much more efficiently as expected, given that the GRU contains less parameters than an LSTM. Single-subject models were more performative than all-subject models as expected, but we were still able to achieve an above-chance accuracy of 54% for all-subject classification via cascaded architectures. There are, of course, other methods that we could have tried in order to further increase performance, and the below sections briefly describe two techniques we attempted: VAE's and GAN's.

4.1. Additional Data Augmentation

4.1.1 Variational Autoencoders (VAEs)

This study included an attempt at augmenting the training dataset using a VAE, which is designed to compress the input image into a latent distribution which is easily sampled to create simulated data. Four VAE's were created, each of which produced simulated data for a specific target. The structure of the VAE's encoder and decoder networks were fully connected networks with a depth of three and a latent dimension of 800. All four VAE's were used to create an additional simulated dataset, resulting in a doubling of the total training data. n Figure 8 contains a sample from the training set along with its corresponding output from the VAE. It was clear from the reconstruction that the VAE did learn some large-scale features of the input data. However, some features were learned more clearly than others, and the reconstructed sample was more sparse than the original.

Augmenting the EEG classification training set with these VAE-simulated samples did not have an appreciable effect on the performance of the best-performing architecture. The emulated samples are promising, but further work is needed to improve the performance of the class-specific VAE.

4.1.2 Generative Adversarial Networks (GANs)

Inspired by [1] and [4] we decided to use Deep Convolutional GAN to improve the performance of our convolutional neural network.

Following the architecture mentioned in [1], the generator $G(z)$ consists of strided 1-dimensional convolutional-transpose layers, 1 dimensional batch norm layers and ReLU activations. The output of the generator then went to a tanh function. (This step is required as for GAN, we normalized the data to be between [-1,1]).

On the discriminator side $D(x)$, we had a binary classification network that took data with the same shape of EEG signals as input and output whether the input was fake (created by the generator network) or not. The architecture was a series of strided 1-dimensional convolutional layers, 1 dimensional BatchNorm and LeakyReLU layers. At the end it output the final probability as a Sigmoid activation function.

However, solving for the minimax optimization of GAN (1), our generator's loss did not converge zero, meaning the network was not learning. We believe it's due to our large selected stride and dilation values. (Architecture mentioned in Appendix 9). Given we got other methods to augment data we did not further debug our GAN implementation.

References

[1] S. C. Alec Radford, Luke Metz. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016. `https://arxiv.org/pdf/1511.06434.pdf`.

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

[3] D. Foster, S. Kakade, and R. Salakhutdinov. Domain adaptation: Overfitting and small sample statistics, 2011.

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

[5] R. Schirrmeister, J. Springenberg, L. Fiederer, M. Glasstetter, 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 information from the human eeg. 03 2017.

Table 2. Model Comparison via Test Set Across Subjects - {After (Before) Preprocessing}

Subject	CNN	LSTM	GRU	CNN + LSTM	CNN + GRU
1	0.56 (0.38)	0.38 (0.36)	0.44 (0.26)	**0.74** (0.48)	0.70 (0.52)
2	0.42	0.36	0.36	0.38	**0.70**
3	0.50	0.36	0.42	**0.74**	0.72
4	0.20	0.40	0.32	**0.44**	0.42
5	0.38	0.34	0.34	**0.40**	**0.40**
6	**0.47**	0.39	0.35	0.35	0.35
7	**0.50**	0.36	0.38	0.48	0.38
8	0.44	0.38	0.38	**0.62**	**0.62**
9	0.43	0.34	0.36	**0.64**	0.66
All	0.38	0.29	0.29	0.53	**0.54**

Table 3. CNN Architecture

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

Table 4. LSTM Architecture

Block	Function	Size	Parameter
LSTM	$LSTM$	$22 \times 64 \times 3$	$dropout = 0.4$
Fully-Connected	$Linear$	64×54	$-$
	$BatchNorm$	$-$	$momentum = 0.2$
	$ReLU$	$-$	$-$
	$Dropout$	$-$	$p = 0.4$
	$Linear$	54×44	$-$
	$BatchNorm$	$-$	$momentum = 0.2$
	$ReLU$	$-$	$-$
	$Linear$	44×4	$-$

Table 5. CNN + LSTM Architecture

Block	Function	Size	Parameter
Conv-Pool 1	$Conv$	1×10	$stride = 1$
	$BatchNorm$	$-$	$momentum = 0.2$
	$Conv$	3×3	$stride = 1$
	ELU	$-$	$alpha = 0.9$
	$BatchNorm$	$-$	$momentum = 0.2$
	$Conv$	3×3	$stride = 1$
	ELU	$-$	$alpha = 0.9$
	$BatchNorm$	$-$	$momentum = 0.2$
	$Conv$	18×1	$stride = 1$
	ELU	$-$	$alpha = 0.9$
	$BatchNorm$	$-$	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1, 3)$
	$Dropout$	$-$	$p = 0.4$
Conv-Pool 2	$Conv$	25×10	$stride = 1$
	ELU	$-$	$alpha = 0.9$
	$BatchNorm$	$-$	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1, 3)$
	$Dropout$	$-$	$p = 0.4$
Conv-Pool 3	$Conv$	50×10	$stride = 1$
	ELU	$-$	$alpha = 0.9$
	$BatchNorm$	$-$	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1, 3)$
	$Dropout$	$-$	$p = 0.4$
Conv-Pool 4	$Conv$	100×10	$stride = 1$
	ELU	$-$	$alpha = 0.9$
	$BatchNorm$	$-$	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1, 3)$
LSTM	$LSTM$	$7 \times 64 \times 3$	$dropout = 0.4$
Fully-Connected	$Linear$	64×4	$-$

Table 6. GRU Architecture

Block	Function	Size	Parameter
GRU	GRU	$22 \times 64 \times 3$	$dropout = 0.4$
Fully-Connected	$Linear$	64×54	–
	$BatchNorm$	–	$momentum = 0.2$
	$ReLU$	–	–
	$Dropout$	–	$p = 0.4$
	$Linear$	54×44	–
	$BatchNorm$	–	$momentum = 0.2$
	$ReLU$	–	–
	$Linear$	44×4	–

Table 7. CNN + GRU Architecture

Block	Function	Size	Parameter
Conv-Pool 1	$Conv$	1×10	$stride = 1$
	$BatchNorm$	–	$momentum = 0.2$
	$Conv$	3×3	$stride = 1$
	ELU	–	$alpha = 0.9$
	$BatchNorm$	–	$momentum = 0.2$
	$Conv$	3×3	$stride = 1$
	ELU	–	$alpha = 0.9$
	$BatchNorm$	–	$momentum = 0.2$
	$Conv$	18×1	$stride = 1$
	ELU	–	$alpha = 0.9$
	$BatchNorm$	–	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1,3)$
	$Dropout$	–	$p = 0.4$
Conv-Pool 2	$Conv$	25×10	$stride = 1$
	ELU	–	$alpha = 0.9$
	$BatchNorm$	–	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1,3)$
	$Dropout$	–	$p = 0.4$
Conv-Pool 3	$Conv$	50×10	$stride = 1$
	ELU	–	$alpha = 0.9$
	$BatchNorm$	–	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1,3)$
	$Dropout$	–	$p = 0.4$
Conv-Pool 4	$Conv$	100×10	$stride = 1$
	ELU	–	$alpha = 0.9$
	$BatchNorm$	–	$momentum = 0.2$
	$MaxPool$	1×3	$stride = (1,3)$
GRU	GRU	$7 \times 64 \times 3$	$dropout = 0.4$
Fully-Connected	$Linear$	64×4	–

Table 8. Variational Autoencoder

Layer	Function	Size	Parameter
1	$Linear$	22000×4000	–
2.1	$Linear$	4000×800	–
2.2	$Linear$	4000×800	–
3	$Linear$	22000×4000	–
4	$BatchNorm$	4000	$momentum = 0.1$
5	$ReLU$	–	–
6	$Sigmoid$	–	–

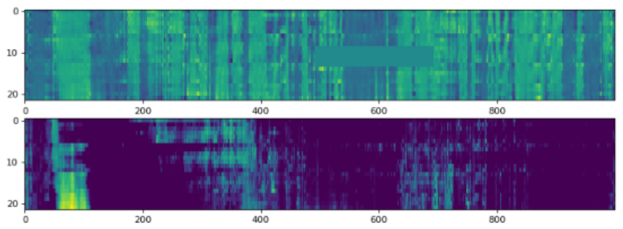
Figure 3. (Top) sample image from training set. (Bottom) VAE reconstruction of the above signal.

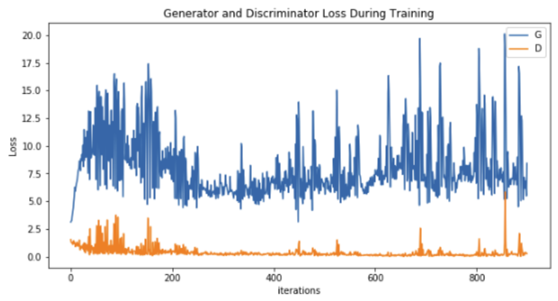

Figure 4. The loss of the discriminator and the generator functions of our GAN.

Table 9. DCGAN Generator

Layer	Function	Parameters {padding, dilation, size, stride}
1	$TransposeConv$	$\{0,0,4,4\}$
2	$TransposeConv$	$\{1,3,4,3\}$
3	$TransposeConv$	$\{2,3,4,4\}$
4	$TransposeConv$	$\{2,3,4,3\}$
5	$TransposeConv$	$\{1,3,4,3\}$

Table 10. DCGAN Discriminator

Layer	Function	Parameters {padding, dilation, size, stride}
1	$Conv$	$\{1,3,4,4\}$
2	$Conv$	$\{2,3,4,3\}$
3	$Conv$	$\{2,3,4,4\}$
4	$Conv$	$\{1,3,4,4\}$
5	$Conv$	$\{0,1,4,4\}$

Figure 5. Minimax Optimization Equation for GAN's:

$$minmax\ V(D,G) = \mathbb{E}[log(D(x)] + \mathbb{E}[log(D(1 - D(G(z)))] \quad (1)$$