Designing a 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 is trained and evaluated on spike recordings of a monkey's brain while it is 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 is 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 is performing. These patterns encode the speed and direction of the movement [3], as well as the location of the target the subject is trying to reach [2]. The truly interesting discovery is 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 (Figure 1).

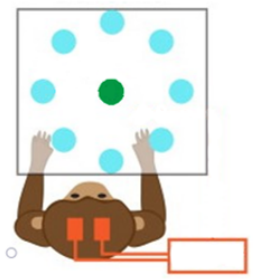

Fig. 1. Reaching tasks were completed on a fronto-parallel screen, where the green dot represents the starting location and blue dots depicts the 8 movement end points [4]. Neural activity and arm trajectory was recorded from 300ms before movement onset to 100ms after movement end.

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 ($30/180\pi$, $70/180\pi$, $110/180\pi$, $150/180\pi$, $190/180\pi$, $230/180\pi$, $310/180\pi$, $350/180\pi$). There are 182 trials per reaching angle (8 reaching angles); 800 (100 x 8) of which are provided to train and validate our model, while the remaining 656 (82 x 8) are inaccessible and used to evaluate the algorithm's performance on unseen data. The data set of 800 trials are separated into 640-160 train-test splits, where training and cross-validation is performed on the 640 trials, and then testing is done on the remaining 160 unseen trials. For the training set, we remove trials that deviate from the other trajectories. This is done by calculating the median trajectory for each angle and removing trials that deviate too far from this value. 75 out of 640 trials are removed through this process. Figure 2 provides an example of the removed deviated trajectories. Neural units 38, 49 and 76 are omitted from the data set as they are identified to be inactive. Inactivity is defined as having a low number of spikes across all trials relative to all 98 units. Low activity throughout all trials implies that these neural units have little to no correlation with any of the movements so they act as unhelpful noise for the model.

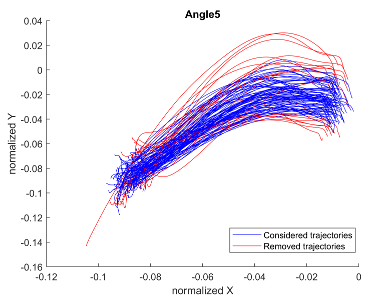

Fig. 2. Example: Trajectories from reaching angle 5 that are removed for model training.

The features from the spiking data for both regression and classification are computed using a windowing method which computes the firing rates of each neuron by summing the spikes over a set window size. For regression, two windows of different sizes are used to slide over the whole trial in 20 millisecond steps. The first window spans across 168 milliseconds while the second window spans across 70 milliseconds. The windows also take into account the delay between the neural firing and hand movement which is about 165 milliseconds. 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 have multicollinearity, which reduces the performance of Machine Learning (ML) models. In order to remove redundant data 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.

The classification setup uses only one window that spans 90 milliseconds which slides over the first 400 milliseconds of the trial with step-size of 18 milliseconds. All of the optimal windowing and delay parameters for regression and classification were found with a grid-search method.

B. Classification

Since there are 8 distinct reaching angles in the data set, our approach is to first classify the reaching angle, and then perform regression on the predicted angle. This allows us to build accurate specialized regression models for each reaching angle. Using the windowing setup explained in the previous section, the spike occurrences in each window are 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 are 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 is summed into a scalar value and added to a final 1x8 vector. These differences are 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 is classified by picking the index of the lowest value (lowest distance) in the array as our classified angle. Performance of the described classifier is compared against standard classification models, which are trained and evaluated with the Classification Learner application using the Matlab's Machine Learning Toolbox.

C. Regression

As explained in the previous section, the classification approach is used to create specialised and more accurate regression models for each angle. 16 models are 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 are simple and computationally inexpensive. A preliminary investigation was conducted to compare the performance of five different regression models. The models evaluated are: 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 is 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 are validated with 5-fold cross-validation to find a model with the best predictive performance on unseen data.

D. Post-processing

The goal of post-processing is to improve the prediction of the monkey's arm position from the regressor. To achieve this, the mean position of each 20 ms window for every angle is computed during training and is used to further increase prediction accuracy. Diverse strategies have been investigated and the difference between them relies mainly on the weight of the experience during training, represented by the mean position for each time step and angle (x_{mean}) and the predicted position given by the regressor (x_{pred}).

The first strategy assumes that the best prediction (x_{final}) is the mean position for each time step and angle and does not consider the prediction of our regressor. The second strategy gives the same weight to the prediction of the regressor and the mean position for that time step. Lastly, the third strategy relies on the weighted and tuned sum of both terms (equation 1). The weights (θ) are determined by tuning through grid search and is dependent on the accuracy of the regressor.

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

The resultant prediction RMSEs from these strategies are compared against the RMSEs without any post-processing to identify the optimal strategy.

III. Results

Unless stated otherwise, all the following results have been obtained from only the validation data set (800 trials).

A. Classification

The classification accuracies of our models are shown in table I. As seen in the table, the four classifiers that are 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 achieves 99.3% accuracy, which means that in the 640-160 train-test split it correctly predicted all angles except for one. All classifiers finished training in less than half a second so computation time is omitted from the analysis.

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 have the lowest RMSE while Linear and Binary Decision Tree training times are significantly faster than the other models that are tested. After optimisation, the Binary Decision model outperforms 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	8.33	2.14
Binary Decision Tree (Optimised)	**8.15**	**0.282**

TABLE II
Comparison of regression models

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}	6.33

TABLE III
Results of the different post-processing strategies

C. Post-processing

The RMSEs for the mentioned post-processing strategies as well as the RMSE without any post-processing are shown in table III.

As shown in table III, all post-processing strategies decrease the error. Among them, the third strategy which consists of the weighted and tuned sum of x_{mean} and x_{pred} produces the lowest RMSE and therefore was included in the final algorithm. The tuning process of the third strategy leads to the following weighted sum:

$$x_{final} = 0.64 * x_{mean} + 0.36 * x_{pred}; \tag{2}$$

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 is 6.33. The model performs slightly worse on the evaluation data set with an RMSE of 8.99.

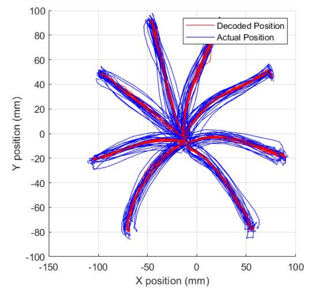

Fig. 3. Comparison between the actual and decoded trajectories predicted by the neural decoder.

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 existing 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 existing classifiers which do not have any previous knowledge about the data set.

The approach of training 16 separate regression models was successful in reducing the complexity of the problem, as shown in Table 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 from multiple trees. The decision was made to optimise the Binary Decision Tree method over the Bagged variation, 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 training data. In a scenario where the computational cost of training the model is less relevant, 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 was 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 shall be considered to be able to capture the specific behavior of trajectories leading to better generalisation. Moreover, the tuned weighted sum gives a better performance than computing the 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 significant room for improvement for the regressor.

More complex algorithms such as Long Short Term Memory (LSTM), were deemed unsuitable as they do not fit the specifications of the decoder design. According to existing literature, they have the capacity to outperform traditional regressors and would allow the algorithm to achieve a lower error, but they require significantly longer training times [1]; which was one of the performance metrics for the decoder.

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. Future Work

As mentioned in the conclusion, the current approach has a good overall performance but further improvements can be introduced for better generalisation 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 specialised models but will only be used in cases where the angle classification is inconclusive and thus will produce a lower error when compared to using an incorrect regressor.

Lastly, the regressor is responsible for the majority of the error so more sophisticated regression algorithms can be used instead of the Binary Decision Tree. For instance, a Kalman Filter can be introduced as existing literature has proved that it can produce an accurate decoder in a similar task [7]. Moreover, as we did for the classification problem, further research can be done to find key features within the data set to improve the performance of the regression 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 generalisation.

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.