EE-559 Deep Learning

Mini-project 1: Prediction of Finger Movements from EEG Recordings

---

Authors:

Terkel Bo Olsen
Anmol Porwal
Leandros Stefanou

Hand-in date: 18th of May 2018

1 Introduction

In this report, a binary-class data set consisting of repeated EEG recordings of a subject using either the left or right index finger is used to demonstrate the classification capabilities of (convolutional) neural networks. The data set was part of the 2003 "BCI Competition II" and was originally modelled and obtained by [1]. The modelling approach used by [1] consist of filtering the signals, using sliding window Fourier transform and specific channel selection followed by several conventional classificiation methods like support vector machines (SVM's), k-nearest neighbors (KNN), Sparse Fisher Discriminant (SFD) etc. With these methods they are able to obtain up to 96.8% mean accuracy.

In the following report a different approach is taken; by using different types of neural networks and with a minimum of preprocessing it is investigated how end-to-end learning can be applied to EEG signals by training directly on the time-series itself. The EEG signal used is sampled at a 100 Hz sampling frequency and consist of 50 samples of data spread across 28 different channels. Hence each channel consist of a 50 ms EEG measurement. We apply a simple preprocessing step to these signals in each model, consisting of a simple 1st order low-pass butterworth filtering with a cut-off frequency of 15 Hz. The filter design is briefly outlined in the next section.

2 Preprocessing

In the following section, it is briefly outlined how the EEG recordings are preprocessed using a first-order digital butterworth filter. This is motivated by the fact that EEG signals have a low signal-to-noise ratio [2] and hence valuable information can be significantly masked in the noise of the signal and hence potentially lead to significant overfitting when applying highly flexible models like a neural network. Hence, before training commences a first-order low-pass butterworth filter with a cut-off frequency of 15 Hz is applied to the signal. The cut-off frequency was treated as a hyperparameter and was tuned such that training loss reduced appropriately compared to the test loss, but values of 5 Hz to 20 Hz seem to work equally well. The frequency response of the filter is visualized in Figure 1 along with a sample channel and the filtered result of that channel. It is clear that the result of the filtering is a slight smoothing of the signal in such a way that most information is still available.

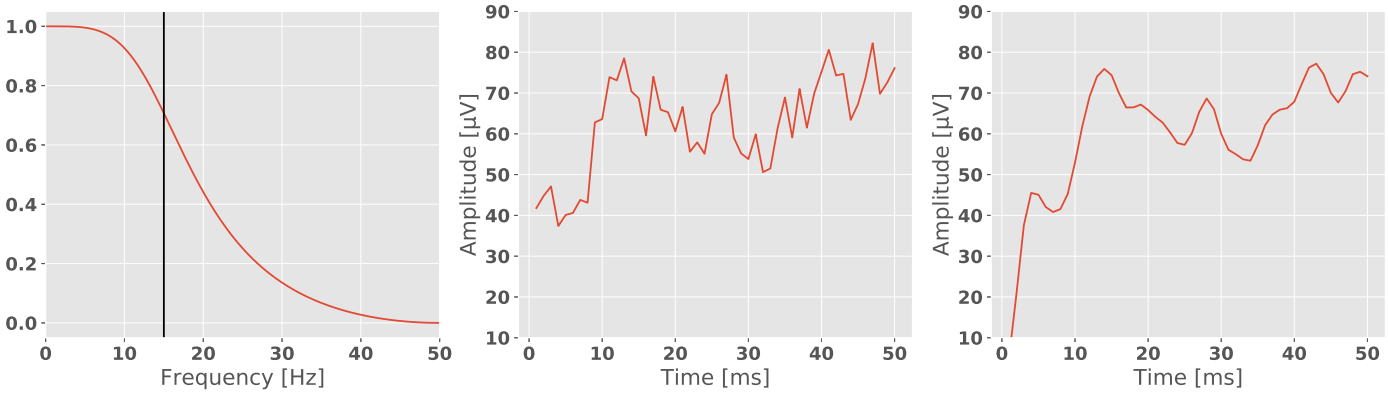

Figure 1: From left to right: Frequency response of the used filter (blue) along with the cut-off frequency (black), a sample channel and the filtered result of that sample channel.

3 Models

Having introduced the preprocessing used, now the models used will be described. Three different models were constructed during the experimental process each with increasingly better performance on the test set. First, a fully connected network is constructed, then a shallow convolutional network is constructed and lastly an extension of this shallow convolutional network into a deeper network is used which obtains good results. As the different networks use different methods, these are briefly introduced when needed. To compare the networks against a linear classifier, a logistic regression model was further created and the results of this model will be presented later in the results section.

3.1 Fully connected network

First, a simple yet large model is created consisting of a two hidden layers. The input size of the network is 1400 as the 28 channels, each consisting of 50 time points, is flattened into one long vector. The full network is visualized in Figure 2 and as seen after each layer the Exponential Linear Unit (ELU) activation function is applied. This activation works point-wise with the following formula:

$$f(x) = \begin{cases} (e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases} \tag{1}$$

Hence this activation function saturates at -1 for values much lower than zero and is linear for values greater or equal than zero. During training we apply batch-normalization and dropout with a relatively high probability of 0.75 to facilitate training and prevent overfitting respectively. The model includes a bias term in each layer and hence this model has an extensive amount of parameters (exactly $752,006$ parameters). Because of the unwieldy size of this network it is also very hard to fit the network to the data since only 316 examples are available in the training set. Hence it is worth also investigating how networks using different types of parameter sharing would work on the problem.

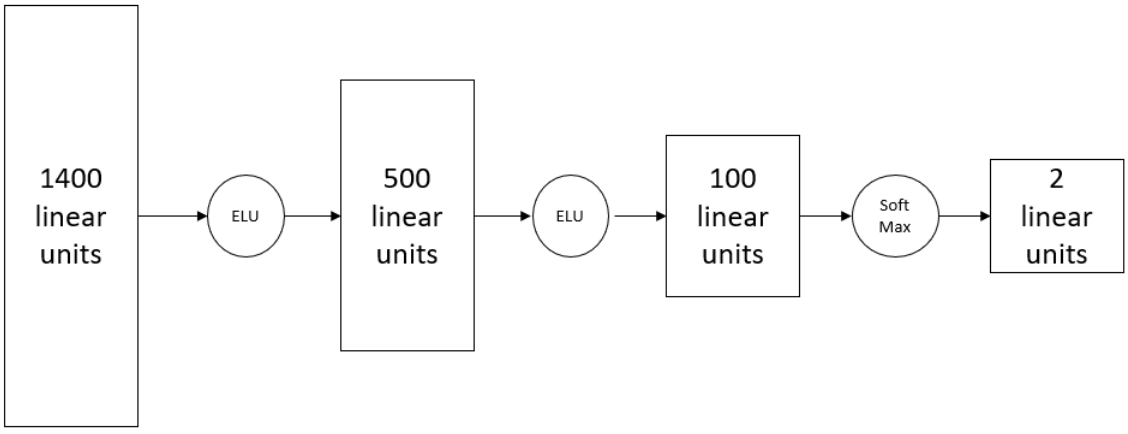

Figure 2: Architecture of the fully connected network.

3.2 Shallow Convolutional Network

As discussed, the fully connected model as an extensive amount of parameters, hence making it very hard to fit this network to the small data set given. A solution to this problem is to introduce the concept of parameter sharing as is the case in convolutional networks. By interpreting the EEG recordings as a 2-dimensional time series, each of the 28 channels with 50 time points, the input size of the data set is $(28, 50)$ and hence a two-dimensional convolutional network can be applied

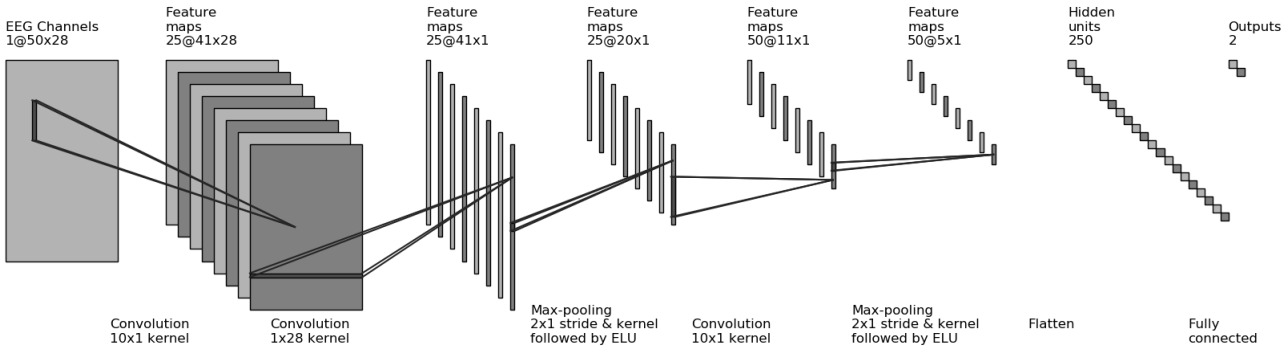

Figure 3: Architecture of the shallow convolutional neural network. This figure was generated by adapting the code from `https://github.com/gwding/draw_convnet`.

to the data. In two-dimensional convolutional networks, at each layer a kernel u of some size (w, h) performs a convolution across the input x of size (W, H) using the convolutional operator, which is the sum of the point-wise product between the kernel and the part of the input being convolved. Without padding of the input this decreases the output sample size y to size $(W-w+1, H-h+1)$. Naturally, more than one convolutional kernel is applied at each layer, hence increasing the amount of different information extracted. Further, to reduce spatial invariance in the predictions, a pooling operator can be applied. An appropriate pooling operator for this problem, and many others, is the max-pooling operator. This similarly works by applying a kernel across the input and returning the maximum value for each input covered by the kernel. Combining this with the aforementioned ELU activation function, the network is constructed as seen in Figure 3. In the first two layers, two different convolutions are applied specifically designed for the problem at hand. First, a convolution is applied to extract temporal information from neighboring time points in a single channel followed by a convolution across all 28 channels to decrease the field of view to only a single channel. This design draws inspiration from [3], which notes that the first two layers in principle, could be combined into a single kernel as no non-linearity exist between these, but we find that the training process is greatly enhanced by explicitly splitting these operations into two. The block is then followed by an ELU activation combined with a max-pooling operation. Then another convolution is applied to increase the amount of filters to 50 followed by ELU and max-pooling. Eventually the input is flattened and a conventional fully connected network is used to output the probabilities of either class. During training batch-normalization and dropout with $p = 0.75$ is applied respectively before and after every ELU activation. This network has a total of $31,002$ hence approximately 24 time less parameters than in the fully connected case, which hopefully will make training the network easier.

3.3 Deep Convolutional Network

It is further of interest to investigate how a deep network performs on the data set. This is because it can be shown that the task of approximating a function f is done more easily by increasing the depth of a network than increasing the width of one.[1] Hence it might improve network performance if the amount of layers are increased and the amount of filters used in each layer likewise. As the convolutional networks use weight sharing, increasing the depth of a network does not increase the amount of parameters as fast as for the fully connected case. By keeping the same structure as the shallow convolutional network seen in Figure 3, a deeper convolutional network is constructed as seen in Figure 4. The main difference in this network is that at each convolutional layer, the kernel size is reduced across the time series, hence the temporal information extracted at each

[1] Lecture notes "Going Deeper" by François Fleuret, EPFL, IDIAP

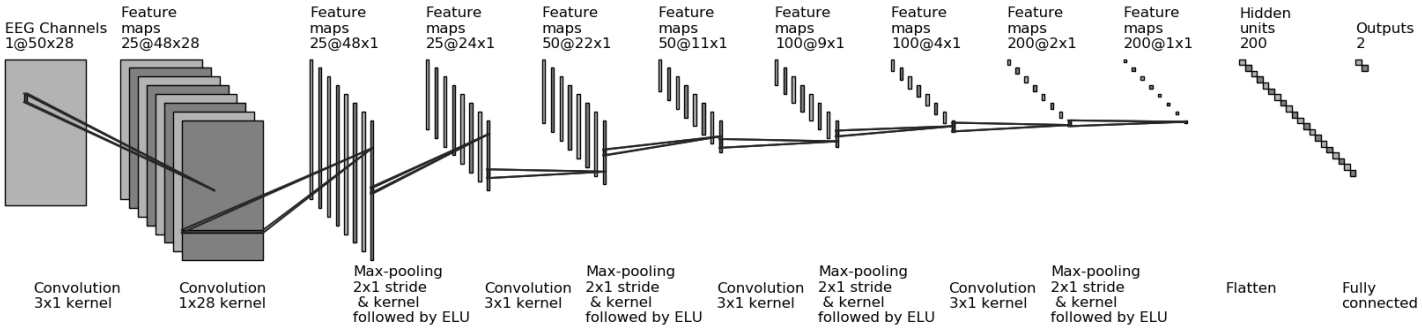

Figure 4: Architecture of the deep convolutional neural network. This figure was generated by adapting the code from `https://github.com/gwding/draw_convnet`.

layer is reduced. Further, by adding two additional convolutional layers and gradually increasing the amount of filters in each, the total amount of filters after the fifth convolutional layer is 200. Hereafter, a fully connected network outputs the probability of each class. Likewise as in the shallow convolutional network, ELU and max-pooling is used after each of the layers after the second convolutional layer. Similarly during training, batch-normalization and dropout with $p = 0.75$ is applied respectively before and after every ELU activation. This network has a total of $97,877$ parameters and hence has approximately 3 times more parameters than the shallow network.

4 Results and discussion

Now, having introduced the basic structures of the three networks constructed, an overview of the results of each of these is given.
All networks were optimized with the Adam optimizer [4] and with mini-batches of size 50 for 2000 epochs. The learning rate is initially set to $1e-3$ and then gradually reduced by a factor 10 upon different milestones, i.e. at epoch number 1000, 1500 and 1750. Theoretically, this is needed to converge to a local minimum, as the gradients are stochastic caused from the usage of mini-batches [5]. It was initially seen that the Adam optimizer obtained significantly better and stable results than the stochastic gradient descent (SGD) optimizer and hence going forward it was decided to use this optimizer as an alternative to the simple SGD algorithm. The evolution of the loss and the accuracy on the training and test set is seen in Figure 5 along with the baseline accuracy score of 78% from the logistic regression model presented earlier. Each line represents a pointwise average of 10 runs of each network.

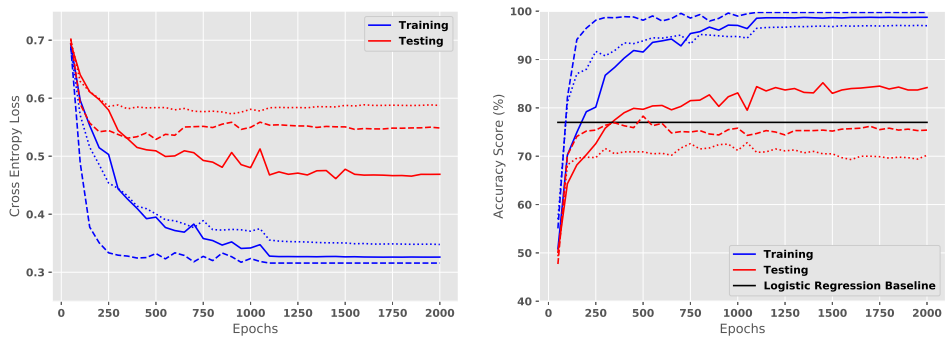

Figure 5: Training and test loss (left) and accuracy (right) of the different models. Fully drawn line is the deep CNN, dashed line is the shallow CNN and dotted line is the fully connected model. Each line represents the pointwise average of 10 runs of each model.

As is clear, only the deep CNN outperforms the baseline logistic regression model. Further, it is seen that the shallow CNN obtains nearly similar accuracy to the logistic regression model and that the fully connected network obtains approximately 70% accuracy. The mean accuracy of the at the last epoch of each of the networks, along with their standard error of the mean, is presented in Table A.

Table A: Mean accuracy at the last epoch ± the standard error of the mean for each of the networks presented. Each model was run 10 times to get an estimate of the standard error of the mean.

	Fully connected model	Shallow CNN	Deep CNN
Mean accuracy	70.20%	75.40%	84.20%
Std. error of the mean	0.24%	0.82%	0.72%

From the table it is clear that the standard error of the mean is relatively low for all networks. The standard error of the point-wise estimates of the networks is $\sqrt{10}$ larger and hence close to 3 % for the CNN's. Hence the actual deviations from the mean can be larger than 3 %, showing how difficult the task of optimizing the network is when moving in an extremely high-dimensional space. Further, it should be noted that techniques like early-stopping probably would lead to higher test accuracy. We choose not to consider this strategy in this work, to avoid overfitting to the test set. However, if more data would have been available, creating a validation set to do early stopping upon, would have been appropriate to avoid this, but given the low amount of data we choose to simply train all networks for 2000 epochs and reducing the learning rate appropriately, assuming that all have converged to some local minima.

5 Conclusions

In this paper various classification techniques for EEG signals are tested and compared to a logistic regression baseline model. The baseline model achieved a 78% accuracy. We first test a simple fully connected network with two hidden layers. This model has an extensive amount of parameters totalling $752,006$ parameters and as was seen, it was extremely difficult to fit this network to the data in a way that it would generalize well to the test set. This lead to a more reasonable approach using convolutional networks and hence introducing weight sharing such that both the total number of parameters were reduced but also making the networks generalize better.

Only the deep convolutional architecture is shown to outperform that baseline which surprisingly obtains a quite high accuracy on the test set. This is probably a cause of the fact that fitting the logistic regression model to the data is a much easier optimization task than for the neural networks. In fitting the convolutional networks, we use popular techniques such as batch-normalization to stabilize and facilitate the training accommodated by advanced SGD approaches such as the adaptive Adam algorithm. Further to avoid overfitting we use dropout extensively during training with success.

It has generally been shown in other research that convolutional networks generalize well when the size of the training set is increased and hence as the training set is small - only 316 examples - better results can be expected with a larger sample size to learn from.

References

[1] B. Blankertz, G. Curio, and K.-R. Müller, "Classifying single trial eeg: Towards brain computer interfacing", in *Advances in neural information processing systems*, 2002, pp. 157–164.

[2] D. M. Goldenholz, S. P. Ahlfors, M. S. Hämäläinen, D. Sharon, M. Ishitobi, L. M. Vaina, and S. M. Stufflebeam, "Mapping the signal-to-noise-ratios of cortical sources in magnetoencephalography and electroencephalography", *Human brain mapping*, vol. 30, no. 4, pp. 1077–1086, 2009.

[3] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, "Deep learning with convolutional neural networks for eeg decoding and visualization", *Human brain mapping*, vol. 38, no. 11, pp. 5391–5420, 2017.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.