No Brain No Gain: Performance of an SVM-Based Neural Decoder on the Motor Cortex of a Macaque

Elena Faillace, Michael Lawrence, Chiara Lazzaroli and Deniss Zerkalijs
Dept. of Bioengineering, Imperial College London, London, UK.
Emails: elena.faillace20@imperial.ac.uk, michael.lawrence16@imperial.ac.uk,
chiara.lazzaroli20@imperial.ac.uk, deniss.zerkalijs20@imperial.ac.uk

***Abstract*—A classifier and the prediction of the x and y coordinates of a monkey's arm trajectory were combined to build a decoder. This was done by using neural activity recorded with 98 electrodes from the cortex of a monkey during the decision-making process for which direction to move its arm and the conduction of said movement. The classifier chosen is support vector machines (SVMs), which is used to determine in which direction the monkey will move its arm. The trajectory is estimated by taking the mean across all trajectories within the training data, provided it is at a distance of less than 7 cm from the testing data. A classification accuracy of 94.05 % was obtained and a root-mean squared error (RMSE) of 14.68 cm with a run time of 7.06 seconds for a 50:50 split of training to testing data. Other methods, such as KNN and linear regression were also tested but proved to be less efficient.**

***Keywords*—neural decoder, classification, regression, brain-machine interface, cortex, macaque monkey, machine learning, support vector machines**

I. Introduction

Neural decoding uses activity recorded from the brain to make predictions about variables in the outside world. Accurately predicting the neuromechanical behaviour from brain signals has always been a challenge for the scientific and engineering communities. *Brain-machine interfaces (BMIs)* specialise in transmitting neuromuscular signals to and from an external robotic device, in order to improve neural and motor rehabilitation for patients with severe neuromuscular disabilities [1, 2]. Machine learning methods are widely used for such non-linear problems as the BMI can learn to associate the complex internal signals with the external variables by analysing gathered real-life data.

Despite the great amount of research done in supervised machine learning methods in the last decade [3, 4], there is no such thing as a universally optimal machine learning algorithm, and each problem requires a unique solution tailored to the problem at hand. For this reason, multiple decoding techniques are usually implemented during decoder development for optimal real-time performance [4]. Typical approaches to neural signal decoding problems include some combination of techniques such as Naive Bayes' classification, k-nearest neighbours classification, Kalman filtering, regression, support vector machines, neural networks, among many others [5].

This study is scoped to design an accurate continuous neural decoder of a macaque monkey for predicting the trajectories of its hand during trained movement. This is achieved by implementing and comparing various decoding algorithms on an existing dataset and ultimately employing the one with least amount of prediction error and processing time.

Developing an accurate model to rapidly decode the motor signals would prove to be beneficial in improving the predictive performance in live neural engineering applications (such as robotic limbs) or our understanding of the relationship between neural activity and the outside world.

II. Methods

The monkey was trained to use its hand to reach towards 8 discrete locations aligned on a 2D plane, referred to also as angles. The position of the hand was recorded along with the neural activity during the movement from 98 electrodes placed on the motor cortex, sampling the signal every 1 ms for a total of $500 - 600$ ms. Each experiment was repeated 100 times.

A. Decoder Structure

The problem can be separated into two distinct supervised learning sections: determining at which angle the monkey will move its arm and predicting the hand's x and y coordinates at any time point given the classified direction.

From all the recordings we only used the first 320 ms as training data set since this is the time taken by the motor cortex to plan the movement before executing it. The motor planning phase is believed to represent a detailed encoding of the muscles that are about to move [6], hence it is sufficient to analyse this phase for the purpose of angle classification. Fig. 1 is representative of what the 4th trajectory looks like (i.e. at $150°$), separated into x- and y-components. The hand movement begins after 320 ms in all trajectories.

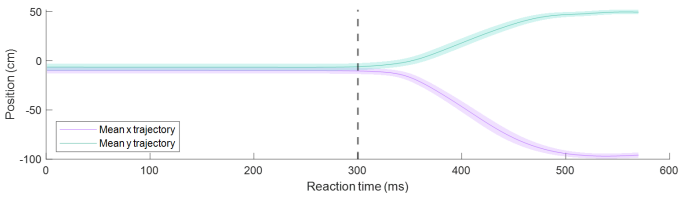


Fig. 1. Mean of the x- and y-components of the 4th trajectory across 100 trials (i.e. at $150°$) with standard deviation. This trajectory is representative of all 8 of them; the movement only begins after 320 ms.

B. Electrode Selectivity

The dataset used contains recordings from 98 electrodes. We decided to look at the tuning curves of the single electrodes to see which ones were responsive or not to the movement towards different directions. In order to calculate this we computed the (normalised) average firing rate of each electrode over the first 320 ms and 100 trials, obtaining a value representing the responsiveness of an electrode given an angle. Fig. 2 shows the tuning for all the electrodes. We then looked at the difference between the maximal and minimal firing rates at the different angles for each electrode. The bigger this range was the higher the tuning of the electrode. In Fig. 3 we plotted the four most selective ones.

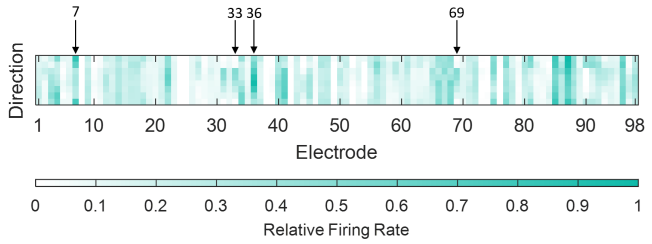


Fig. 2. Firing rate for each electrode at each direction, highlighting the four most tuned electrodes. Each column represents a tuning curve.

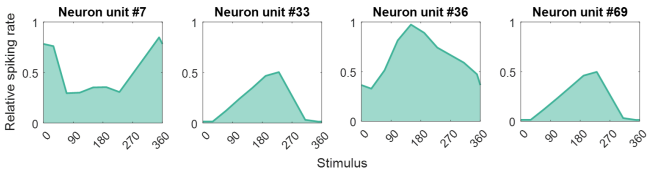


Fig. 3. Tuning curves of the most selective electrodes to the movement towards one of the 8 directions, also referred to as angles. For the full demonstration of selectivity across 98 neurons, their tuning curves can be found in Fig. 9 (Appendix).

C. Classification of Trajectory Angle with SVM

To recognise which of the 8 possible directions the monkey reaches for we used a *Support Vector Machine (SVM)* model. This is a supervised learning method that finds a hyperplane that separates the feature datapoints $\mathbf{x}$ into two classes. The hyperplane is defined as $\mathbf{w}^T\mathbf{x} + b = 0$, where $\mathbf{w}$ and b are the parameters learnt by the model and $\mathbf{x}$ is a given feature datapoint. Some of these points are defined as support vectors $\mathbf{x}_n$. They are characterised by having the smallest distance from the hyperplane.

A correctly classified $\mathbf{x}_n$ will have a distance from the hyperplane greater than 0, $|\mathbf{w}^T\mathbf{x}_n + b| > 0$. We define the margin as the distance from the hyperplane to the nearest support vector, and normalise it to be equal to 1, setting $|\mathbf{w}^T\mathbf{x}_n + b| = 1$. To find the margin we first take a vector $\mathbf{w}$ perpendicular to the hyperplane. We then take any given point $\mathbf{x}'$ on the hyperplane, such that $|\mathbf{w}^T\mathbf{x}' + b| = 0$, as shown in Fig.4. We can now project the vector $\mathbf{x}_n - \mathbf{x}'$ onto the unit vector $\hat{\mathbf{w}} = \frac{\mathbf{w}}{||\mathbf{w}||}$ and find the margin, defined as $|\hat{\mathbf{w}}^T(\mathbf{x}_n - \mathbf{x}')|$, which simplifies into $\frac{1}{||\mathbf{w}||}$. To increase

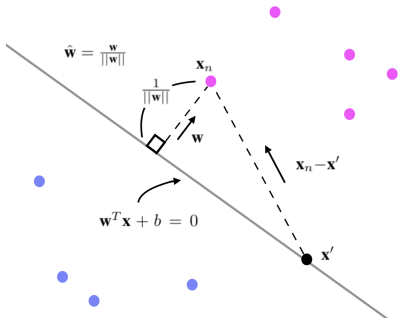


Fig. 4. Visual demonstration of how to find the margin of a 1 dimensional SVM in 2 dimensional space. This figure was constructed by a member of the team.

the data separation we want to maximise $\frac{1}{||\mathbf{w}||}$, which is equivalent to minimising its inverse $\frac{1}{2}\mathbf{w}^T\mathbf{w}$. The constraint to this optimisation is the correct classification of the datapoints $\mathbf{x}_i$. If points are correctly classified the signs will match (-1 $\times$ -1 = +1 and +1 $\times$ +1 = +1), hence we can multiply the label y_n with the output of the hyperplane's prediction, so $y_n(\mathbf{w}^T\mathbf{x}_n - b) = 1$. As the actual distance may be larger than 1 for non support vector points, $y_n(\mathbf{w}^T\mathbf{x}_n - b) \geq 1 \ \forall n$.

The underlying optimisation problem of SVM is a convex quadratic optimisation problem, for which we can derive a closed form solution exploiting the *Lagrangian duality theory*. The Lagrange dual function can be found by first constructing the *Lagrangian formulation* of the problem $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})$):

$$\begin{aligned}\min \ \mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N}\alpha_n(y_n(\mathbf{w}^T\mathbf{x}_n + b) - 1) \\ &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{n=1}^{N}\alpha_n y_n \mathbf{w}^T\mathbf{x}_n - b\sum_{n=1}^{N}\alpha_n y_n + \sum_{n=1}^{n}\alpha_n,\end{aligned} \tag{1}$$

where $y_n(\mathbf{w}^T\mathbf{x}_n - b) - 1 \geq 0$. α_n is the *Lagrangian multiplier* and is held in vector $\boldsymbol{\alpha}$. We want to minimise $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})$ w.r.t. $\mathbf{w}$ and b, and maximise it w.r.t. bounded domain $\boldsymbol{\alpha} \geq 0$. Using Karush-Kuhn-Tucker (KKT) [7] conditions we can remove $\mathbf{w}$ and b from this problem. If we imagine $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})$ as a surface then the saddle points where the gradients are 0 will be the optimal solutions to this problem. We find these by differentiating $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})$:

$$\begin{aligned}\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n = 0 \ \rightarrow \ \mathbf{w} = \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n \\ \nabla_b\mathcal{L} = -\sum_{n=1}^{N}\alpha_n y_n = 0 \ \rightarrow \ \sum_{n=1}^{N}\alpha_n y_n = 0\end{aligned} \tag{2}$$

We can use the identities to eliminate the third term of equation

(2) and simplify it further by using the new identity for $\mathbf{w}$:

$$\mathbf{w}^T\mathbf{w} = \mathbf{w}^T \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = \sum_{n=1}^{N} \alpha_n y_n \mathbf{w}^T \mathbf{x}_n \tag{3}$$

As α and y are scalar values and can be moved outside of the vector product. By using this new identity we can substitute out the remaining $\mathbf{w}$ vectors and rewrite $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})$ into its dual formulation:

$$\max\ \mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m \tag{4}$$

Subject to constraints $\alpha_n \geq 0\ \forall n$ and the previously generated identity $\sum_{n=1}^{N} \alpha_n y_n = 0$. We now have only one parameter that needs to be tuned, so we can implement the SVM by simply having a training loop where it goes through the training data and $\boldsymbol{\alpha}$ converges towards a solution using gradient descent based on some error function. The error function used in our implementation was the difference between the prediction matrix and the target matrix, but others such as Euclidean norm could also have been used.

We used 4 SVMs that were trained on the first 320ms of neuronal activity to classify the data into one of the 8 angles. Each SVM split the trajectories into one half or the other, but with different trajectories in each half. So one SVM may differentiate between 1, 2, 3, 4 and 5, 6, 7, 8, and another SVM could differentiate between 4, 5, 6, 7 and 8, 1, 2, 3. By using some basic case control the output of these SVMs could be used to calculate the correct trajectory, providing a fast accurate classifier.

D. Trajectory Localisation

After the direction has been identified we need to estimate the x and y coordinates of the monkey's hand. Looking at Fig. 1, we can see how small the standard deviation is of a given trajectory, suggesting that a method based on looking at the average trajectory can suit the problem. We then came up with a heuristic for estimating the coordinates based on their average value: we looked at the x and y coordinates of the test data at the 320th ms, right before the monkey began moving its arm but after it had been classified to an angle. We then averaged the training data set for the classified angle taking only the trajectories whose x and y coordinates, at that time instant, were less than a certain euclidean distance d from the test data's x and y coordinates. The optimal distance d was found to be 7, as shown in Fig. 5. If there is no training data at a distance whose norm is smaller than or equal to d from the test data then the average trajectory was calculated over all of the training data of the classified angle.

III. Results

Training the SVM classifier on 50% of the data resulted in a 94.5% accuracy on identifying the correct direction. When this classification was combined with our heuristic for prediction of average trajectory, we calculated the *root-mean-squared error (RMSE)* of the x and y coordinates, obtaining a value of 14.68 cm. Fig. 8 shows the performance of the 4-SVM decoder plotting the RMSE against different splits of training and testing. This was plotted by using the average and standard deviation across multiple runs for which we recorded the RMSE as shown in Fig. 12 (Appendix). Approximately 40% of the data was required to train the model such that out of sample data could be accurately predicted. RMSE was also affected by the number of electrodes used. Ordering them from most to least responsive, Fig. 6 shows that the top 40% electrodes were enough to reach near optimal performance. Using all available electrodes does not introduce significant noise as error did not increase proportionally with the number of electrodes used.

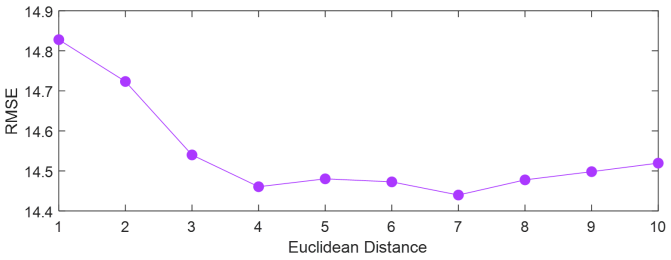


Fig. 5. Variation of RMSE given different values of d, the Euclidean distance threshold used to average trajectories.

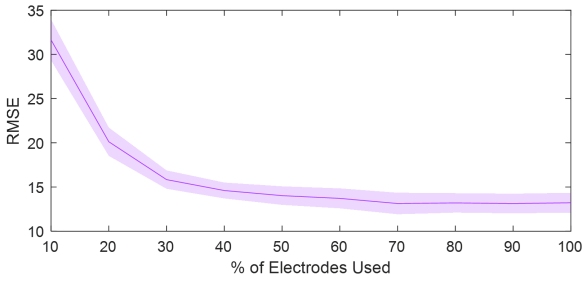


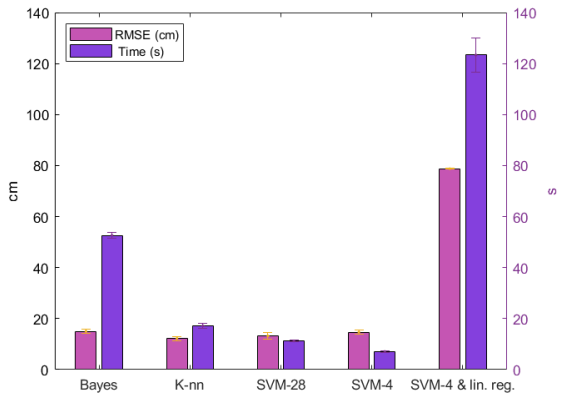
Fig. 6. RMSE against percentage of electrodes used for training and estimation. The electrodes kept are prioritised by their responsiveness looking at their tuning curves. Results were obtained by averaging over 10 different runs.

IV. Discussion

We compared our decoder with other methods commonly used in the field [8]. The classifiers we tried use *Naive Bayes*, with a Gaussian likelihood, *K-nearest neighbours (KNN)* and another version of *SVM*. For each of these classification techniques we used the average trajectory method to estimate the x and y coordinates. The results are shown in Fig. 7 in which we compare the running time and RMSE. We can see that although they all have similar RMSEs there is a significant difference in the run times. The Bayesian classifier takes the longest and our model (*SVM-4*), is the fastest. We can also see that the Bayes classifier is the least accurate and the second version of SVM the most accurate.

The difference between the two SVM versions is the number of support vector machines used. In the other version, *SVM-28*, we trained a support vector machine for every possible 2-angle

combination resulting in 28 different models. Each of these 28 models was used to classify the data to an angle and then the mode of all the classifications was used as the predicted angle. This was highly accurate but the time complexity was high, and the training stage of the final version was 7 times faster. Other attempts at optimising the SVM model were found to have more drawbacks than benefits. When testing with a soft margin SVM the results barely changed and when testing with a Gaussian kernel the error increased drastically, suggesting that the data is already separable and does not require more complex algorithms. As these are also somewhat more computationally intensive this meant we could accelerate our classification and prediction by using the simpler model.



Classifier	N. Bayes	K-NN	SVM-28	SVM-4
Accuracy	87.25%	91.5%	96.28%	94.05%

Fig. 7. (**A**) Average RMSE and run time for the different classifiers tried. We include SVM-4 with both average trajectory and linear regression to display the superiority of the average trajectory method. (**B**) Classification accuracy of every classifier using a $50/50$ split.

We also compared two different methods to perform the trajectory retrieval: average trajectory with heuristic and linear regression.

Although linear regression would be a more sophisticated approach it turns out it does not perform as well as using the mean trajectory. This can be seen in Fig. 7 in which we compare the run time and RMSE of our model for both trajectory retrieval methods. Using linear regression results in a slower decoder and a higher RMSE. The difference between the estimation based on the standard average trajectory instead of our heuristic is little, resulting in an RMSE of 16.32 cm vs. 14.68 cm.

V. Conclusions

By first using SVM-4 and then estimating the average trajectory with our heuristic, we get an average RMSE of 14.68 cm, using half of the data to train on and the other half to test on. For greater accuracy of real-life prediction of arm movements, more data would be required to train the model.

Attributions

The neural data have been generously provided by the laboratory of Prof. Krishna Shenoy at Stanford University.

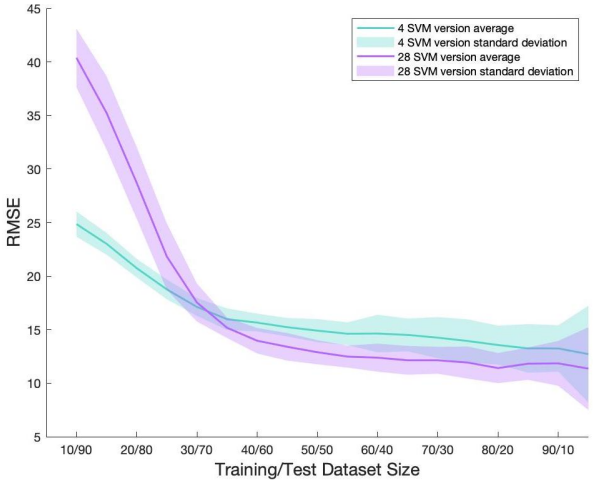


Fig. 8. RMSE of the predicted trajectory against increasing training sizes. We can see that overfitting begins to happen when the training set is very large, as the standard deviation of RMSE increases. Results were obtained by averaging over many runs.

Each member contributed equally and helped writing the report. **E.F.** developed the Bayes' classifier, **K.L.** developed the SVM-4 classifier, **C.L.** developed the SVM-28 classifier and the average trajectory based on euclidean distance for trajectory prediction, **D.Z.** developed the (weighted-) KNN classifier and linear regression for trajectory prediction. All the code is open source and can be found on GitHub [9].

References

[1] M. Spüler, E. López-Larraz, and A. Ramos-Murguialday. On the design of eeg-based movement decoders for completely paralyzed stroke patients. *Journal of NeuroEngineering and Rehabilitation*, 15(1), 2018. doi: 10.1186/s12984-018-0438-z.

[2] E. López-Larraz, F. Trincado-Alonso, and L. Montesano. Brain-machine interfaces for motor rehabilitation: Is recalibration important? *14th International Conference on Rehabilitation Robotics (ICORR)*, 08 2015. doi: 10.1109/ICORR.2015.7281203.

[3] Y.C.A. Padmanabha Reddy and N. Mohan Krishna Varma. Review on supervised learning techniques. In P. Venkata Krishna and Mohammad S. Obaidat, editors, *Emerging Research in Data Engineering Systems and Computer Communications*, pages 577–587, Singapore, 2020. Springer Singapore.

[4] J.I. Glaser, A.S. Benjamin, R. Farhoodi, and K.P. Kording. The roles of supervised machine learning in systems neuroscience. *Progress in Neurobiology*, 175:126–137, 2019. doi: 10.1016/j.pneurobio.2019.01.008.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[6] S.H. Scott. Optimal feedback control and the neural basis of volitional motor control. *Nature Reviews Neuroscience*, 5(7): 532–545, 2004.

[7] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. doi: 10.1017/CBO9780511809682.

[8] N. Das, N. Nagpal, and S.S. Bankura. A review on the advancements in the field of upper limb prosthesis. *Journal of medical engineering & technology*, 42(7):532–545, 2018.

[9] D. Zerkalijs, C. Lazzaroli, E. Faillace, and Lawrence K. No brain no gain. `https://github.com/deezee30/nbng`, 2021.

APPENDIX

Appendix A - Supplementary Plots

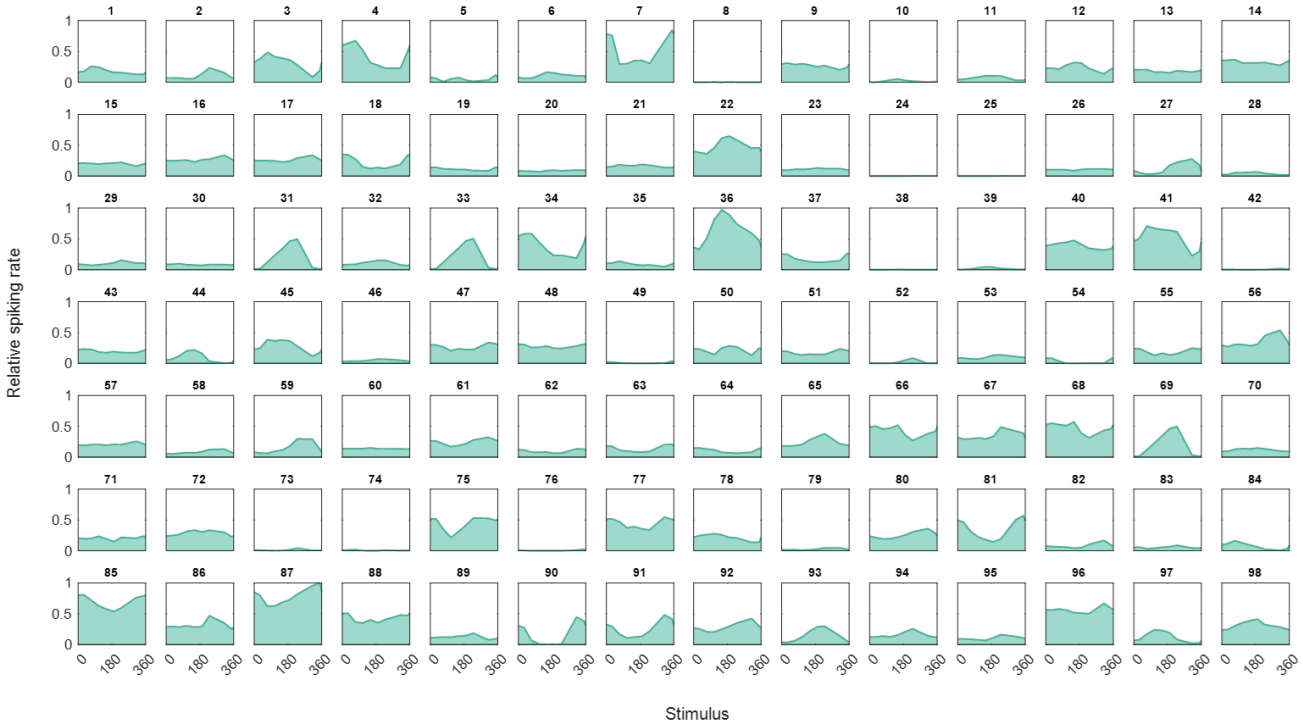


Fig. 9. Tuning curves for all 98 electrodes, indicating their degree of selectivity towards a particular binned stimulus (direction) or range of binned stimuli (directions). The responses (firing rates) are averaged out across 100 trials and normalised with respect to each other.

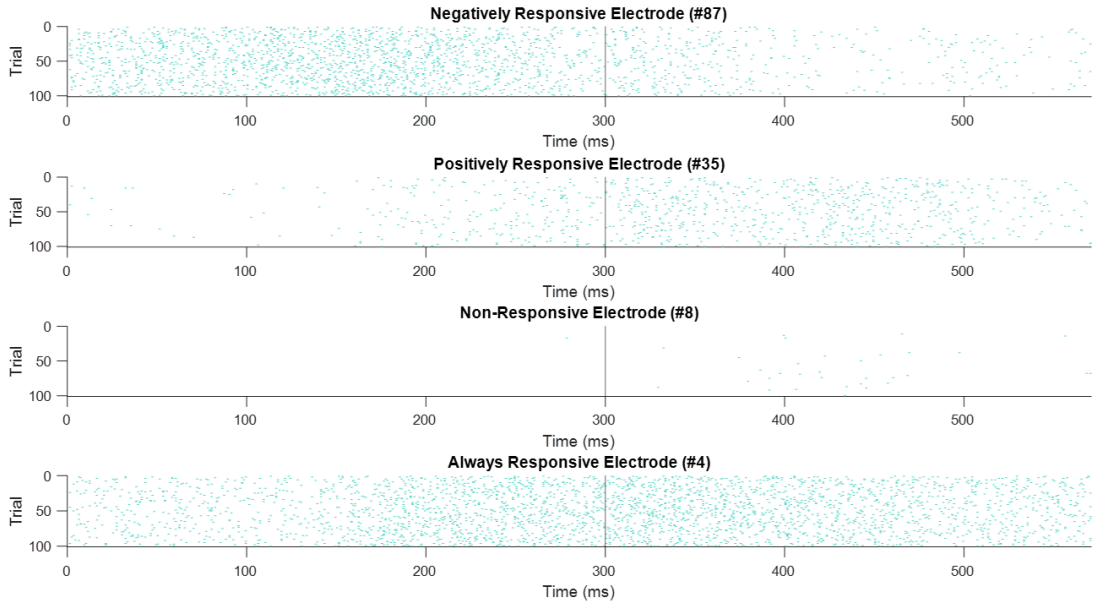


Fig. 10. Raster plot of four electrodes for movement in direction 1. The first electrode records activity mostly before the onset of the movement (negatively responsive), the second electrode records activity mostly after the onset of the movement (positively responsive), the third electrode almost never records activity (non responsive) and the fourth electrode almost always records activity (always responsive).

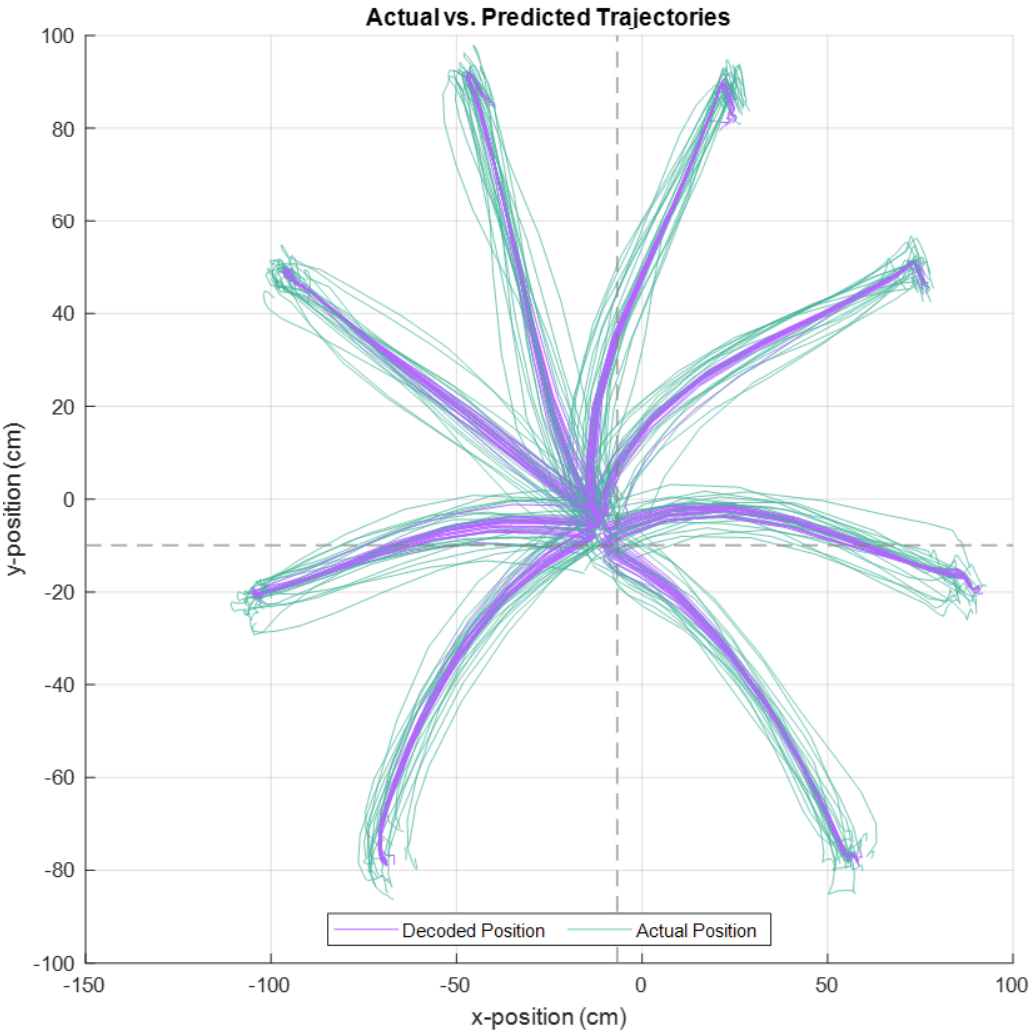


Fig. 11. Actual and predicted trajectories with SVM-4 and average trajectory model.

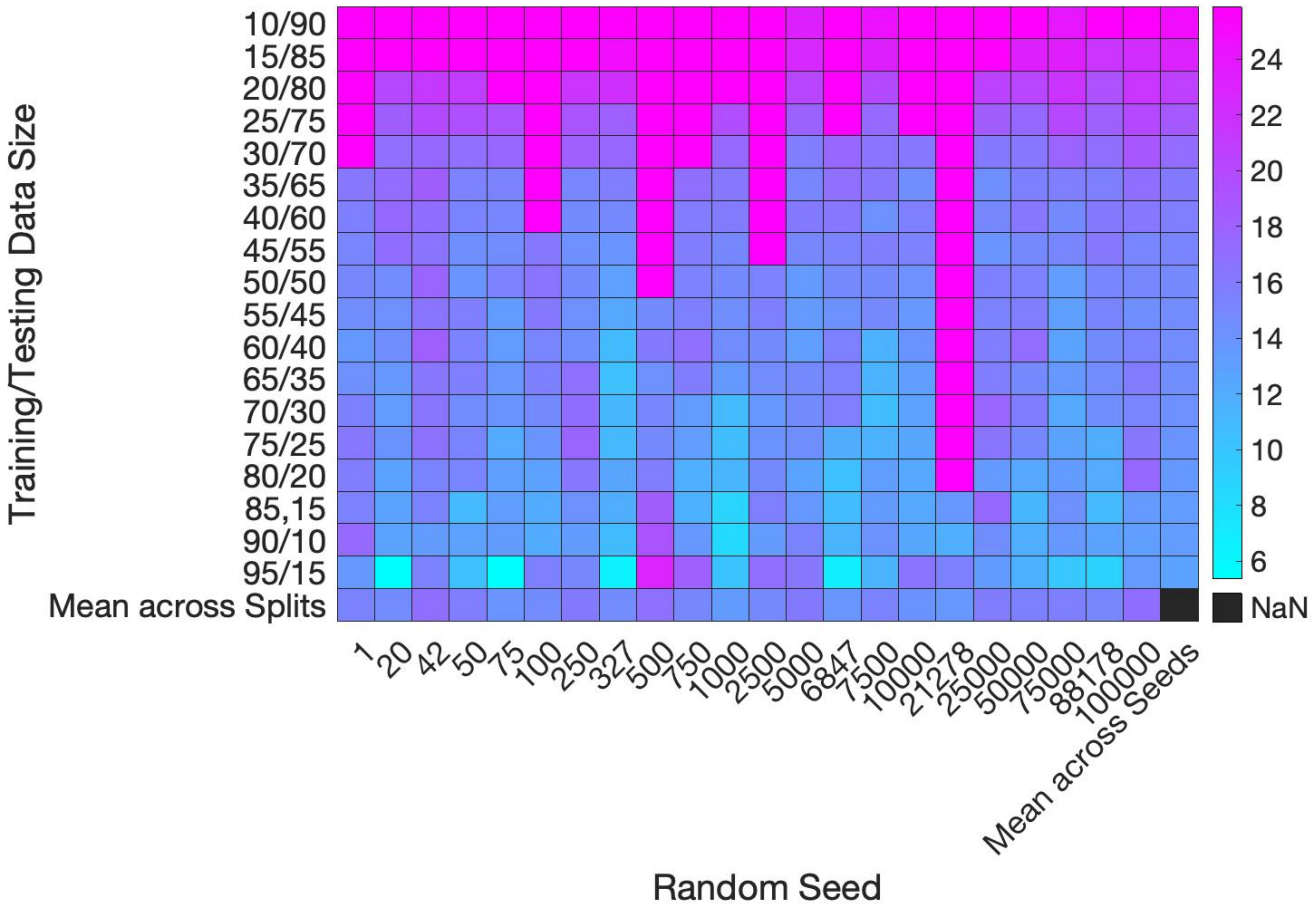


Fig. 12. Heatmap of the RMSE across runs with various random number generation seeds and training/test data splits.