# Deep Learning Project I: Prediction of finger movements from EEG recordings via deep neural networks

Cheng-Chun Lee, Sung-Lin Chan, Haziq Razali
School of Computer and Communication Sciences, EPFL

**Abstract**

The past decade have seen deep neural networks emerge as the de-facto solution for almost every machine learning related task. Core to its prevalence is the availability of large training databases and powerful modern computer hardware. In this paper, we conduct an empirical study on the use of deep neural networks for the prediction of finger movements from EEG recordings. We experiment with convolutional neural networks and recurrent neural networks to determine their usefulness when faced against a small sized time-series dataset. Experimental results indicate that a shallower simpler model is preferable for such a task.

*Keywords:* Deep Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks

## 1. Introduction

The renewed interest in deep neural networks began in 2011 when an architecture developed by Krizhevsky et al. [1] smashed the Imagenet challenge in said year. At the fundamental level, deep neural networks belong to the family of feature learning methods as opposed to methods that are task specific. Thanks to their representational power, deep networks have continued to hold state-of-the-art results in a wide range of tasks. From video analysis [2] to speech recognition [3], text generation [4], machine translation [5] and image captioning [6], the versatility of deep neural networks has proven to be an indispensable tool for machine learning practitioners.

In this paper, we experiment with deep neural networks for the prediction of finger movements from EEG recordings. We use the Data Set 4 of the BCI competition II [7] that contains 316 training samples and 100 test samples. The main challenge here is thus in developing an effective model given a relatively small set of training samples. The remainder of this paper is organized as follows. We provide a brief description of the dataset in section 2 and the components used in this paper in 3. We then specify training details in section 4.1 before presenting the results of our experiments in 4.2. Lastly, we conclude and suggest some possible ways to increase performance in section 5. The source code and results presented here are available at `https://github.com/SonyChan0807/DeepLearning-project1`.

## 2. Dataset

The training set contains 157 samples labeled 0 (upcoming movement in the left hand) and 159 samples labeled 1 (upcoming movement in the right hand). Each sample is composed of 28 EEG channels sampled at 1khz for 0.5s, thus having a dimension of (28 x 50 x 1000). Also provided in the dataset is the version that has been downsampled at 100Hz (28 x 50 x 100) which we use in our experiments as it is more manageable.

## 3. Deep Net Components

Two of the most prominent variants of deep nets that we will be experimenting with in this paper are the convolutional neural network (CNN) and recurrent neural network (RNN). For the purpose of this paper, we will omit a detailed discussion on the above-mentioned variants and would instead encourage the reader to refer to the vast literature available in the internet. We will only mention that while convolutional networks process the input in a hierarchical manner, recurrent networks do it in sequentially, both of which can be trained via gradient descent. For the subsequent experiments, we use a variant of RNNs, called Long-Short-Term-Memory (LSTM) cells due to its ability to combat the problem of vanishing / exploding gradients [8, 9], an issue which cripples the performance of vanilla recurrent networks.

# 4. Experiments

## 4.1 Training details

The performance and stability of all our models are tested using 5-fold cross validation. After model selection, in order to maximize usage of the training dataset, we re-train our best model over the entire dataset for 200 epochs. We initialize all parameters by sampling from a normal distribution with standard deviation of 1. We transform the categorical labels to a one-hot encoding and optimize over the cross entropy loss using stochastic gradient with a batch size of 10. We use the learning scheme of Adam [10] with a learning rate of 0.001, $\beta_1$ and $\beta_2$ of 0.9 and 0.999 respectively, and a weight decay of 0. We use the Pytorch library [11] to develop the models.

## 4.2 Results

**Baselines.** In the first experiment, we establish a simple baseline through several classifiers: Naive Bayes and Linear Support Vector Machines using the scikit library [12] with default parameters, and a 3-layered neural network with ReLu activation functions in the hidden layers using PyTorch. Since all these methods require the data to be a 1-dimensional input, we reshaped the raw signal to form a vector of length 1400. As expected, the performance of all these methods with the data processed naively is poor. This is most probably due to the fact that such a method assume no dependence among the inputs; their output is simply a non-linear combination of all input variables.

| **Method** | **Train** | **Val** | **Test** |
|---|---|---|---|
| Linear SVM | 88 % | 75 % | 60 % |
| Naive Bayes | 58 % | 53 % | 57 % |
| 3 Layered Neural Network | 85 % | 73 % | 70 % |

Table 1: Accuracies of the baselines

**Classification with CNN.** In the second experiment, we study the effects different configurations of the CNN have on the accuracies. Using CNN is a good idea based on the following two reasons. Firstly, Signal processing is often applied on time series to remove the noise or extract frequency components, and this can be done by 1D convolution. Secondly, 2D convolution can be used to find relations among channels during a time interval. We try the following two configurations: 1 CONV layer + 2 FC layers and 2 CONV layers + 1 FC layer. Each configuration is tested with different sets of hyperparameters (i.e., size of filter). In the first configuration, the CONV layer is composed of 28 2D filters with size 28x3/28x7, and we use dropout, maxpool and relu to prevent overfitting. In the second configuration, however, we don't use any of the techniques above because we found convolutional layers already suppress the effect of overfitting. And the first CONV layer is composed of 1 1D filter with size 1x13/1x11, and the second CONV layer is composed of 3 2D filter with size 28x7/28x5. That is, the first layer acts like signal processing on each channel, and 2D filter tries to find the relation among 28 channels. Lastly, we noted that preprocessing the input before feeding them to the architecture resulted in a degradation of accuracies.

**Classification with LSTM.** In our third experiment, we study the effects different configurations of the LSTM have on the accuracies. To avoid having to experiment with an excessively large number of combinations, we conduct our tests as follows: we first vary their number of hidden units and fix the subsequent fully connected layers - with the exception of the final classification layer - to have an equal number of units. The results of some of these experiments have been summarized in table 3 where the reported train and test accuracies are the ones that correspond to the best obtained validation accuracy. Firstly, it can be observed that increasing the number of fully connected layers from 1 to 2 worsens the accuracy. We also also noted in our experiments that the training accuracy is able to reach 100% even with the use of only 1 fully connected layer. This would make sense if we view the output of the LSTM as a feature vector representing the movement of the finger which are linearly separable. Overfitting on the training set thus becomes more likely with increasing model complexity (number of FC layers). Secondly, although the table seems to suggest that the ideal architecture is the one with 25 lstm and 25 fc units, we would like to point out it is not neccessarily true as the training is rather unstable. In other words, the three architectures with a single fully connected layer tend to outperform each other over repeated runs. This is also shown in the LSTM loss plots (figure 1) where it can be seen that there is a lot of oscillation going on. Lastly, we noted that preprocessing the input and/or augmenting the architectures presented in table 3 with dropouts [13] and ReLu [14] did not result in any improvement.

**Comparison between CNN and LSTM**. Next, if we look at the accuracies achieved by both models, it can be observed that the CNNs are performing better than their LSTM counterparts.

This is interesting as it is contrary to the results presented in many literature that have concluded otherwise. We speculate that this is due to the following reasons. Firstly, LSTMs presented in the literature have been trained on a much larger datasets ($\approx 10^5$ compared to 316). It is thus possible that the LSTMs in our experiments are learning spurious correlations that occur in the training data but not the real world (i.e. the test set). This hypothesis is further corroborated if we look at the loss plots in table 1 where we can see that the training and validation loss plots for the LSTMs tend to oscillate and are a lot more unstable compared to the CNNs that have a smoother plot. Secondly, we also think that this degradation in accuracy is due to the fact that the LSTM is unable to capture the most important elements of the past timesteps. This is because intuitively, the raw signal at each timestep does not contain as much information as a glove embedded word vector [15] or a feature vector extracted by a trained convolutional neural network in [2] since the two have been produced by trained classifiers. As such, information being propagated across the timesteps contain redundancy and or noise. Convolving across time via CNNs however, might have alleviated the issue of noise since its' filters can span several steps in both time and channel.

| **Architecture** | **Train** | **Val** | **Test** |
|---|---|---|---|
| CNN ($28 \times 28 \times 3$), FC (336,2) | $90 \pm 5\%$ | $84 \pm 4\%$ | 76% |
| CNN ($28 \times 28 \times 7$), FC (308,2) | $91 \pm 1\%$ | $79 \pm 3\%$ | 74% |
| CNN ($3 \times 1 \times 13$, $3 \times 28 \times 7$), FC (2) | $85 \pm 1\%$ | $83 \pm 3\%$ | 81% |
| CNN ($3 \times 1 \times 11$, $3 \times 28 \times 5$), FC (2) | $88 \pm 2\%$ | $83 \pm 3\%$ | 76% |

Table 2: CNN architecture and hyperparameters. The entry in the bracket indicates the number of units of the component with commas denoting a successive layers. The dimensions of the CNN are formatted as (no. of filters, height, width)

| **Architecture** | **Train** | **Val** | **Test** |
|---|---|---|---|
| LSTM (20), FC (2) | $87 \pm 2\%$ | $80 \pm 4\%$ | 64% |
| LSTM (25), FC (2) | $82 \pm 6\%$ | $82 \pm 2\%$ | 65% |
| LSTM (30), FC (2) | $87 \pm 4\%$ | $85 \pm 4\%$ | 63% |
| LSTM (20), FC (20,2) | $89 \pm 3\%$ | $88 \pm 4\%$ | 60% |
| LSTM (25), FC (25,2) | $90 \pm 2\%$ | $87 \pm 4\%$ | 59% |
| LSTM (30), FC (30,2) | $87 \pm 2\%$ | $88 \pm 3\%$ | 62% |

Table 3: LSTM architecture and hyperparameters. The entry in the bracket indicates the number of units of the component with commas denoting a successive layers.

## 5. Conclusion

We have presented several architectures for the prediction of finger movements from EEG recordings. Experimental results show that it is much more favourable to build a shallow model due to the lack of training samples, and that a pipeline leveraging CNNs seem to be more suitable when faced against a small time-series dataset set. We also noted an interesting outcome, that the use of any preprocessing methods seem to further degrade accuracy. Whether the results obtained here are conclusive - i.e. whether they generalize to larger datasets - are not known as we believe that a high-dimensional time-series dataset with only 316 samples is not representative of the true distribution of the data.

We believe that there are several improvements that can be made which might further enhance the performance of the models developed in this paper. Firstly, the inclusion of domain specific feature engineering and/or processing could prove useful as input to the architectures would be more meaningful than the raw signal, allowing it to learn better representations of the data. Secondly would be to use the CNN as a feature extractor for the LSTM. It was not attempted in this paper because we believe the data is too small for it to be effective. It is therefore more important to collect more training samples as it would make the training more robust.

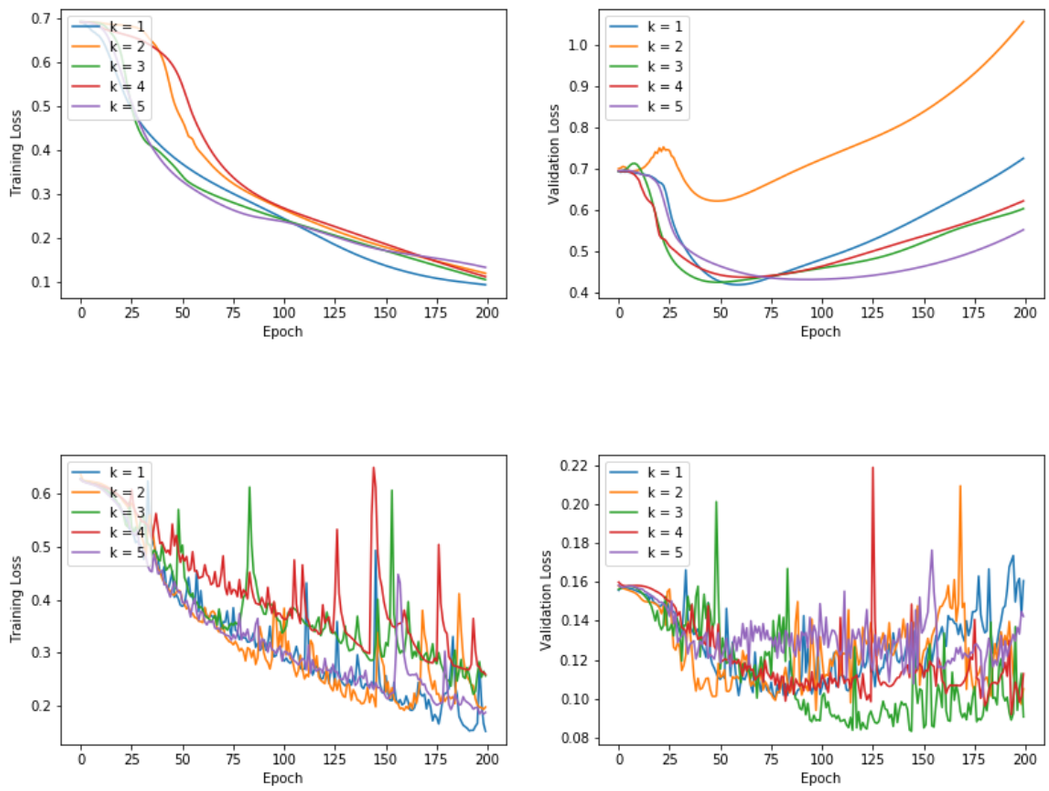


Figure 1: Training and validation loss plots for the (top) best CNN architecture and (bottom) best LSTM architecture.

## References


[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, 2012.

[2] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *Computer Vision and Pattern Recognition*, 2014.

[3] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," *Proceedings of Machine Learning Research*, 2014.

[4] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," *International Conference on Machine Learning*, 2011.

[5] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Neural Information Processing Systems*, 2014.

[6] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *Computer Vision and Pattern Recognition*, 2015.

[7] B. Blankertz, G. Curio, and K.-R. Muller, "Classifying single trial eeg: Towards brain computer interfacing," *Neural Information Processing Systems*, 2002.

[8] Y. Bengio and P. F. Patrice Simard and, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, 1994.

[9] S. Hochreiter, Y. Bengio, and J. S. Paolo Frasconi and, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies." *A field guide to dynamical recurrent neural networks*, 2001.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[12] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

[14] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *International Conference on Machine Learning*, 2010.

[15] R. S. Jeffrey Pennington and C. D. Manning, "Glove: Global vectors for word representation," *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.