Design of Neural Decoder for Planar Reaching Movements

Dept. of Bioengineering, Imperial College London, London, UK

***Abstract*—The following paper outlines a trajectory estimator neural decoder, which was trained and evaluated on spike recordings of a monkey's brain while it was performing reaching movements in eight different directions. The developed algorithm performs data pre-processing and classification of neural signals corresponding to the planning phase, before the reaching movement is executed, to estimate the movement direction of the arm. Once movement direction is classified, regression is performed with a Binary Decision Tree to continuously estimate the position of the monkey's hand and post-processing to increase the regression accuracy. The methods in this paper focus on both the accuracy and the computing time of the algorithm. The accuracy of the reaching angle classification was 99.3%, and the Root Mean Square Error (RMSE) of the decoder was 8.99, with a run time of 17.18 seconds.**

***Index Terms*—neural decoder, brain machine interface, classification, regression, binary decision tree, principal component analysis**

I. Introduction

The idea of neural decoders holds a huge potential for paralyzed patients around the world. The motor (M1) and premotor areas of the cortex in the brain produce neural signals that are responsible for planning and controlling voluntary movements. When neural signals from these regions were recorded, it was found that there is a series of characteristic pulses for each task the subject was performing. It was found that these patterns encode the speed and direction of the movement [3], as well as the location of the target the subject was trying to reach [2]. The truly interesting discovery was that these areas of the brain are still active, even if the paths that deliver these signals to the muscles are damaged, such as the spinal cord. This realisation gave birth to the idea of controlling prosthetic devices through brain-machine interfaces. Such devices would record these neural muscle control signals, decode them and then control artificial limbs, allowing patients to move again.

Neural decoders for prosthetic control can be divided into two main categories, target location estimators and trajectory estimators. Target location estimating algorithms usually rely on probabilistic approaches such as Bayesian models or maximum likelihood methods. Their goal is to output an estimated location for the target of the motion and rely on neural data from the planning phase, which lasts from the first neural activity until the first movement cue [6]. These methods are especially useful when the patient wishes to pick between a set of possible movement locations. Trajectory estimation algorithms have a goal of predicting the continuous motion of the limb and use the data generated by the neurons during movement. They usually rely on linear filters or population vectors, but the use of such systems will always be an approximation due to the nonlinear nature of the data [5].

This paper outlines a neural decoder developed to decode neural signals captured in a monkey's brain to predict its hand's movement, exploring methods that focus on training and decoding speed as well as accuracy.

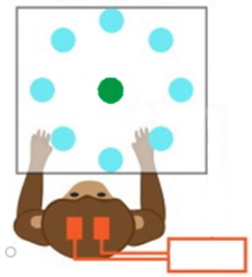

Fig. 1. Reaching tasks were completed on a fronto-parallel screen, where the green dot represents the starting location and the blue ones depicts the eight possible endpoints of the movement [4]. Neural activity and arm trajectory was recorded from 300ms before movement onset to 100ms after movement end.

II. Methodology

A. Data and Pre-processing

The data set provided consists of spike train recordings of 98 neural units and three-dimensional arm trajectory of the monkey for 8 reaching angles (30/180π, 70/180π, 110/180π, 150/180π, 190/180π, 230/180π, 310/180π, 350/180π). There are 182 trials per reaching angle (8 reaching angles); 100 of which were provided to train and validate our model, while the remaining 82 were inaccessible and used to evaluate the algorithm's performance on unseen data. The data set of 800 trials (100 x 8) were separated into 640-160 train-test splits, where training and cross-validation was performed on the 640 trials, and then testing was done on the remaining 160 unseen trials. For the training set, we removed trials that deviated from the other trajectories. This was done by calculating the median trajectory for each angle and removing trials that deviated too far from this value. 75 out of 640 trials were removed through this process. Figure 2 provides an example of the removed deviated trajectories. Neural units 38, 49 and 76 were omitted from the data set as they were identified to be inactive. Inactivity was defined as having a low number of spikes across

all trials relative to all 98 units. Low activity throughout all trials implied that these neural units had little to no correlation with any of the movements so they act as unhelpful noise for the model.

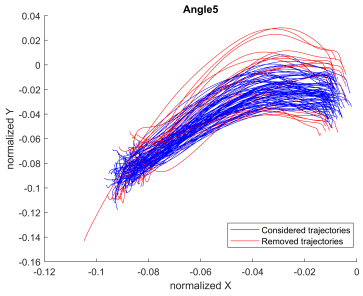

Fig. 2. Trajectories from reaching angle 5 that were removed and considered for model training.

The features from the spiking data are computed using a windowing method which computes the firing rates of each neuron by summing the spikes over a set window size. Starting from the very beginning of the trial segment, the first window captures the firing rate over 168 milliseconds and the second window captures the firing rates right after the end of the first window over a 70 millisecond period. These windows are computed every 20 milliseconds until the end of each trial. The windows also take into account the delay between the neural firing and hand movement. A delay of 165ms was found to give the best results.

The windowing process yields a data set that consists of 190 dimensions (95 neuron firing rates x 2 windows). Considering the fact that we use two windows and 70% of the neurons are multi-neuron units, many of the neurons will have multicollinearity, which reduces the performance of Machine Learning (ML) models. To reduce the dimensionality, we perform Principal Component Analysis (PCA). First we remove linearly dependent columns as this is shown to increase the PCA's performance. This lowers the dimension from 190 to 83. We perform PCA on that data with 94% explanation threshold and further reduce the dimensionality down to 57. This dimensionality reduced data is used to train regression models as explained in section C. Regression.

B. Classification

Since there are 8 distinct reaching angles in the data set, our approach was to first classify the reaching angle, and then perform regression on the predicted angle. This allows to build accurate specialized regression models for each reaching angle. The classification was done using the first 400 milliseconds of each trial. During training a window of size 90 sliding over the trials with a step-size of 18 was used. These window parameters gave the best results empirically, based on our experiments with various randomized train-test splits. Spike occurrences in each window were summed into a single 1x95 vector, where each value in the vector corresponds to the firing count of a particular neuron over the windowed period. Firing rates were calculated with this method over all training trials and then averaged and labeled for each of the 8 angles. During testing when encountering new unseen neural activity, the algorithm calculates the spike count over the same window size as in training, and then compares that spike vector against all 8 mean spike vectors for each corresponding window segment. The absolute difference against each angle was summed into a scalar value and added to a final 1x8 vector. These differences were added up over all segments so that at the end of this process, the 1x8 vector contains the total differences for each corresponding angle. The unknown angle was classified by picking the index of the lowest value (lowest distance) in the array as our classified angle.

C. Regression

As explained in the previous section, the classification approach was used to create specialised and more accurate regression models for each angle. 16 models were trained to predict movement from the processed spike train data, one for each movement dimension, x and y, for each one of the 8 angles. With 16 individual models, the complexity of each regression model is reduced. By breaking down the task to a series of simple linear regression problems, the decision was made to focus on evaluating the regression models that were simple and computationally inexpensive. A preliminary investigation was conducted to compare the performance of five different regression models. The models evaluated were: 1. Linear least squares regression; 2. Binary Decision Trees; 3. Bagged Decision Trees; 4. Support Vector Machine (SVM) with a linear kernel function; 5. Gaussian Process with an exponential kernel function. Model performance was based on time taken to train the model and prediction accuracy on the test data set. The Binary Decision Tree method was identified as the overall best performing model and was optimised by performing a grid search to tune the minimum leaf size of each of the 16 models to minimise the prediction Root Mean Square Error (RMSE).

All of the models for both classification and regression were validated with 5-fold cross-validation to find a model with the best predictive performance on unseen data.

D. Post-processing

The goal of the post processing was to improve the estimation of the monkey's hand which is obtained from the regressor. To achieve this, the mean position of each 20 ms window for every angle is computed during the training and it is used for further correction of the position. Diverse strategies have been investigated and the difference between them relies mainly in the weight of the experience during training, represented by the mean position for each time step and angle, x_{mean} and the one of the position estimations given by the regressor (x_{pred}).

The first strategy consists of just considering the best prediction (x_{final}) and the mean position for each time step and angle without considering the actual prediction of our regressor.

The second strategy consists of giving the same weight to the current estimation of the regressor and the mean position for that time step.

The third strategy relies on the weighted and tuned sum of each one of the two terms. The weight (θ) which each term has is determined by tuning through grid search and its weight mainly depends on the precision of the regressor.

$$x_{final} = \theta * x_{mean} + (1 - \theta) * x_{pred}; \quad (1)$$

To know which one of these methods was the best, their results are compared between them and the value obtained without any post-processing.

III. Results

Unless stated otherwise, all the following results have been obtained from only the validation data set.

A. Classification

The classification accuracies of our models are shown in table 1. As seen in the table, the four classifiers that was trained using the Matlab's Classification Learner App all achieved lower accuracies than the custom classifier that we built from scratch, with the best performing model being Linear Discriminant achieving 98.7% accuracy. Our custom classifier achieved 99.3% accuracy, which means that in the 640-160 train-test split it correctly predicted all angles except for one.

Classifier Type	Accuracy
Fine Tree	96.2%
Fine KNN	97.5%
Linear SVM	97.5%
Linear Discriminant	98.7%
Custom Classifier	**99.3%**

TABLE I
COMPARISON OF CLASSIFIER ACCURACIES

B. Regression

The prediction RMSE and Training computation time for each model from the preliminary test and the final optimised model is shown in Table II. From the preliminary test, models based on decision trees had the lowest RMSE while linear and binary decision tree training time was significantly faster than the other models that were tested. After optimisation, the Binary Decision model outperformed all the other tested models in both RMSE and computation time.

Regression Model	RMSE	Computation Time (s)
SVM	10.76	2.80
Linear	10.73	0.303
Gaussian process	10.38	71.9
Binary Decision Tree	8.63	0.314
Bagged Decision Trees	8.39	2.14
Binary Decision Tree (Optimised)	**8.15**	**0.282**

TABLE II
COMPARISON OF REGRESSION MODELS

C. Post-processing

The RMSEs for each one of the tried post processing strategies described in section X as well as the obtained RMSE without any post processing are seen in table III.

Strategies	RMSE
No post-processing	8.15
Strategy one: $x_{final} = x_{mean}$	6.94
Strategy two: $x_{final}=(x_{mean}+x_{pred})/2$	6.43
Strategy three: $xfinal=0.64*x_{mean}+0.36*x_{pred}$	6.34

TABLE III
RESULTS OF THE DIFFERENT POST-PROCESSING STRATEGIES

As shown in the table, all the post-processing strategies decreases the error. Among them, the third strategy which consists on weighted and tuned sum of x_{mean} and x_{pred} is the one with the lowest error and the one which is included in the final algorithm. The tuning process of the third strategy has led to the following weighted sum:

$$x_{final} = 0.64 * x_{mean} + 0.36 * x_{pred}; \quad (2)$$

D. Final Model

The best performing methods for classification, regression and post-processing were combined to design the neural decoder. Figure 3 displays the performance of the neural decoder. The low deviation in the decoded trajectories can be attributed to the weighing of the mean trajectory across trials from the post-processing stage. The RMSE of this model on the validation data set was 6.33. The model performed slightly worse on the evaluation data set with an RMSE of 8.99.

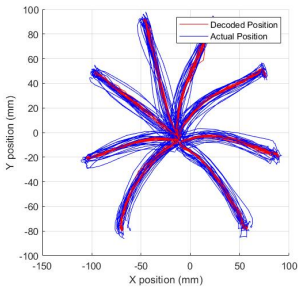

Fig. 3. Comparison between the actual and decoded trajectories predicted by the neural decoder.

IV. Conclusion

From the results, it can be concluded that the developed algorithm performs well and fits the tasks specification and it shows that it is feasible to decode the hand's movement from neural activity. The developed methods follow a logical and intuitive flow based on angle classification and specific regression training for each one of these angles. This allows it to achieve a good performance on the validation set without the need for high training times.

Regarding the classification task, instead of relying on Linear Discriminant Analysis (LDA) or other already built

approaches, we built our own classifier extracting the relevant features of the data which enabled the algorithm to achieve a very high classification accuracy on the validation set (99.3%). This capability to extract important features of the data based on our previous knowledge and analysis of it was the reason behind the high performance of our classifier and it is also why it outperforms other already built classifiers which do not have any previous knowledge about the dataset. The approach of training 16 separate regression models was successful in reducing the complexity of the problem, as shown in II where simple and computationally quick methods such as linear regression and decision trees had similar prediction RMSE to more complex methods such as SVM and Gaussian Process. During the preliminary test, Bagged Decision Trees produced a higher prediction accuracy than Binary Decision Trees as expected as the former is designed to reduce over-fitting in Decision Trees by aggregating results of multiple trees. The decision was made to optimise the Binary Decision Tree method, since the 580% faster training time was deemed more important than the 2.8% increase in accuracy. A higher RMSE on the evaluation data set from the neural decoder compared to the validation data set suggests that there was over-fitting of the model to the validation data. In a scenario where the computational cost of training the model is not a factor, optimisation of the Bagged Decision Tree model may have resulted in a better prediction accuracy than the presented final model.

Lastly due to the inaccuracy of the regressor, post processing is performed. The results show that the introduction of the mean position at each time step and angle provides valuable information for the final estimation of the position.However, relying on just these mean values will lead to always predicting the same trajectory for each angle. The estimated position by the regressor should be able to capture the specific behavior of trajectories leading to better generalization. Moreover, the tuned weighted sum gives a better performance than computing mean between the x_{mean} and x_{pred} as it considers the accuracy of the regressor. The final weight of x_{mean} is higher than the one of x_{pred}, thus the major contribution to the final estimation is provided by the mean position during trajectory showing that there is a big gap for improvement for the regressor.

More complex algorithms such as Long Short Term Memory (LSTM), which were our first attempt, were finally discarded as they do not fit the specifications of the competition. According to literature, they have capacity to outperform traditional regressors and would allow the algorithm to achieve a lower error, but they require longer training times [1] which was one of our limitations.

The designed neural decoder is successful in predicting simple reaching motions in predetermined directions with good accuracy. However, a major limitation of this design is its poor scalability. For the model to predict additional reaching angles, 2 additional regression models would need to be trained per angle. The model has also only been tested with planar, linear reaching movements and may not be as effective in decoding more complex movements.

V. Contributions

M. Ilsinski worked on data pre-processing, H. Son worked on the classification model, I. Issak and C. Martin worked on the regression model, and C. Martin worked on post-processing.

VI. Future Work

As mentioned in the conclusion, the current approach has a good overall performance but further improvements can be introduced for better generalization and lower error.

Regarding the classification problem, more features can be considered such as velocity or displacement which can further increased its accuracy. Due to the high relevance of classification in our algorithm and the high error which can be introduced in our system if the angle is misclassified, a general regressor can be built for the cases where there is more uncertainty regarding the classification problem. This regressor may have a higher regression error than each one of the specific ones but as it will only be used in the cases where the angle classification is in doubt it will introduce lower error than using an incorrect regressor in that cases.

Lastly, the regressor is responsible for the majority of the error thus more sophisticated regression algorithms can be used instead of the actual regression tree. For instance, a Kalman Filter can be introduced based on literature, which shows that it can produce an accurate performance in a similar task [7]. Moreover, as we did for the classification problem, further research can be done to find valuable features within the data-set to improve the performance of the regressor algorithm. A more accurate regressor will allow reduction or even removal of the weight of the mean trajectory when computing the hand position, leading to better generalization.

References

[1] Joshua I. Glaser, Ari S. Benjamin, Raeed H. Chowdhury, Matthew G. Perich, Lee E. Miller, and Konrad P. Kording. Machine learning for neural decoding. *eNeuro*, 7(4), 2020.

[2] Julie Messier and John F Kalaska. Covariation of primate dorsal premotor cell activity with direction and amplitude during a memorized-delay reaching task. *Journal of neurophysiology*, 84(1):152–165, 2000.

[3] Daniel W Moran and Andrew B Schwartz. Motor cortical representation of speed and direction during reaching. *Journal of neurophysiology*, 82(5):2676–2692, 1999.

[4] Amy L. Orsborn, Helene G. Moorman, Simon A. Overduin, Maryam M. Shanechi, Dragan F. Dimitrov, and Jose M. Carmena. Closed-loop decoder adaptation shapes neural plasticity for skillful neuroprosthetic control. *Neuron*, 82(6):1380–1393, 2014.

[5] Justin C Sanchez, Sung-Phil Kim, Deniz Erdogmus, Yadunandana N Rao, Jose C Principe, Johan Wessberg, and Miguel Nicolelis. Input-output mapping performance of linear and nonlinear models for estimating hand trajectories from cortical neuronal firing patterns. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 139–148. IEEE, 2002.

[6] Krishna V Shenoy, Daniella Meeker, Shiyan Cao, Sohaib A Kureshi, Bijan Pesaran, Christopher A Buneo, Aaron P Batista, Partha P Mitra, Joel W Burdick, and Richard A Andersen. Neural prosthetic control signals from plan activity. *Neuroreport*, 14(4):591–596, 2003.

[7] W Wu, M. Black, Y. Gao, M. Serruya, A. Shaikhouni, J. Donoghue, and Elie Bienenstock. Neural decoding of cursor motion using a kalman filter. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2003.