Improving Deep Learning models for EMG decoding used for prosthesis control enhancement

Lawrence Brun, Florian Genilloud, Dario Müller
Department of Computer Science, EPF Lausanne, Switzerland

***Abstract*—Introducing a genetic algorithm to an existing deep net [2] resulted in insightful findings on hyperparameter choice for this specific application. When using raw data in the time domain, validation performance was around 4% (Chapter III-A for scoring metrics) higher than a standard method with manual feature extraction. Furthermore, it was shown that using the Fast Fourier Transform to generate data points in the frequency domain did not improve the best loss, yet performs with more stability in cross-validation tests.**

I. Introduction

With the advancement of robotic capabilities, amputees are able to deploy prosthetic hands to regain lost capabilities. By placing EMG electrodes on muscles that record remaining activity when the patient thinks of moving his missing (phantom) hand. The result is sought to replicate natural movements as close as possible. With increasing numbers of electrodes (6-12), machine learning algorithms have the potential to replace this simple control approach by extracting patterns from muscle activity to perform grasp classification for example. With the advent of deep neural networks, feature engineering loses its importance to feature learning using raw data as input. It has been demonstrated that this deep learning approach offers better performance than hand-crafted features for grasp classification and simultaneous single-finger and wrist movement classification. The aim of this project is to deploy hyperparameter tuning using a genetic algorithm approach applied on three different Deep networks, a standard net with manual feature extraction, a deep net resembling the "Ameri-like model" used in [2], and a last one similar to the second but running on the Fast Fourier Transform (FFT) of the data points, and compare them.

II. Experimental setup

The task to be performed can be described as a multiple regression: Six electromyographs on able-bodied people (because ground truth is required, hand has to be present for training) deliver information on muscle activity at a sampling rate of 2000 Hz after the person tries to imitate a motion shown on a screen. The ground truth (labels) are six finger angles, recorded in real time. The goal is to regress these angles based on the EMG recordings so that this can be deployed on a disabled person to control a robotic hand. Figures 6 and 7 show a training session, where the same hand gesture was performed 5 times.

Three of those repetitions were used to train the model on, one was used for testing and the last one for validation.

A. Data extraction

First, the ground truth labels and the EMG signals needed to be aligned to account for differing data sampling rate [2]. To this extent 384 EMG data points (0.192 s of window length times 2000 Hz) were windowed and assigned to one angle (label). In order to maximize the amount of data to regress, the window was shifted by 10 ms and recalculated.

B. Models

a) Standard method: The standard method is a deep net trained on 90 manually devised scalar features on each window, such as number of zero crossings of the signal, max amplitude, standard deviation etc. This method was provided and is a reference other findings in this report.

b) Time windows (Ameri-like[2]): The deep net architecture is pretty standard and can be seen in (Fig. 4). after each convolutional layer, either group normalization or batch normalization were added, and at the end of each block of convolutions we apply average pooling. The number of blocks, layers per block, normalisation method, filter dimensions, channels, pooling dimensions etc. are all hyperparameters to be tuned.

In input, the dimensions of 384x6 (samples x channels) of the windows were reshaped to 48x48 by cutting them in pieces in time and stacking each piece of (6 channels x time) on top of each other. This was done for a few reasons: First, deep nets are commonly applied to image processing tasks in which images are usually square shaped. It also allows our filters to sample correlations between distant times. It also allows closer proximity between channels 1 and 6. Lastly it enables more pooling in the channels dimension. In order to account for the lost layer due to the convolutions, padding was employed which implies that edge layers were symmetrically mirrored to avoid underrepresentation of data on the edges.

c) Fourier: The last method is the same as the Time windows method, but the FFT was applied to the raw signals on each channel and for each window.

C. Pooling

Since the EMG data is noisy (6), pooling based on averaging was introduced to downsample the data with the prospect of yielding in better results.

D. Scrambling

Another idea was to scramble the order of the EMG channels in the windows (not the finger channels, however) to avoid overfitting and to account for the possibility that there is a symmetry in which the order of the channels do not matter in guessing the labels.

E. Randomized sampling

To avoid picking up on time correlations between windows (because until now they are all in the order of the sequence imitated by the subject) the windows were randomly reordered with no more time linearity. When performing gradient descent this might yield better results as we are not focusing on one type of finger position for many batches.

F. Genetic Algorithm

In order to find the best fitting model, a genetic algorithm (available in the GeneAI library) was introduced. Compared to traditional Grid search algorithms, they can be applied to problems with unknown mathematical expressions underlying them [1] while being able to avoid computationally expensive grid searches (for 20 hyperparameters taking each 4 values the grid search would be infeasible 4^{20} combinations). The algorithm tries to mimic nature's way of optimizing life forms by mutations. To this extent, a population of size 20 with 21 genes each (representing variations of hyperparameter compositions) was used for training. One corresponds to training all the data points exactly once. To avoid unnecessary computations if the fitness would not increase, early stopping was introduced with parameters patience = 13 and a minimum delta of 0.05. This indicates that if the loss after 1 epoch did not increase by more than 0.05, a new set of hyperparameters (chromosome) was introduced and tested on. After the training of this generation, the "fittest" 60% were selected and a mutation rate of 10% introduced to ensure avoiding being stuck in local minima. A more detailed description can be found in [1]. As an expression of the loss, this method employs the term fitness which is also sought to be maximized.

G. Technical implementation

The algorithm was run on a Nvidia RTX 2070 (Laptop version) and needed 24 hours to to compute 60 generations for just one dataset. Hence, only one session was modelled and fitted due to time constraints.

III. Results

A. Scoring metric

In order to provide a meaningful loss, a predictor was introduced that randomly predicts an angle between 0 and 10 from a uniform distribution and for each finger (label), the mean square loss was calculated with respect to the validation (or test) labels. We will consider the square root of our losses to bring the pseudo unit of $angle^2$ back to an angle (as we are considering mean square error). $loss_{naive}$ was denoted as the square root of the previous mean square error. We now attribute a score from 0 to 100% for our validation data set defined as:

$$score = 1 - \frac{loss}{loss_{naive}}$$

where loss is the square root of the mean square error for our validation set (or any other set). If our $loss = loss_{naive}$ then score is 0%, if $loss = 0$ then score=100% we now have a more meaningful grasp of how well we are performing, naively or better. Note that on our validation set we get $loss_{naive} \sim$ 5.6 this value hardly changes over different random guesses of the validation labels.

B. Performance analysis

Considering the validation losses in Figure 1, it is evident that the model after employing the Fast Fourier Transform scores worse than the ones using raw data, both the model used in [2] and in this study. The latter also show significant improvement in first 10 generations, afterwards validation scores start to plateau. However, it has to be stated that the oscillations while training subsequent epochs were less prominent for this method. When taking into account the score development with respect to the training (top three plot lines), a big discrepancy in initial values as well as volatility over generations can be observed.

TABLE I: Average and standard deviation of performance for each method used, for each 30 epochs and 60 epochs

	30 epochs		***60 epochs***	
Score	**Avg**	**Std**	**Avg**	**Std**
Time	42.99	16.83	48.71	3.57
Fourier	45.41	5.95	49.00	1.49

C. Hyperparameter analysis

Moving on to hyperparameter tuning, Table II in the Annex shows the variety of hyperparameters (genes) used to create different chromosomes. After training the model for several hours, the optimal values were found for the best performing one and are hereby listed in bold:

Out of all the values for lambda_ tested, the greatest one (1e-5) was found to yield the best results. Interestingly enough, the top left plot Figure 3 shows that the chosen lambda only started to assume higher values in the last few generations, implying that a higher regularization term was needed with advancing generation, possibly to prevent overfitting phenomena from arising. Since the validation score stopped increasing within the second half of generations (30-60), the significance of these findings are questionable. The droprate on the other hand, for instance, consistently assumed the lowest possible value of 0.1. It can thus be inferred that removing random links between layers cannot be considered beneficial. In order to include a metric to control the amount of layers of the convolutional network, different hyperparameters for each block were introduced. The optimal model employs very differing values ranging from 1-3. Moreover, the according plots in Figure 3 do not prove to be very insightful either. Conversely, the number of filters stands for the amount of features that were chosen to train the model. For almost all blocks, the maximum value 32 was assumed in the best outcome, suggesting that the more features created, the better. Using this many filters is luckily enough made possible by parallelized GPU operation, so these operations can be easily reproduced on other systems too. The gamma value represents the initial training rate. It was shown that a small gamma

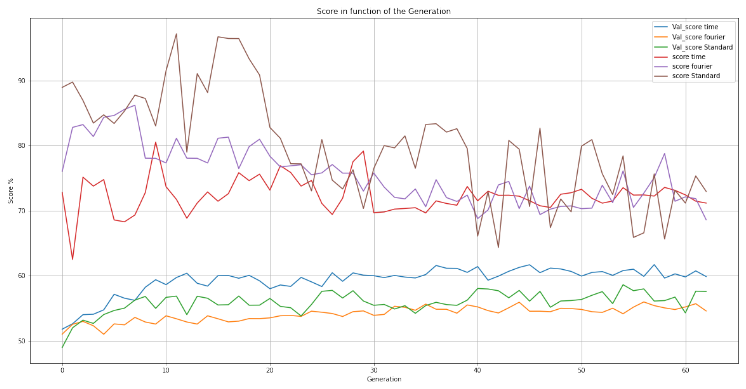

Fig. 1: Evolution of the Training losses (red, violet and brown) and the validation loss (blue, green and yellow) of the models using data in time and frequency domain compared to the standard method introduced in [2].

size should be preferred. It should also be noted that the gamma was continuously reduced after a few epochs in order to continue oscillating too much by overly adapting the weights. In order to provide a sensible grasp of the scoring, Figure 5 was introduced.

The predictions in orange correspond to a supposedly low score of 52%. However, if the regression task were to be changed to a classification problem, predictions would improve since the classification occurs in the right windows.

IV. Discussion

Since the movements in one session are always similar (see Fig. 7), the model is tested on the same repetitions of movements. Even though the EMG data points are different (Fig. 6), this is likely to yield overly optimistic losses, meaning the model would perform a lot worse if different movements were to be predicted. The evaluation of scrambling resulting in worse results indicates that experiments should focus on electrode placing and conservation when creating the input data. A strange phenomena can be observed when comparing training and validation score evolution in Figure 1. With the validation score remaining constant and the train score decreasing with advancing generations, the conclusion can be drawn that the training loss is actually increasing, but when testing it on other data points the model performs equally well. Further research could therefore focus more on intersession analysis (running the model on different people's EMG data) rather than on data collected from an intrasession. Furthermore, introducing scrambling resulted in higher losses, meaning that the spatial information is crucial to the model's accuracy. Lastly, a smaller batch size could improve the model's accuracy at the trade-off of computation time which was the limiting factor for this study.

V. Summary

By tuning hyperparameters, we achieved a score of over ~ 60 % on validation set with the Ameri-like model surpassing the standard method [2]. Even though Fourier renders lower scores than the time Ameri-like, it gets better cross validation standard deviation and seems more stable in the gradient descent. The net could probably be improved but preprocessing of the data might result in better predictions overall.

VI. Annex

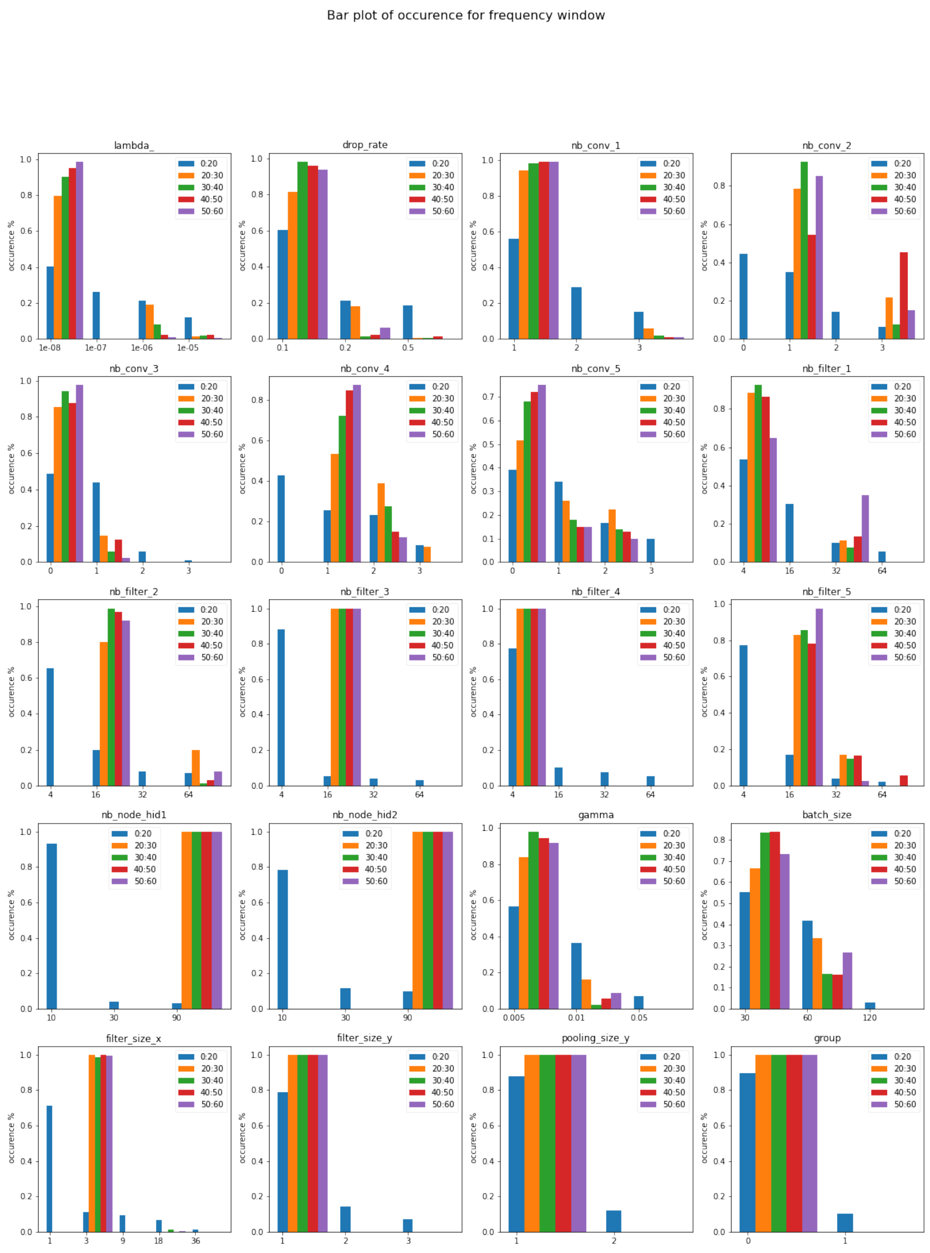

Fig. 2: Hyperparameter (x-axis) evolution for different generation brackets in the genetic algorithm (colored bins) for the model after applying the Fast Fourier Transform

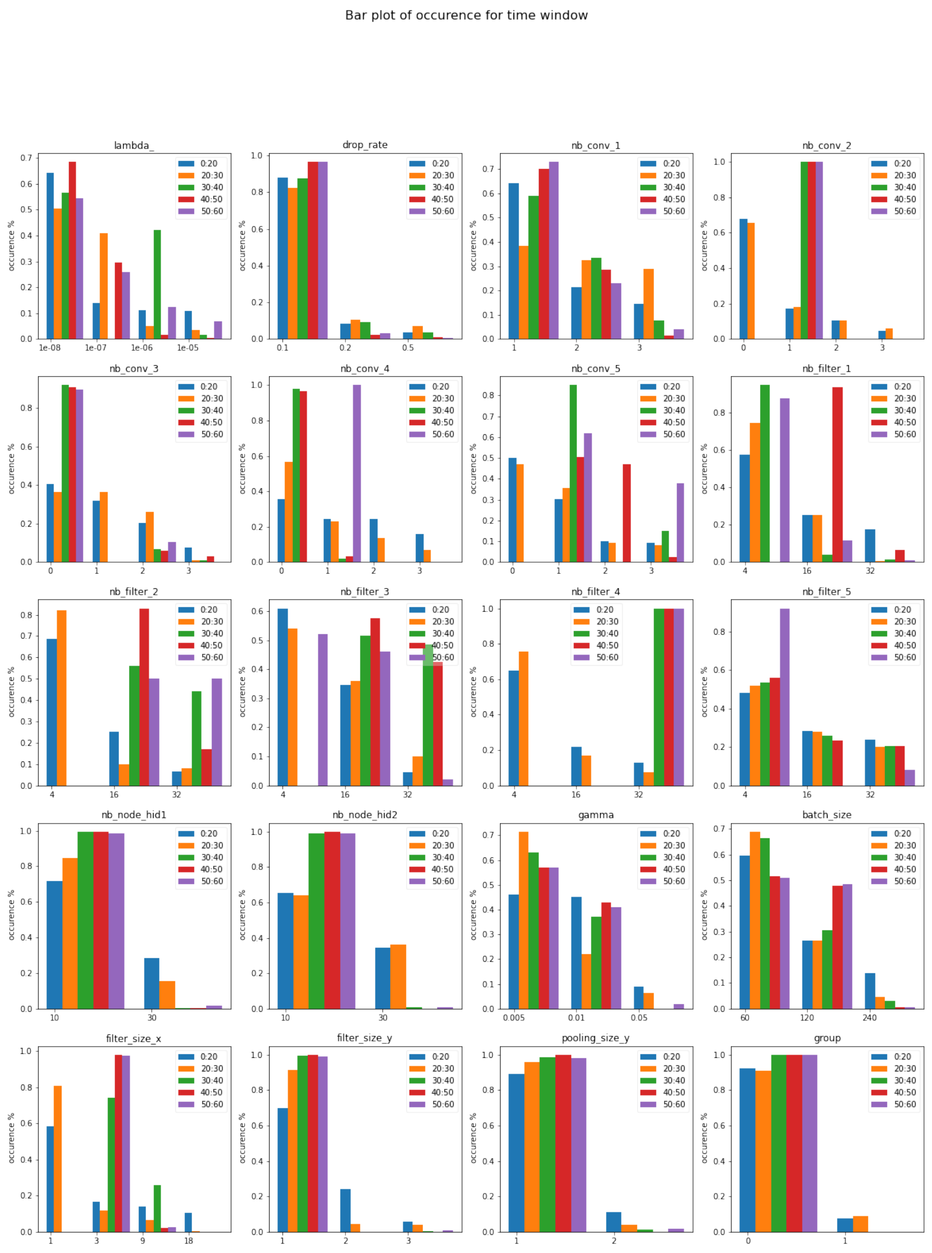

Fig. 3: Hyperparameter (x-axis) evolution for different generation brackets in the genetic algorithm (colored bins) for the model using raw data

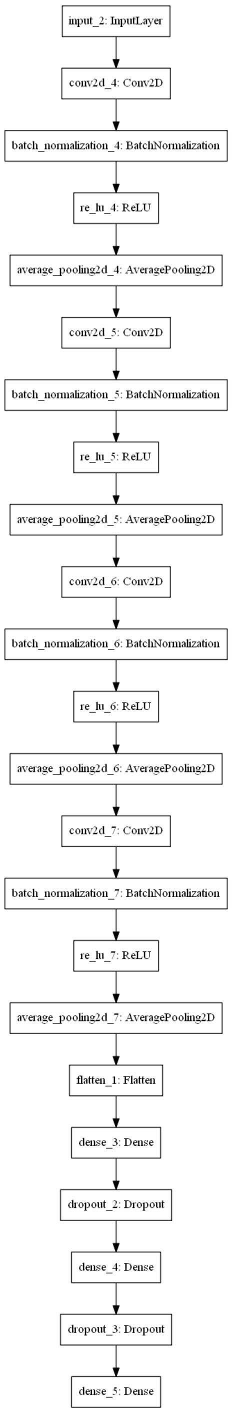
input_2: InputLayer
conv2d_4: Conv2D
batch_normalization_4: BatchNormalization
re_lu_4: ReLU
average_pooling2d_4: AveragePooling2D
conv2d_5: Conv2D
batch_normalization_5: BatchNormalization
re_lu_5: ReLU
average_pooling2d_5: AveragePooling2D
conv2d_6: Conv2D
batch_normalization_6: BatchNormalization
re_lu_6: ReLU
average_pooling2d_6: AveragePooling2D
conv2d_7: Conv2D
batch_normalization_7: BatchNormalization
re_lu_7: ReLU
average_pooling2d_7: AveragePooling2D
flatten_1: Flatten
dense_3: Dense
dropout_2: Dropout
dense_4: Dense
dropout_3: Dropout
dense_5: Dense

Fig. 4: Deep net architecture

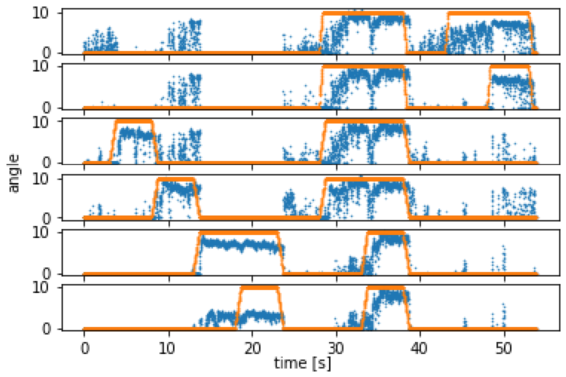

Fig. 5: Visualization of the predictions by the time domain model (orange) and EMG data (blue). For referencing: This prediction corresponds to a score of 52%

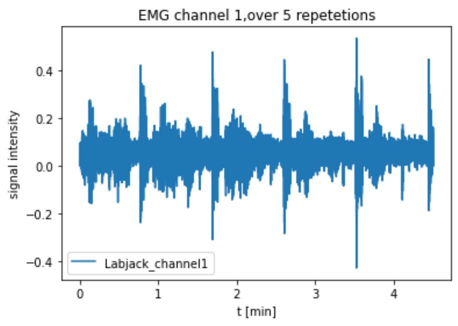

Fig. 6: Sample visualization of data from electromyographs

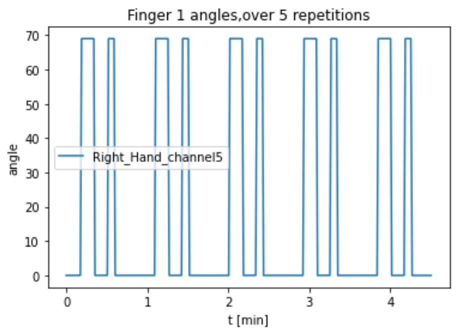

Fig. 7: Sample visualization of the corresponding ground truth to the signals observed in Figure 6

TABLE II: Hyperparameters used on different genes on which the algorithm iterated. **Optimal hyperparameters** found are listed in bold script.

Parameter	Values			
lambda_	**1e-5**	1e-6	1e-7	1e-8
drop_rate	**0.1**	0.2	0.5	
nb_conv_1	0	**1**	2	3
nb_conv_2	0	**1**	2	3
nb_conv_3	0	1	**2**	3
nb_conv_4	0	**1**	2	3
nb_conv_5	0	1	2	**3**
nb_filter_1	4	16	**32**	
nb_filter_2	4	16	**32**	
nb_filter_3	4	**16**	32	
nb_filter_4	4	16	**32**	
nb_filter_5	4	16	**32**	
nb_node_hid1	10	20	**30**	
nb_node_hid2	**10**	20	30	
gamma	0.05	0.01	**0.005**	
batch_size	60	**120**	240	
filter_size_x	1	3	**9**	18
filter_size_y	**1**	2	3	
pooling_size_y	**1**	2		
group	True	**False**		
group_size	2	**4**	8	

References

[1] Diogo Matos Chaves. *Introducing GeneAl: a Genetic Algorithm Python Library*. URL: https://towardsdatascience.com/introducing-geneal-a-genetic-algorithm-python-library-db69abfc212c (visited on 12/12/2020).

[2] Leonardo Pollina. *Exploratory research of Deep Learning models for EMG decoding*. 2020-12-12.