”I Can Read Your Mind”: Supervised Classification of Human Imagnination based on EEG data using Convolutional and Recurrent Neural Networks

Mani Tadayon
University of California, Los Angeles
Electrical Engineering Department
manitadayon@ucla.edu

Sahba Aghajani Pedram
University of California, Los Angeles
Mechanical Engineering Department
sahbaap@ucla.edu

Abstract

In this project, we have designed different architectures of convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) and assessed their performance and accuracies on the EEG dataset. Various techniques were introduced to incorporate more randomness into the dataset and avoid the network memorizing patterns within our training data only. Two different training strategies were deployed. In the first one, the net has been trained on one subject and tested across all and each individual subjects. In the second one, the net has been trained and tested across all subjects combined.
The best result for CNN was an architecture with three stacks of convolutional layers followed by a fully connected layer and a softmax which resulted in 0.6 testing accuracy (training on subject 1 and testing on subject 3). For the RNN, the architecture with 1 convolutional layer, followed by a LSTM layer and softmax resulted in 0.5 testing accuracy (training on subject 4 and testing on subject 1). For both the CNN and RNN, training across one subject and testing across different subject give us better results than training and testing across all subjects.
Overall, CNN architectures are superior to RNN counterparts when training on individual subjects while the performance of both nets are quite similar when training on all subjects combined.

1. Introduction

The main objective of this project is to use Electroencephalography (EEG) signals and predict human imagination performing one of four actions namely I) motion of the left hand, II) right hand, III) both feet, and IV) tongue. To accomplish this task, we deploy supervised learning framework and train different neural network architectures and assess their performances. The dataset used for this project [1] is gathered for 9 subjects; each subject is the recording of 25 electrodes. Each subject also consists of 288 trials. Each trial consists of 22 EEG and 3 EOG channels, and each channel contains 1000 time series data recorded at 250 Hz. Only the 22 EEG channels were used for classification purposes in this project.
Although CNNs have shown great promise especially in vision and image classification tasks [2, 3], the recent literature has suggested great potential of CNNs for brain-signals decoding [4, 5]. Moreover, since the EEG is temporal data by nature, RNNs are tested as well. To avoid the problem of vanishing the gradients, Long Short-term Memory (LSTM) [6] and Gated Recurrent Unit (GRU) [7] are used for this supervised classification task. We have implemented and assessed numerous CNNs, LSTM/GRU, and combination of CNN and LSTM/GRU architectures with different number of convolution and LSTM layers and cells, different number of filters, and dropout probabilities, among others.

1.1. Training/Testing Strategies

Two different schemes are employed to train/test neural net architectures in this project. These schemes are summarized as follow:
1- Under the first scheme, the data is trained across each subject and tested across all nine subjects. The logic behind this method is to assess how different nets can capture the ”within subject” features. Under this paradigm we are interested to examine how training across one subject can be generalized to other subjects.
2- Under the second schemes, the nets are trained and tested across all subjects. In this case, the nets ought to capture both ”within subject” and ”between subject” features and variations. Due to this complexity, we hypothesis that the performance, i.e. testing accuracies of nets would be lower as the data has more complexities to be learnt by the nets.

1.2. Data Augmentation and Processing

Data Augmentation: In comparison with the current practice of machine learning algorithms on computer vision tasks with millions of labeled data, relatively small number of data is available for EEG decoding. For example, the dataset in our project contains $\sim$ 2500-2600 labeled data across all subjects and trials combined which is three orders of magnitude less. Hence, we have employed methods for data augmentation to increase our data size. For example, we break the 4 sec signals for each subject and trial into smaller portions, i.e. 8 chunks of 0.5 sec signal with 125 time points and use them each time to train the network. It should be noted that this data augmentation could have a drawback as we might unintentionally break important temporal correlation embedded into the signal across different times. Along this line of thought and as suggested by Prof. Kao, this method of data augmentation was merely performed with nets including LSTM architectures.

Centering Signals: We zero out the mean of our data. This means we make all input centering around the origin which would positively give the classifier more room to wiggle. This might be a necessary task in many neural network frameworks as data is obtained through experiment and they have different means.

It should be noted that different methods were used for centering the data in each training/testing strategies described section 1.1. When the nets were trained across individual subjects, the mean of the data for that subject but across different trials was subtracted for the subject respectively. On the other hand, for training/testing across all subjects, the mean was first obtained across both trials and subjects combined and then subtracted from each subject and trial.

Noise (Randomness) Injection into Dataset: As we tested various architectures we realized that sometimes and especially for LSTM, the net only memorizes training data and cannot be well generalized. As a result, we have implemented various algorithms to introduce more randomness (noise) into our dataset. These algorithm are briefly discussed below:

1- Shuffling data across subject every time we train the network to introduce more randomness. We found this really helpful in increasing testing accuracy. This can be thought of using Monte Carlo method to train/test our net.

2- Adding additive white Gaussian noise (AWGN) to the data to force our network to see something new and random every time. The way this is implemented is as follow: First we calculate the mean and standard deviation of the signals then We generate and add noise to signals according to the desired SNR (signal to noise ratio). We used different methods for calculating mean and standard deviations of signals when individual or whole subjects were used for training.

3- Another method is to shuffle the data across time especially for the LSTM nets. Note that this might have revere impact on the the training/testing as we might break the temporal information in the signal.

2. Result

All architectures for this project were trained on three machines: I) 2 in 1 Lenovo Yoga with 16GB memory and 2.80 GHz CPU, II) a MSI laptop with 32GB memory and 2.8 GHz CPU and III) Lenovo X1 Yoga with 2.5 GHz CPU and 8 GB memory. No GPU has been used in this project. Figure 1 shows the top six neural net architectures (both CNN and RNN) which resulted in good classification accuracies amongst numerous nets that we have tried.

Table 1 also shows training, validation, and testing accuracies corresponding to these 6 architectures. Note that the "Training subject" in this table specifies which subject the net has been trained on. For example, '1' means that the net has been trained on subject 1 and 'all' means we have trained the net across all subjects. Note that after training the net, we test it for each subject which results in 9 different accuracies and we report the statistics of these numbers in the Test columns. Statistics in Train and Validation columns correspond to a given subject.

For the RNN we have implemented both LSTM and GRU with different number of filters and filter sizes. According to our observations CNN on average is marginally better than RNN when testing on the individual subjects. For example the best testing accuracies for CNN (Architecture II) was for training on subject 1 and testing on subject 3 which gives us 0.6 testing accuracy. The highest testing accuracy, however, was 0.5 for the LSTM architecture (Architecture IV). This was obtained by training on subject 4 and testing on subject 1.

For training across all subjects, RNN and CNN are comparable in performance ,i.e. 0.37 testing accuracy for CNN (architecture III) and 0.39 testing accuracy for LSTM (architecture I). It should be noted that, however, we could not overfit our architecture in either cases due to lack of resources. Upon accessing more computational resources and using GPUs, we could have trained these nets with more epoch/iterations and further differentiate between the networks. Table 1 includes more details about performance of each architecture.

3. Discussion

This section is divided into two parts. First, we discuss the effect of changing hyperparameters and including or excluding certain blocks on network's performance. Second, we discuss which architectures (CNN or RNN) perform better and why along with which methods of

training and testing is more effective.

3.1. Hyperparameters

The results of training numerous neural net architectures suggest that the recently developed bachnormalization algorithm [8] can significantly improve testing accuracies. We observed that it is better to include it immediately after convolutional and before nonlinearity layer. Adjusting value of dropout within the range of 0.5 to 0.8 proved to be very useful in our architectures; it avoids the network from overfitting. Our results also show that including dropout layer [9] right after pooling, LSTM, and fully connected networks can be very useful to overcome overfitting. However, we also found out that using too many dropout layers might significantly affect and reduce the performance of the net for capturing essential features of the data, i.e the net would not be able to even overfit the training data. We in fact found out that it is a good practice to come up with architectures that are able to overfit the training data first and then reduce overfitting by methods such as dropout afterward.
Based on our results choosing appropriate filter sizes and number of filters does really depend on architectures and layers used before and after it. For example 32 filters with kernel size 11 results in very good performance for LSTM case while it was insufficient for CNN architectures. We also found out that adding average pooling as the first layer significantly increases the ability of the net to overfit the training data especially for the LSTM architecture. Average pooling layer performs a moving average, i.e. low pass filter, to smooth out the data. Our results might suggest that it is easier for LSTM layer to learn low frequency features of EEG training signals compared to high frequency features. The reason for this can be simply that there are a lot less features at lower frequencies or these features are easier to learn by nature. Our result also shows that increasing number of cells in LSTM can further increases testing accuracies at the expense of training speed, which might not be feasible for many epochs and using machines running only on CPU. It was also observed that not necessarily adding and stacking layers of LSTM on top of each other increase accuracies, i.e. adding up to four layers can seriously lower the performance of the network and there is no guarantee to improve the performance. Another interesting observation is the difference between LSTM and GRU in terms of efficiency and speed. While they both give comparable testing accuracies (i.e ability to overfit the training data), GRU is almost three times faster than LSTM. for instance, training LSTM with 128 cells takes 37s per epoch while GRU with the same number of cells and on the same machine only takes 12s per epoch. This makes sense since GRU has less parameters than LSTM.
Another interesting observation was the choice of optimizer. While it is customary to choose the Adam optimizer as a default for the vision tasks, we observed that for networks with LSTM, stochastic gradient descent (SGD) with the right choice of learning rate and decay rate (i.e. 1 and 1e-6 respectively) can outperform Adam optimizer by small margin. For networks with convolutional layers only, however, Adam optimizer is still the best option.
we also observe that including one layer of CNN and incorporating CNN architecture in LSTM network seems to be best option for not only learning the features through a convolutional layer but also use LSTM to learn the temporal dependencies.

3.2. Training Methodologies for CNN and RNN

Now we analyze the results based on two different training/testing methods we introduced. The first observation is that the testing/training across each individual subject results in better testing accuracies. This result confirms our hypothesis stated earlier as training across all subjects the net has to learn variations between subjects as well which can be quite complicated. According to our results training across subject 4 with LSTM results in higher averaged testing and validation accuracies amongst all subjects. Interestingly, this observation was consistent for three other architectures with different dropout values, number of filters and filter sizes. Subject 9 is also the second best subject that gives us very good testing accuracies on average. training an LSTM network across all subject we obtain 0.39 testing accuracies while our training and validation accuracies were roughly 0.45. This means that we were unable to overfit the system due to lack of computational resources. For CNN architectures, we found out that training across subject 1 results in average testing accuracy of 0.58 followed by training accuracy for subject 2 with testing accuracy of 0.5. Complete results for various CNN and RNN architectures for individual and all subject training and testing is reported in Table 1.

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] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[4] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human eeg. CoRR, abs/1703.05051, 2017.

[5] Pouya Bashivan, Irina Rish, Mohammed Yeasin, and Noel Codella. Learning representations from eeg with deep recurrent-convolutional neural networks. arXiv preprint arXiv:1511.06448, 2015.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[7] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1):1929–1958, 2014.

	Training subject	Train			Validation			Test			
		min	mean	max	min	mean	max	min	mean	max	std
Architecture I	1	0.77	0.88	0.81	0.88	0.96	0.91	0.38	0.50	0.45	0.05
	2	0.83	0.88	0.85	0.85	0.92	0.87	0.36	0.38	0.36	0.07
	3	0.8	0.83	0.83	0.88	0.94	0.90	0.44	0.46	0.45	0.07
	4	0.98	1	1	0.98	1	1	0.44	0.46	0.45	0.04
	5	0.98	1	1	0.97	1	1	0.36	0.38	0.36	0.04
	6	0.56	0.61	0.59	0.63	0.77	0.75	0.42	0.48	0.46	0.06
	7	0.0.93	0.95	0.93	0.96	0.98	0.97	0.38	0.46	0.45	0.05
	8	1	1	1	1	1	1	0.42	0.44	0.42	0.04
	9	0.98	1	1	1	1	1	0.40	0.41	0.40	0.06
	all	0.45	0.45	0.45	0.44	0.4	0.44	0.39	0.39	0.39	
Architecture II											
	1	0.56	0.92	0.84	0.57	0.87	0.70	0.55	0.60	0.58	0.12
	2	0.64	0.91	0.81	0.78	0.99	0.90	0.41	0.5	0.45	0.04
	3	0.64	0.90	0.80	0.71	0.88	0.82	0.34	0.38	0.35	0.04
	4	0.59	0.93	0.80	0.65	0.97	0.85	0.4	0.42	0.41	0.07
	5	0.75	0.98	0.9	0.81	0.99	0.90	0.38	0.39	0.38	0.04
	6	0.72	0.97	0.88	0.81	1	0.95	0.38	0.44	0.43	0.07
	7	0.8	0.98	0.92	0.84	0.99	0.95	0.34	0.4	0.38	0.04
	8	0.72	0.97	0.87	0.79	0.99	0.92	0.32	0.34	0.33	0.03
	9	0.0.8	0.98	0.92	0.77	1	0.90	0.38	0.4	0.38	0.04
Architecture III											
	1	0.74	0.76	0.75	0.83	0.88	0.86	0.36	0.4	0.37	
	2	0.67	0.92	0.73	0.75	0.96	0.83	0.36	0.5	0.43	
	3	0.8	0.82	0.82	0.88	0.89	0.89	0.36	0.38	0.38	
	4	0.78	0.81	0.80	1	1	1	0.44	0.45	0.44	
	5	0.81	0.94	0.89	0.89	0.99	0.93	0.41	0.42	0.41	
	6	0.80	1	0.90	0.84	1	0.92	0.36	0.38	0.36	
	7	0.84	1	0.94	0.85	1	0.93	0.34	0.38	0.37	
	8	0.8	0.98	0.96	0.87	1	0.97	0.38	0.42	0.40	
	9	0.9	1	0.96	0.94	1	0.97	0.42	0.44	0.42	
	all	0.48	0.48	0.48	0.46	0.46	0.46	0.37	0.37	0.37	
Architecture IV											
	1	0.84	0.91	0.84	0.92	1	0.95	0.4	0.41	0.4	
	2	0.79	0.80	0.79	0.75	0.79	0.76	0.34	0.38	0.36	
	3	0.85	0.87	0.86	0.88	0.92	0.89	0.44	0.46	0.45	
	4	1	1	1	1	1	1	0.5	0.5	0.5	
	5	1	1	1	1	1	1	0.38	0.40	0.39	
	6	0.8	0.86	0.83	0.83	0.89	0.85	0.34	0.38	0.36	
	7	0.94	0.95	0.94	0.96	1	1	0.40	0.44	0.43	
	8	1	1	1	1	1	1	0.36	0.48	0.47	
	9	1	1	1	1	1	1	0.42	0.50	0.48	
	all	0.39	0.39	0.39	0.38	0.38	0.38	0.36	0.36	0.36	
Architecture V											
	all	0.45	0.45	0.45	0.44	0.44	0.44	0.39	0.39	0.39	
Architecture VI											
	all	0.46	0.46	0.46	0.45	0.45	0.45	0.38	0.38	0.38	

Table 1: Training, validation, and testing accuracies for different CNN and RNN architectures.

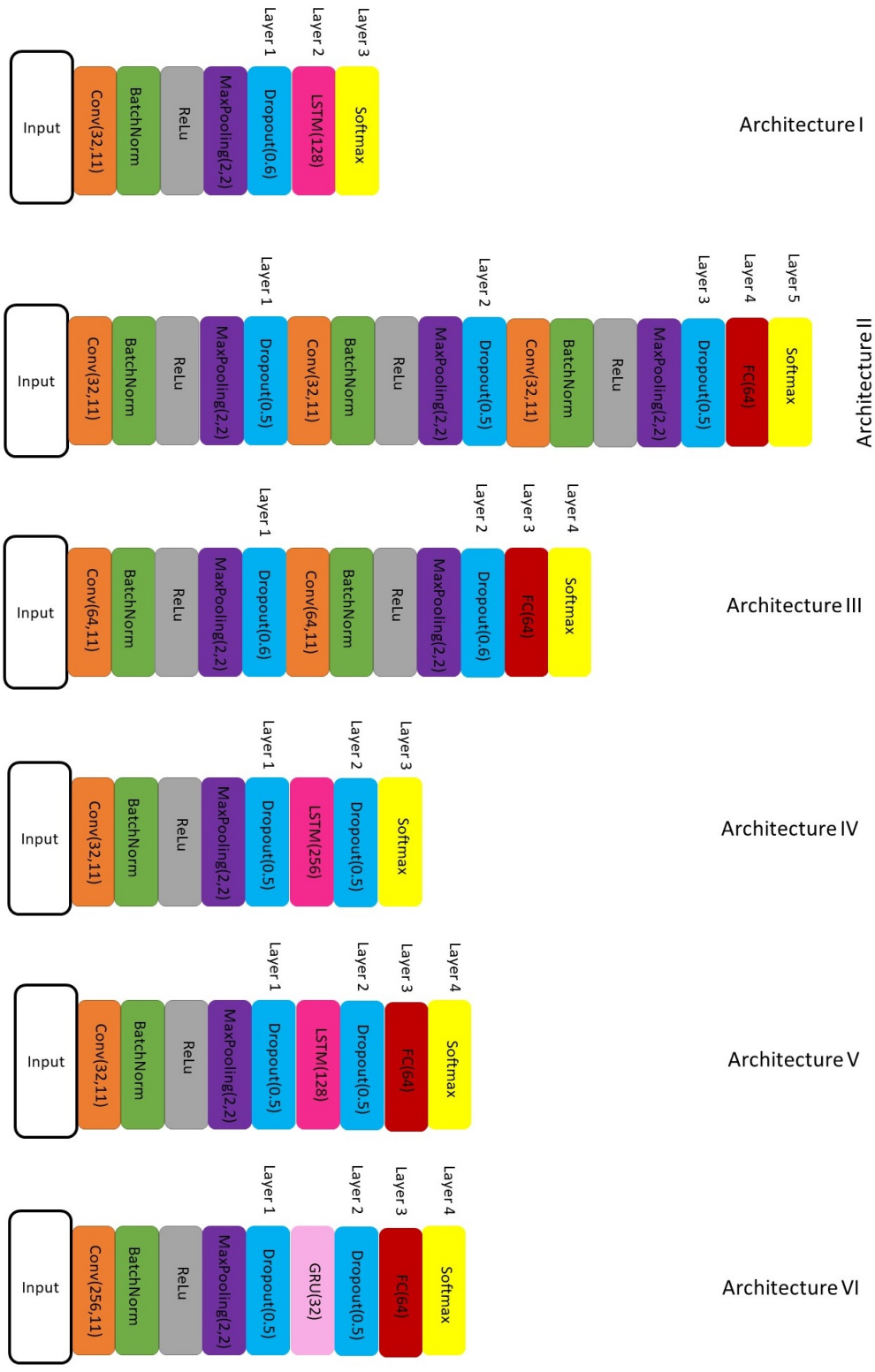

Figure 1: CNN and RNN architectures