Two Architectures for Between-Subjects EEG Classification

Nicolas Trammer
colet@ucla.edu
005395690

Alexander Yu
alexy23@ucla.edu
105295708

Joseph Yu
josephyu@ucla.edu
105423503

Matthew Yu
matthyu@ucla.edu
305422178

March 20, 2023

Abstract

For this project, we examined two distinct architectures for use in classifying nine subjects' EEG data. The first architecture consisted of one-dimensional convolutional layers and recurrent layers. The second architecture consisted of one-dimensional convolutional layers and transformer-encoder layers. The transformer-based model achieved 61.2% test accuracy. The recurrent model achieved 70.6% test accuracy after ensembling two of these models and required fewer training epochs and parameters than the transfomer-based model. In conclusion, we found that the 1D-convolutional recurrent model provides an efficient and performant algorithm for EEG classification.

1. Introduction

We tested two types of architectures trained on the entire nine-subject dataset:

1. Architecture consisting loosely of one-dimensional convolutional layers followed by recurrent LSTM layers.
2. Architecture consisting of one-dimensional convolutional layers followed by transformer-encoders.

Detailed specifications of these architectures can be found in A.3 of the appendix. During training and testing of these models, we performed data augmentation as outlined in A.1.

1.1. 1D CNN-LSTM

Our best performing architecture, which we refer to as 2CRNN, was inspired by [5], who devised a 1D CNN-LSTM for use in classifying EEG data. Intuitively speaking, the CNN allows the model to learn local spatiotemporal features. The LSTM layers then learn long-term associations in the time series data while hopefully avoiding vanishing gradients.

Our model consists of an initial dense layer, two convolution layers, a bidirectional LSTM layer, and a final dense layer. We used the ReLU activation function for the initial dense layer and the convolution layers, and tanh for the LSTM layer. Each of the convolution layers is followed by a max-pool layer, a batch normalization layer, and a dropout layer. There is another dropout layer following the LSTM layer as well. We empirically found that these layers helped the model converge faster and prevented overfitting. There are a total of 110,532 trainable parameters. We trained two of these models and ensembled them during test time.

Independently of 2CRNN, we also explored another architecture, to which we refer as 3CRNN, inspired by [3]. We found that 3CRNN did not perform as well as 2CRNN, so we will not report findings in detail here but rather in the appendix section.

1.2. Transformer

We also investigated applying a Transformer-Encoder model based on Vaswani et al.'s seminal paper on attention [4]. One challenge we had training 2CRNN was reducing the number of time samples given as input, given that it is hard for the model to encode every single time-slice into a single hidden state vector. A transformer-based approach reduces these challenges, because it does not condense the entire time history, and instead the model learns which time-slices to pay attention to. On the other hand, transformer models tend to have more parameters and thus require more training data to achieve good performance.

Since we were concerned only with classification, the model applied transformer-encoders in succession, instead of having both encoder and decoder blocks. The encoder required embeddings for each time slice. These were learned using 2 successive 1D convolution layers, which extracted local features in the EEG data. Additionally, an additive sinusoidal positional encoding was added to each time slice. To reduce overfitting, the model had dropout layers after each CNN layer, as well as in the Transformer-Encoders. The transformer architecture required more parameters than 2CRNN (329,444 compared to 110,532).

2. Results

The best individual model of 2CRNN was able to reach a validation accuracy of 71.4% and a test accuracy of 67.3%. We then ensembled two 2CRNNs from separate training runs, which achieved a test accuracy of 70.6%. As shown in Figure 1, 2CRNN achieved relatively high validation and training accuracy in fewer than thirty epochs.

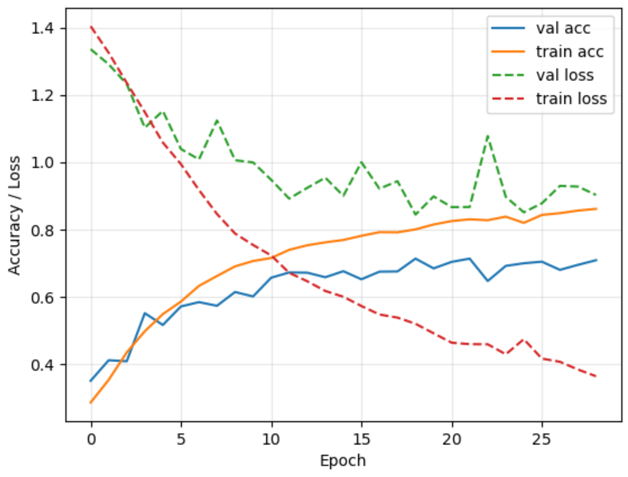

Figure 1. Accuracy and loss for 2CRNN

The best individual model of Transformer-Encoder was able to reach a validation accuracy of 61.5% and a test accuracy of 61.2%. Compared to 2CRNN, the transformer was about 10% worse on the validation set and 6% worse on the test set, and was trained on a larger number of epochs.

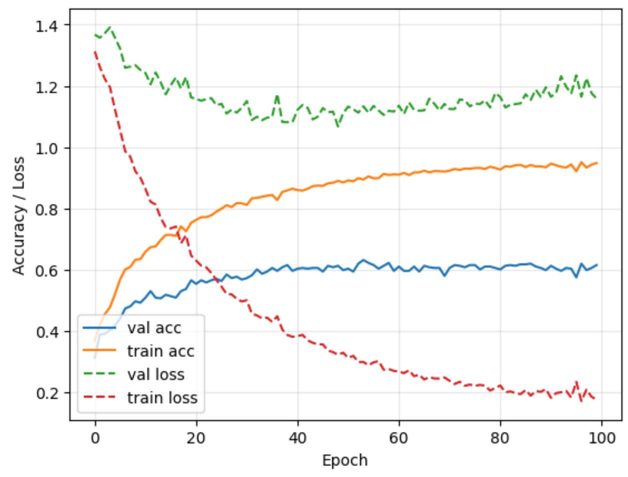

Figure 2. Accuracy and loss for the Transformer-Encoder

3. Discussion

To improve model performance, we tested and implemented the following techniques: dropout, batch normalization, layer normalization, and dataset augmentation. Dropout improved the models' performance on the validation and test sets, likely because it reduces the amount of overfitting. Empirically, we found batch normalization or layer normalization improved the 2CRNN model's performance by approximately +2%.

We found the most crucial technique to improving performance to be dataset augmentation. To augment the data, we sliced and applied stride starting at different offsets with respect to the time dimension. This procedure is explained in depth in A.1. This additional data allowed the a more complex model to be learned without overfitting too much. Via grid search, we empirically found that the best performing time slice to be time bin 50 to 450 with a stride of 6. However, we ultimately chose to use time bins 0 to 500 with a stride of 5. We expanded the time slice to be cautious about data loss and to hopefully improve generalization to the test data. We chose a stride of 5 in order for the length of the time slice to be divisible by the stride.

We suspect that slicing and striding improved performance since ignoring certain data points may allow the model to focus on global features rather than local features. This in turn might allow the model to learn better by providing data with lower dimensionality. When training the model on every time sample, we found that every model we tried did not generalize well, achieving only 25% validation accuracy. We suspect that these later time samples are not useful for this classification task.

As noted previously, Figure 1 shows that the 2CRNN model attains good performance in terms of accuracy within relatively few epochs. However, the validation loss appears somewhat unstable. This observation suggests that the model may not generalize relatively well but also that we may be able to improve 2CRNN by introducing even more regularization.

As shown in Figure 2, we noticed very fast convergence in the Transformer model and after which, not much improvement was made. Annealing the learning rate could result in slightly higher performance. But more importantly, adding more data augmentation could help the larger transformer model to not overfit so severely.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens,

Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 4

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. 5

[3] Yingnan Sun, Frank P.-W. Lo, and Benny Lo. Eeg-based user identification system using 1d-convolutional long short-term memory neural networks. *Expert Systems With Applications*, 125:259–267, 2019. 1

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 1

[5] Dechun Zhao, Renpin Jiang, Mingyang Feng, Jiaxin Yang, Yi Wang, Xiaorong Hou, and Xing Wang. A deep learning algorithm based on 1d cnn-lstm for automatic sleep staging. *Technol Health Care*, 30(2):323–336, 2022. 1

A. Methods

A.1. Data augmentation

We performed data augmentation on each data example in the following manner:

1. The data were trimmed to the first 500 out of 1000 time bins.
2. We then defined a hyperparameter s.
3. s augmentations were derived as follows:
   (a) One augmentation was constructed by taking every s time bins, starting at time bin 0. In other words, this step took time bins 0, s, $2s$, etc.
   (b) The above step was repeated, except starting at different time bins from 1 to $s - 1$. For example, starting at an offset of 1 used time bins 1, $s + 1$, $2s + 1$, etc.
4. Another augmentation was constructed by taking the average of every s time bins and adding Gaussian noise.
5. Another augmentation was constructed by taking the max of every s time bins.

This procedure generates $(s + 2) \cdot m$ total examples, where m is the original number of examples. We carried out data augmentation on the training and the validation sets separately as to maintain independence of the two sets during training time. At test time, we also applied this same data augmentation procedure to the test set. For any given test example, the classifier's prediction was the class with the highest cumulative score across this example's augmentations.

A.2. Summary of Results

Table 1 shows a summary of the test accuracies for each of the three architectures that we tested. We trained and tested only using all nine subjects' data after applying the aforementioned data augmentation strategy.

Model	Validation Accuracy	Test Accuracy
2CRNN	71.4%	67.3%
2CRNN ensembled	-	70.6%
3CRNN	61.9%	59.6%
Transformer-Encoder	61.5%	61.2%

Table 1. Summary of the performance of various architectures

A.3. Architecture Specifications

No.	Layer	Unit Size	Specifications	# Params	Output Size
1	Dense	128	ReLU	2,944	(N,100,128)
2	Conv1D	64*3	ReLU, stride=1	24,640	(N,100,64)
3	MaxPool	-	length=stride=2	-	(N,50,64)
4	BatchNorm	-	-	256	(N,50,64)
5	Dropout	-	rate=0.5	-	(N,50,64)
6	Conv1D	128*5	ReLU, stride=1	41,088	(N,50,128)
7	MaxPool	-	length=stride=3	-	(N,17,128)
8	BatchNorm	-	-	512	(N,17,128)
9	Dropout	-	rate=0.5	-	(N,17,128)
10	LSTM	32	tanh	41,216	(N,64)
11	Dropout	-	rate=0.5	-	(N,64)
12	Dense	4	softmax	260	(N,4)

Table 2. 2CRNN Architecture

We built and trained 2CRNN using the TensorFlow Keras API [1]. 'Same' padding was included in the convolution layers, and we used a batch size of 64. We trained this model with the Adam optimizer with learning rate $1\text{e}{-3}$ and zero weight decay, and sparse categorical cross entropy as the loss function. We trained this model with an early stopping condition that terminated training when the validation loss had not decreased after ten epochs.

No.	Layer	Unit Size	Spec	Params	Output
1	AvgPool	-	length=stride=2	-	(N,22,100)
2	Dropout	-	rate=0.25	-	(N,100,22)
3	Conv1D	32*2	ReLU, stride=1	1,440	(N,32,100)
4	Conv1D	64*2	ReLU, stride=1	4,160	(N,64,100)
5	Conv1D	128*2	ReLU, stride=1	16,512	(N,128,100)
6	Dense	98	ReLU	12,640	(N,100,98)
7	Dropout	-	rate=0.5	-	(N,100,98)
8	LSTM	64	41,984	41,216	(N,64)
9	Dense	4	softmax	260	(N,4)

Table 3. 3CRNN Architecture

We built and trained 3CRNN using the PyTorch API [2]. ‘Same’ padding was included in the convolution layers, and we used a batch size of 64. We trained this model with the Adam optimizer with learning rate $1e{-}4$ and weight decay $1e{-}5$, and sparse categorical cross entropy as the loss function. We trained this model for 200 epochs.

No.	Layer	Unit Size	Spec	Params	Output
1	Conv1D	32*2	ReLU, stride=1	1,440	(N,32,100)
2	Dropout	-	rate=0.5	-	(N,32,100)
3	Conv1D	64*2	ReLU, stride=1	4,160	(N,64,100)
4	PositionalEncoding	-	dropout=0.5	-	(N,64,100)
5	TransformerEncoder	4H→1024FF	dropout=0.5	149,056	(N,100,64)
6	TransformerEncoder	4H→1024FF	dropout=0.5	149,056	(N,100,64)
7	LayerNorm	-	-	128	(N,100,64)
8	Flatten	-	-	-	(N,6400)
9	Dense	4	softmax	25,604	(N,4)

Table 4. Transformer-Encoder Architecture

We built and trained the transformer-encoder model using the PyTorch API [2]. ‘Same’ padding was included in the convolution layers, and we used a batch size of 64. Each of the TransformerEncoder layers consists of a four-head self-attention and 1024-unit feedforward network, as well as layer normalization. We trained this model with the Adam optimizer with learning rate $1e{-}4$ and weight decay $1e{-}3$, and sparse categorical cross entropy as the loss function. We trained this model for 100 epochs.