Investigating Deep Neural Network Architectures for Decoding EEG Data

Rohan Varma
University of California, Los Angeles
rvarm1@ucla.edu

Michael Moon
University of California, Los Angeles
oceanlittle@ucla.edu

Jennifer Liaw
University of California, Los Angeles
jdliaw@ucla.edu

Abstract

The ABSTRACT is to be in fully-justified italicized text, at the top of the left-hand column, below the author and affiliation information. Use the word "Abstract" as the title, in 12-point Times, boldface type, centered relative to the column, initially capitalized. The abstract is to be in 10-point, single-spaced type. The abstract may be up to 3 inches (7.62 cm) long. Leave two blank lines after the Abstract, then begin the main text.

1. Introduction

To tackle the task of classification on EEG data, several different algorithms were evaluated in order to find which approach performed best on the task. For classification, Convolutional Networks and Recurrent Network architectures immediately came to mind. For data generation, which we also investigated, a variation auto encoder was used.

1.1 CNNs

We hypothesized that convolutional neural networks could be useful for this task as one natural interpretation of the data is as a 22 * 1000 image.

1.1.1 First iteration: AlexNet

The first CNN architecture was loosely modeled after the AlexNet [1]. The AlexNet architecture is as follows: [CONV-POOL-NORM]x2 - CONVx3 - POOL - FCx3 – OUT. The architecture we drew up also had 5 convolutional layers and 3 FC layers, but we pooled and applied batchnorm at every convolutional layer instead of only the first two, and also applied batchnorm before we did the maxpool: [CONV-NORM-POOL]x5 - FCx3 - OUT .

1.1.2 Second iteration: LeNet

The second iteration on our CNN architecture was modeled closer to the LeNet-5[2]. The LeNet architecture is as follows: [CONV-POOL]x2 - CONV - FC – OUT. Since other architectures such as ResNet showed that more layers in a net is better, we increased the number of convolutional layers to four and included batchnorm to achieve the following architecture: [CONV-NORM-POOL]x4 - FC - OUT

1.2 RNNs

For RNNs, in order to mitigate the vanishing and exploding gradients problem, we investigate the use of an LSTM cell and a GRU cell, for performance reasons. A recurrent network structure makes natural sense for this data, as the data is presented as 22 features changing across 1000 timesteps.

1.2.1 LSTM Cell

The vanishing and exploding gradient problems are well known issues in training recurrent neural network; these issues are largely avoided by the use of an LSTM [3] cell. We investigate a wide range of hidden unit size for the LSTM cell, ranging from 8 hidden units to 128 hidden units.

1.2.2 GRU Cell

Gated Recurrent Units, or GRUs [4], are an alternative to LSTM cells that have fewer parameters, since they do not maintain a cell state. We hypothesized that the use of fewer parameters will not only improve our computation time, but also help with overfitting.

1.2.4 Bidirectional RNN, Variational Dropout, Dropout

We hypothesized that for optimal prediction of an output label at a certain timestep, it would be beneficial to have data from the past and from the future. This is accomplished with the use of a bidirectional recurrent structure, which maintains recurrent connections going forward and backwards in time. In addition, to combat overfitting, we attempted to dropout in the recurrent connections of our network, known as variational dropout, as well as traditional dropout before our fully-connected layer in our recurrent network.

1.3 VAEs

Variational Autoencoders (VAEs) are an architecture that allow learning the distribution of the features, which in turn allows the machine to generate examples from the distributions it has learned. In this way, it is a generative model. We sought to use a VAE to achieve a compact representation of our EEG signal data, and by minimizing the reconstruction error and KL divergence from our original data, we hoped that this encoded representation could be used as features into our model. Since we did not end up using the reconstructed data in a classification model, we only discuss our VAE in the appendix.

2 Results

2.1 CNN Results

In this section, the results from the best version of both a deep and shallow CNN will be presented and evaluated. For each CNN, the average and max statistics were taken across the learning rates, where the max corresponds to the best learning rate. Finally, to compare the results of the deep and shallow CNNs, the average and maximum of the best test accuracies were calculated.

2.1.1 Deep ConvNet

The full architecture of the Deep CNN can be found in our appendix. The bolded cells in the table represent the best test accuracies achieved for each learning rate.

Average test accuracies for train on one, test on one

batch size	epoch				
	1	5	10	20	40
238	0.293	0.330	**0.343**	0.305	0.306
200	0.287	0.293	**0.373**	0.356	0.303
150	0.287	0.347	0.313	0.303	**0.387**
100	0.322	0.310	0.310	0.347	**0.375**
50	0.316	0.276	0.320	0.330	**0.330**
average		0.362	maximum		0.387

Average test accuracies for train on one, test on one

batch size	epoch				
	1	5	10	20	40
238	0.293	0.330	**0.343**	0.305	0.306
200	0.287	0.293	**0.373**	0.356	0.303
150	0.287	0.347	0.313	0.303	**0.387**
100	0.322	0.310	0.310	0.347	**0.375**
50	0.316	0.276	0.320	0.330	**0.330**
average		0.362	maximum		0.387

For train on all, test on all, the hyperparameters used were: epochs=50, learningrate=0.001, batchsize=150.

Test accuracies for train on all, test on all

Subject	Test accuracy
1	0.40
2	0.22
3	0.30
4	0.31
5	0.38
6	0.32
7	0.32
8	0.27
9	0.33

2.1.2 Shallow ConvNet

The architecture of the Shallow CNN can be found in our appendix. The bolded cells in the table represent the best test accuracies achieved for each learning rate.

Average test accuracies for train on one, test on one

batch size	epoch				
	1	5	10	20	40
238	0.347	0.403	**0.430**	0.397	0.440
200	0.327	0.380	**0.433**	0.420	0.370
150	0.345	0.420	0.420	0.435	**0.450**
100	0.297	0.373	0.420	**0.470**	0.440
50	0.303	0.378	0.380	0.420	**0.440**
average		0.445	maximum		0.470

Maximum test accuracies for train on one, test on one

batch size	epoch					lr (learning rate)
	1	5	10	20	40	
238	0.38	**0.52**	0.46	0.36	0.46	5e-4
200	0.34	0.42	**0.52**	0.44	0.36	5e-4
150	0.30	**0.52**	0.43	0.43	0.44	1e-3
100	0.28	0.38	**0.52**	0.50	0.40	1e-3
50	0.24	0.44	0.52	**0.56**	0.40	1e-3
	average		0.53	maximum		0.56

For train on all, test on all, the hyperparameters used were: epochs=50, learningrate=0.005, batchsize=100.

Test accuracies for train on all, test on all

Subject	Test accuracy
1	0.30
2	0.46
3	0.32
4	0.21
5	0.19

6	0.32
7	0.30
8	0.32
9	0.33

2.2 LSTM & GRU

For the LSTM and GRU, we considered the problem of training and then testing on each of the nine subjects. Specifically, we trained our model on data from one subject, and then evaluated the test accuracy, and then repeated this process across all subjects. Hyperparameters used were learningrate = 0.001, hidden dim = 200, and the RNN was bidirectional. These hyperparameters were selected via running a grid search, and we considered the hyperparameters that gave the best accuracy on a validation dataset, with using the LSTM architecture. Due to computation restraints, we did not optimize hyperparameters for the GRU, instead using the same hyperparameters as the LSTM. Our results are given in the table below:

Subject	Test accuracy, LSTM	Test accuracy, GRU
1	0.38	0.34
2	0.39	0.36
3	0.37	0.35
4	0.31	0.31
5	0.39	0.32
6	0.38	0.39
7	0.34	0.4
8	0.33	0.37
9	0.36	0.38

3 Discussion

In terms of CNN architectures, we compared a shallow and deep architecture. We found that the shallow net performed better when training on one subject and testing on one subject however performed worse in the trial including all subjects. We believe this occurred because the shallow net performs worse on larger training sets as it does not have the expressivity, where the deep net performed better because of its depth. Overfitting was the main motivation for exploring the number of epochs and batch sizes as a hyperparameter, finding that somewhere between 10 to 20 epochs was an optimal period for training length and around 100 to 150 for the batch size.

In general, we did not achieve great success with either of the recurrent neural network architectures. We believe that this is primarily due to overfitting, an issue we were unable to completely mitigate. While varying the hidden units between 8 and 128, we noticed that around a hidden size of 20, both RNN architectures begin to simply memorize the data. Therefore, we incur an extremely small loss, but often, our architecture does no better than random during test time. Using hidden units less than a size of 20, we discovered that our network would not learn at all. We believe that these two issues were because the LSTM/GRU model is overly expressive for the problem at hand – even though there are 1000 timesteps and 22 features at each timestep, there are only 4 output labels; therefore, it is likely that a lot of features and timesteps are redundant, and our network was only fitting the noise in these features and timesteps, as opposed to learning an overall representation that can generalize to unseen inputs.

In addition, we came across an interesting paper [5] that compares recurrent neural network architectures to convolutional neural network architectures, and evaluates their performance on sequential modeling tasks. The authors ultimately recommend that "convolutional networks should be regraded as a natural starting point for sequence modeling tasks", a statement that we agree with, at least based on the results of experimenting on EEG data.

Finally, we summarize our efforts to create a VAE in order to encode the data into a compact representation. We dealt with significant bugs while implementing the VAE, most prominently, a negative loss. To mitigate this issue, we ultimately had to define the loss as the sum of the MSE between the original data and the reconstruction, plus the KL divergence. However, this resulted in an insignificant decrease in the loss during our training process, and the reconstructed data were not visually similar when plotted against the original. We believe that this is due to replacing the binary cross entropy loss in the original paper with an MSE loss.

5 References

[1] Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 2012.

[2] LeCun et al. Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE. 1198.

[3] Hochreiter et al. Long Short-Term Memory. 1997.

[4] Chung et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. 2014.

[5] Bai et al. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. Preprint. 2018.

Appendix: Summary of Performance of All Algorithms

Train on one, Test on One

Network Type	Max Accuracy
Shallow ConvNet	0.52
Deep ConvNet	0.56
LSTM	0.39
GRU	0.4

Deep CNN, train on all, test on all

Test accuracies for train on all, test on all

Subject	Test accuracy
1	0.40
2	0.22
3	0.30
4	0.31
5	0.38
6	0.32
7	0.32
8	0.27
9	0.33

Shallow CNN, train on all, test on all

Test accuracies for train on all, test on all

Subject	Test accuracy
1	0.30
2	0.46
3	0.32
4	0.21
5	0.19
6	0.32
7	0.30
8	0.32
9	0.33

LSTM & GRU, individual accuracies, training on one, testing on one:

Subject	Test accuracy, LSTM	Test accuracy, GRU
1	0.38	0.34
2	0.39	0.36
3	0.37	0.35
4	0.31	0.31
5	0.39	0.32
6	0.38	0.39
7	0.34	0.4
8	0.33	0.37
9	0.36	0.38

Appendix: Summary of Architectures & Additional discussion

Our first CNN was modeled after ALexNet. Full architecture details are given below. At a high level, we had 5 conv layers and 3 FC layers, and used pooling and batchnorm at every convolutional layer, instead of only the first 2.

Further details on each individual layer:

1. CONV: 32 filters, 8x8
1.1. NORM: normalization layer
1.2. POOL: 2x2 (future pool layers omitted because they are all the same)
2. CONV: 32 filters, 5x5 [NORM-POOL]
3. CONV: 64 filters, 3x3 [NORM-POOL]
4. CONV: 64 filters, 3x3 [NORM-POOL]
5. CONV: 64 filters, 3x3 [NORM-POOL]
6. FCx3 - OUT

The maxpool filters were of size 2x2 and the first convolutional had an 8x8 filter, the second convolutional had a 5x5, and the remaining three convoluationals all had 3x3 filters. The activation chosen for this architecture was ReLU.

Our next CNN architecture was modeled similar to LeNet. Also taking motivation from ResNet, we increased the number of conv layers and added batchnorm. Further details on each layer:

Further details on each individual layer:

1. CONV: 32 filters, 5x5
1.1. NORM: normalization layer

1.2. POOL: 2x2, applied at stride 3 (future pool layers omitted because they are all the same)
2. CONV: 32 filters, 3x3 [NORM-POOL]
3. CONV: 64 filters, 3x3 [NORM-POOL]
4. CONV: 64 filters, 3x3 [NORM-POOL]
5. FC: FC layer, 704 x 4
6: OUT: cross-entropy loss

This architecture still used ReLU for its activations and had 2x2 filters with a stride of 3 for the maxpool layers. The first convolutional layer used a 3x3 filter while the rest used 5x5 filters.

To further optimize the performance of the CNN, we explored some more enhancements to our existing architecture. Dropout was introduced and applied to the input of every convolutional layer with the exception of the first. The activation function was also switched from ReLU to ELUs:

$f(x) = x$ for $x > 0$, $f(x) = e^x - 1$ for $x <= 0$

To handle the large number of input channels, we tested out a split input approach. The first convolutional layer was split into two convolutions: one temporal (across time) and one spatial (across electrodes). There is no activation function in-between these two convolutions, which allows us to combine it into one convolutional layer.

Architecture details:

1.1 CONV: (temporal) 25 filters, 10x10
1.2 CONV: (spatial) 25 filters, 1x1
1.3 NORM: normalization layer
1.4 POOL: 3x3 maxpool
2. CONV: 50 filters, 10x10 [NORM-POOL]
3. CONV: 100 filters, 10x10 [NORM-POOL]
4. CONV: 200 filters, 10x10 [NORM-POOL]
5. FC: FC layer, 1400 x 4
6. OUT: cross-entropy loss

Although deep nets generally empirically perform better, we also decided to test out a Shallow CNN to see if it could produce better results or act as a baseline. The architecture we used for the Shallow CNN simply only had one convolutional layer, followed by a fully connected layer.

CONV - POOL - FC - OUT

The single convolutional layer still implemented the splitted input from 2.1.3, and also used ELU as its activation function.

Architecture details:

1.1 CONV: (temporal) 25 filters, 10x10
1.2 CONV: (spatial) 25 filters, 1x1
1.3 NORM: normalization layer
1.4 POOL: 3x3 maxpool
2. FC: FC layer, 1650 x 4

LSTM & GRU details

The hyperparameters we chose were 200 hidden units, a learning rate of 0.001, bidirectional as true, and we applied variational dropout in the recurrent layers as well as dropout before the linear layer in order to prevent overfitting. We used the Adam Optimizer, a batch size of 1, and a sequence of length 1 (i.e. we presented the inputs one timestep at a time).

We also tried several variants in training our recurrent network, as we were experiencing issues with fitting the data. We tried using backpropagation at each timestep, so that at timestep 1, we backprop 1 timestep, but at timestep 200, we backprop all 200 timesteps, and so on. This proved to be prohibitively computationally inefficient for us.

As a workaround, we used the assumption that the data at a timestep should not depend too much on a timestep that came far before – for example, the output at timestep 900 should not have a strong dependency on say, the input at timestep 200. Although we would like to backpropagate as much as possible, this proved to be infeasible, so we tried to limit our backpropagation 100 timesteps; effectively capping BPTT after it went back 100 timesteps.

Next, we also tried just a single 1000 timestep backpropagation, after presenting our entire input to the model. This proved to fit our data the best, indicating that we do not need to repeatedly backpropagate across timesteps while presenting the data; just once is enough.

VAE Discussion

We had several difficulties in training the VAE, the most prominent one being the issue of negative losses. We ultimately had to go with an MSE loss plus KL divergence to fix this bug, but we believe that deviating from the original paper led to subpar results.

Conclusion

Training deep neural networks is hard in practice, especially with unfamiliarity in the problem domain. We believe that we could have used data preprocessing and understanding the context of the domain to achieve better results. We are thankful to the professors, the TAs, and to piazza for helping us through this project and the class.