Neural Decoder for Continuous Estimation of Monkey Hand Positions

Aaruni Arora (*A.A.*), Joanna Gryczka (*J.G.*), Mahad Parwaiz (*M.P.*), and Indumita Prakash (*I.P.*).
Team *JamiBananas.mat*, Brain Machine Interfaces, Department of Bioengineering, Imperial College London, London, UK.

***Abstract*—Brain–Machine Interfaces (BMIs) enable motor intention decoding from neural activity to restore movement and communication. This work presents a decoding pipeline for predicting monkey hand trajectories during eight-direction reaching movements using multi-unit spike recordings. Preprocessing included padding, binning, square-root transformation, EMA smoothing, and dimensionality reduction via PCA and LDA. Empirical evaluation showed that linear models outperformed non-linear and regularised variants, with soft kNN and Principal Component Regression (PCR) achieving the best accuracy (99.4%) and lowest trajectory RMSE (6.713 cm). Although direction was reliably classified, predicted trajectories lacked spatial variance. Results highlight the effectiveness of linear methods for decoding in low-noise datasets and provide a robust foundation for real-time BMI applications.**

***Index Terms*—Brain–Machine Interface, Neural Decoding, PCA, LDA, soft kNN, PCR.**

I. Introduction

Brain-Machine Interfaces (BMIs) enable direct communication between neural circuits and external devices, significantly enhancing the quality of life for individuals with mobility impairments. It has applications in prosthetics, rehabilitation, and communication. However, accurate decoding of neural signals into precise movements remains a challenging goal due to neural signal variability and noise. This study developed and compared machine learning models trained on neural activity recorded from a monkey performing reaching movements along eight distinct directions. The primary goal was to optimise causative decoding algorithms capable of accurately predicting hand position trajectories, while reducing computational latency. [1]

II. Methods

A. Dataset and Approach

The dataset consists of 182 trials per direction. Each trial contains neural data from 98 neural units and corresponding hand position recordings along the x, y, and z axes. The neural data are represented as binary spike trains recorded at $1\,ms$ resolution. Each trial spanned from $300\,ms$ prior to movement onset to $100\,ms$ after movement end.

Initial exploratory analysis involved generating tuning curves to identify neurons' preferred directions, raster plots to visualise spike activity, and peri-stimulus time histograms (PSTHs) to examine temporal firing patterns (plots in GitHub). These analyses revealed variability in trial durations, noise levels, and data inconsistencies, highlighting the need for systematic pre-processing and alignment before decoding and regression analyses.

Model performance was rigorously evaluated using three complementary metrics: (i) trajectory root-mean-squared error (RMSE), computed between the true and predicted hand paths (ii) Direction classification accuracy, defined as the proportion of trials where the model correctly predicted the intended movement direction; and (iii) per-update latency, i.e., the wall-clock time required to generate each position estimate. Results were obtained using K-fold cross-validation with $K = 10$, training the model on $K-1$ folds and averaging metrics across the folds. This approach ensures an unbiased estimate of model generalisation.

B. Data Preprocessing

1) Padding

Initial preprocessing involved padding spike trains with zeros to match the longest trial duration across the dataset, ensuring uniform trial lengths. Similarly, hand position trajectories were padded by repeating the final recorded position, maintaining dimensional consistency.

2) Handling Missing Values and Mean Centring

Missing values (`NaNs`) in spike data were replaced with zeros, while missing hand positions were interpolated using forward and then backwards filling. The hand positions were also mean-centred for consistency across trials.

3) Binning and Removal of Low Firing Rate Neurons

Both the spike and hand positions data were then aggregated into non-overlapping bins of $20\,ms$, significantly reducing dimensionality and noise while preserving temporal alignment. Neurons exhibiting average firing rates below 0.5 Hz were excluded to reduce signal-to-noise ratio and maintain relevance.

4) Square Root Transformation

To further stabilise variance often observed with low spike counts, a square root transformation was applied: $y = \sqrt{x}, \quad \text{for} \quad x \geq 0$. [2]

5) Smoothing and Feature Extraction

Thereafter, two smoothing filters were evaluated on the transformed data: Gaussian and Exponential Moving Average (EMA). Gaussian smoothing involved the convolution of spike counts with a discrete Gaussian kernel of window size $\sigma = 50$. Alternatively, the EMA filter is applied recursively, weighting recent data more heavily than older data:

$$y_t = \alpha x_t + (1-\alpha) y_{t-1}, \quad \text{with} \quad y_0 = x_0, \tag{1}$$

where x_t is the current input value, y_t is the filtered value, and α is the smoothing factor (with $0 \leq \alpha \leq 1$) [3].

The resulting filtered values are then scaled by a conversion factor of ($\frac{\text{bin group}}{1000}$) to yield firing rates in *spikes/second*. Finally, spike data was reshaped into feature matrices, forming the input for subsequent classification and regression analyses.

C. Dimensionality Reduction of Neural Data

1) Principal Component Analysis (PCA)

To reduce dimensionality while preserving key neural variance, Principal Component Analysis (PCA) [1] was applied to the preprocessed neural data. PCA linearly and orthogonally transforms the original features into uncorrelated principal components (PCs) in a descending order of their explained variance. After mean-centring, the sample covariance matrix A is computed as:

$$\mathbf{A} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T \quad \text{for} \quad var(v) = v^T\mathbf{A}v. \quad (2)$$

The eigenvectors v and eigenvalues λ of A satisfy $Av = \lambda v$, where v is the direction of the PC and λ the corresponding variance. By selecting only a subset of the top principal components and projecting these back onto the neural data, dimensionality is reduced while retaining structural relevancy.

2) Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) [1] was applied after PCA to improve separability in reduced space and support classification of movement direction from neural activity. LDA seeks a linear projection that maximises class separability by finding a direction w that best separates the means of different classes (μ_1, μ_2) while minimising within-class variance. The optimal projection is then given by $w = \Sigma^{-1}(\mu_1 - \mu_2)$. Each new sample x is classified based on the sign of the linear discriminant $y = \text{sign}(w^\top x + w_0)$ with the threshold w_0 typically set midway between projected class means. This formulation maximises the Fisher criterion, i.e., the ratio of between-class to within-class variance, yielding a decision boundary optimal under Gaussian assumptions. In our multi-class case ($K = 8$ reach directions), LDA reduces the data to at most $K - 1$ dimensions.

D. Classification

To classify neural signals into discrete movement categories after dimensionality reduction, three classifier families - SVM, Nearest Centroid, and k Nearest Neighbours (kNN) - were evaluated for their effectiveness in neural decoding literature.

1) Support Vector Machines (SVM)

Support Vector Machines (SVM) [1] work by finding the hyperplane that best separates data points of different classes, maximising the margin - the distance between the hyperplane and the nearest data points (called support vectors). A Linear-SVM kernel gave the highest accuracy and computational efficiency.

2) Nearest Centroid Classification

In this supervised learning method, the training data is first grouped by the eight movement directions, and each group's centroid in PCA space is computed. For each test sample, the squared Euclidean distance to all centroids is calculated, and the sample is classified according to the nearest centroid [1].

3) k-Nearest Neighbor (kNN)

The k-Nearest Neighbour algorithm [1] assigns class labels based on a defined proximity metric in the feature space to known labelled points.

a) Hard kNN

The hard kNN approach classifies each test point x by majority voting among its k nearest neighbours. Mathematically, the predicted class c is given by:

$$c = \text{argmax}_j \sum_{i \in N_k(x)} I(y_i = j), \quad (3)$$

where $N_k(x)$ denotes the set of k nearest neighbours of x, $I(\cdot)$ is an indicator function that returns 1 if the condition is true, and 0 otherwise, and y_i is the class label of the i^{th} nearest neighbour [4].

Hard kNN equally weights neighbours, making it prone to misclassification when classes are closely spaced. To address this, we introduced a confidence measure, computing a score based on the fraction of neighbours voting for the predicted label across test samples. However, results showed increased computation without significant accuracy improvement.

b) Soft kNN

Lastly, soft kNN assigns weights to neighbours based on their distance from the test point. Two weighting formulae were evaluated:

1. Inverse Distance Weighting: $w_i = \frac{1}{d_i^p + \epsilon}$, where w_i is the weight assigned to neighbour i, d_i is the Euclidean distance from the test point to neighbour i, p controls the weight decay (typically $p = 1$), and ϵ is a small constant to avoid division by zero [5].

2. Exponential Distance Weighting: $w_i = e^{-\alpha d_i}$, where α is a scaling factor controlling the exponential decay of weights based on distance [6].

Here, the predicted class c is determined by $c = \text{argmax}_j \sum_{i \in N_k(x)} w_i \cdot I(y_i = j)$. This variant provides a more aggressive emphasis on nearer neighbours, enhancing classification performance in noisy, high-dimensional datasets.

E. Regression

1) Linear Regression

Initial attempts used ordinary least squares (OLS) regression [1], [7] on doubly-reduced features (PCA followed by LDA), but the second projection discarded informative variance and increased RMSE. As a result, LDA was excluded for hand position decoding, and principal component regression (PCR) was adopted. The PCA scores matrix $R \in \mathbb{R}^{n \times p}$ was computed from preprocessed firing rates, where n is the number of trials and p the number of principal components (PCs). Mean-centred hand position vectors $z_x, z_y \in \mathbb{R}^n$ were then regressed using OLS:

$$B_x = (R^T R)^{-1} R^T z_x, \quad B_y = (R^T R)^{-1} R^T z_y. \quad (4)$$

The resulting coefficient vectors $B_x, B_y \in \mathbb{R}^p$ reside in PCA space. To obtain full-dimensional regression weights, they

were mapped back using the PCA loading matrix $V \in \mathbb{R}^{d\times p}$:

$$w_x = VB_x, \quad w_y = VB_y. \tag{5}$$

Predictions for new test data r_{test} were then made using:

$$\hat{x} = w_x^\top r_{\text{test}} + \bar{x}, \quad \hat{y} = w_y^\top r_{\text{test}} + \bar{y}, \tag{6}$$

where $\bar{x}, \bar{y}$ are the average training positions. Initially, a single weight vector per direction was trained, but accuracy was limited due to temporal drift in the neural–kinematic mapping. The final model fit separate weight vectors for each time bin and movement direction.

2) PCR Modifications

To further explore regularisation effects, ridge-regularised PCR was implemented by adding an L_2 penalty to stabilise weight estimation using the formula $\beta_{\text{ridge}}^{(t,k)} = (X^TX + \lambda I)^{-1}X^Tz$, where λ controls shrinkage.

Similarly, sparsity was investigated through lasso-regularised PCR, solving the L_1-penalised problem with $\beta_{\text{lasso}}^{(t,k)} = \arg\min_\beta \left(\|X\beta - z\|_2^2 + \lambda\|\beta\|_1\right)$.

Finally, polynomial regression (PR) models of order $p = (1, 2, 3, ..)$ were evaluated by expanding PCA scores with polynomial features before fitting, assessing whether non-linear mappings improved decoding [7].

3) Kalman Filtering (KF)

Kalman Filtering (KF) [1] is a recursive Bayesian estimator that operates in two steps: a prediction step (next state estimation based on current state) and an update step (correction based on new observations). KF is widely used for neural decoding due to its ability to efficiently handle noisy, time-varying data and it applications in real-time neural decoding.

4) Least Mean Squares (LMS)

The Least Mean Squares (LMS) algorithm [8] minimises the mean squared error between predicted outputs $\hat{\mathbf{Y}}$ and true values $\mathbf{Y}$ through iterative weight updates. Given input features $\mathbf{X}$ and learning rate μ, the prediction and update steps are $\hat{\mathbf{Y}} = \mathbf{XW}$ and $\mathbf{W} \leftarrow \mathbf{W} + \mu\mathbf{X}^T(\mathbf{Y} - \hat{\mathbf{Y}})$, respectively. LMS uses a gradient descent approach to update weights in real time, making it suitable for large datasets.

F. Hypertuning

Weights & Biases[1] was used to tune hyperparameters for preprocessing and classification. Grid search was selected for its effectiveness, given the small number of hyperparameters and well-defined value ranges. Each run was ranked by its RMSE on a validation split. After selecting the top few best configurations, the model was retrained on the combined training + validation data and evaluated once on a holdout test set, to minimise the risk of overfitting, reducing RMSE by 9%.

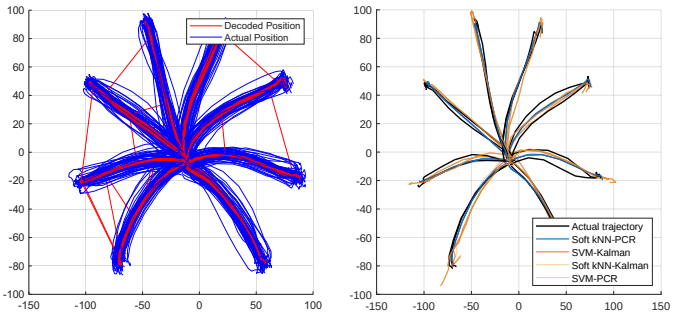
III. Results and Discussion

The final decoded trajectories and comparisons with alternate model variants are shown in Fig. 1.

A. Data Linearity

Exploratory analysis suggested that the neural dataset exhibited a predominantly linear structure. This justified the use of linear techniques such as PCA over non-linear methods like t-SNE, and linear regression over higher-order polynomials for our model design.

[1]Weights & Biases. [Online]. Available at https://wandb.ai



(a) True (blue) vs. decoded (red) hand position trajectories across all trials using the final decoder (soft-kNN + PCR).

(b) Comparison of hand position trajectories across decoder variants and directions.

Fig. 1: Decoded hand position trajectories in X–Y space.

B. Preprocessing

The final decoder used a 20 ms bin size, padding, square root transformation, exclusion of 3 low-firing neurons, and exponential moving average (EMA) smoothing. Padding was preferred over truncation to preserve temporal structure in variable-length trials. Empirical results indicated that EMA outperformed Gaussian smoothing in both RMSE and runtime. Table I suggests that EMA is more robust in real-time decoding contexts.

TABLE I: 10-fold cross validation RMSE (cm) and run-time (s) results for different preprocessing window types and bin sizes with soft-kNN and linear regression. EMA with a bin size of 20 ms and smoothing parameters $\alpha_{\text{train}} = 0.35$, $\alpha_{\text{test}} = 0.30$ achieved the lowest RMSE (6.5066 cm).

Window Type	Bin Size (ms)	RMSE	Run-Time (s)
Gaussian ($\sigma = 50$)	5	7.6518	412.66
	10	6.9974	72.38
	20	6.8051	15.01
EMA ($\alpha_{train} = 0.35, \alpha_{test} = 0.3$)	5	7.3398	473.75
	10	6.6863	109.27
	20	**6.5066**	**19.73**

C. Dimensionality Reduction

Two PCA strategies were evaluated to reduce dimensionality: (a) using a fixed number of PCs (b) selecting a number based on a variance-retention threshold to calculate PCs. The threshold-based approach allowed adaptive dimensionality depending on data variability and resulted in faster runtime with no degradation in decoding accuracy. Fig 2 summarises the selected hyperparameters.

D. Classification

Among the classification methods evaluated, soft kNN using inverse distance weighting achieved the highest accuracy in the test set (Table II), with $k = 20$ and Manhattan distance ($p = 1$) providing the best bias-variance trade-off. Compared to hard kNN, soft kNN more effectively managed noise and improved class boundary distinction.

Although the nearest-centroid method offered faster inference, its sensitivity to outliers and class overlap led to abrupt misclassification. SVMs achieved slightly lower accuracy and

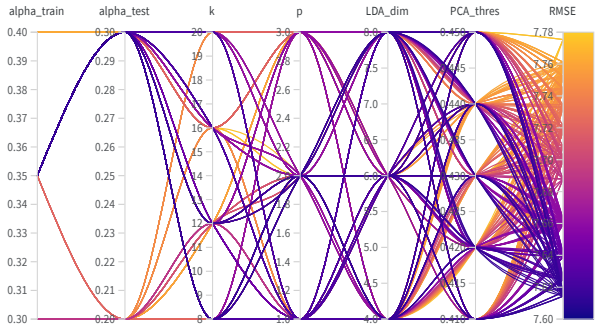


Fig. 2: Optimised hyperparameters for the final decoder, visualised using a parallel coordinates plot generated via Weights & Biases. Each line represents one hyperparameter trial, coloured by the resulting RMSE. The selected configuration retained 44% variance via PCA, reduced the LDA space to 6 dimensions, used $k = 20$ neighbours for soft kNN, and applied EMA smoothing with $\alpha_{\text{train}} = 0.35$ and $\alpha_{\text{test}} = 0.30$.

exhibited significantly higher runtime, making them unsuitable for real-time deployment.

TABLE II: Classification accuracy (%) and run-time (s) comparison for 5 decoding methods using PCA–LDA features. Soft kNN and Nearest Centroid variants achieved similar accuracy, with soft kNN (inverse distance weighted) preferred due to robustness to class overlap.

Method	Accuracy (%)	Run-Time (s)
LDA - NN	99.49	10.85
LDA - kNN (hard)	99.45	10.96
LDA - kNN (soft - dis)	**99.40**	**12.53**
LDA - kNN (soft - exp)	99.45	12.29
SVM	98.95	19.40

E. Regression

Standard PCR achieved the lowest RMSE (Fig 3), and fastest inference time (Fig 4). Ridge and Lasso regression offered no consistent performance improvement and increased computational cost due to regularisation. PR models with orders > 1 led to over-fitting and degraded performance, confirming that a linear mapping was sufficient. LMS regression produced the highest RMSE and was therefore excluded from the final decoder. KF, while effective in continuous estimation, often produced overshooting trajectories, as seen in Fig 1b.

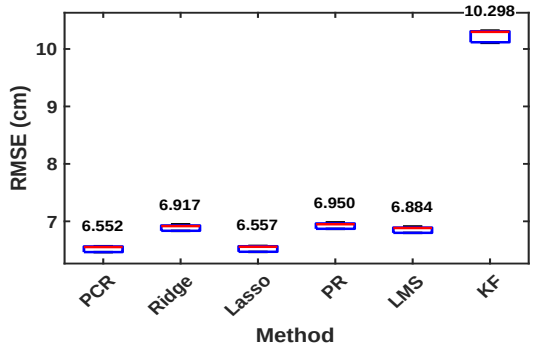


Fig. 3: Comparison of RMSE for different regression models on test data using 10-fold cross-validation across 5 random seeds.

IV. Conclusion and Future Work

A neural decoding pipeline was developed to predict monkey hand trajectories from spike data, combining structured preprocessing, dimensionality reduction, soft kNN classification, and principal component regression. While the pipeline achieved a low RMSE of 6.552 cm with an average runtime of 6.78 s, Fig. 1a reveals frequent directional misclassifications along the trajectory, causing abrupt jumps. Ensemble strategies and majority voting were tested but increased decoding time significantly. Predicted trajectories were also overly centred, indicating limited reconstruction of spatial variance. KF was evaluated to address this, but it suffered from overshooting, which means it requires a stop function. Future work could explore recurrent or latent state-space models to improve temporal consistency and spatial accuracy.

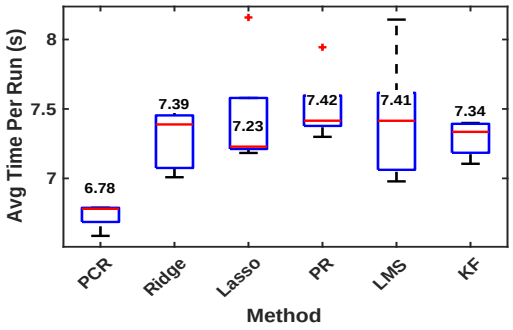


Fig. 4: Runtime comparison of regression models averaged over 10-fold cross-validation with 5 random seeds.

Attributions

All authors contributed to this paper. The main responsibilities were: *A.A.*: Preprocessing, kNN, KF, PCR. *J.G.*: Exploratory Analysis, LMS, Hypertuning. *M.P.*: Preprocessing, Nearest Centroid, PCR, Evaluation. *I.P.*: Dimensionality Reduction, SVM, PR. Link to GitHub Repo: `https://github.com/BMI_Coursework.git`

References

[1] R. P. Rao, *Brain-Computer Interfacing: An Introduction*. Cambridge, U.K.: Cambridge University Press, 2013.

[2] Number Analytics. (2022) Enhancing data analysis: Square root transformation. Accessed: 2025-05-04. [Online]. Available: https://www.numberanalytics.com/blog/enhancing-data-analysis-square-root-transformation

[3] M. Cousins. (2021) Exponential moving average (ema) filter. Accessed: 2025-05-04. [Online]. Available: https://blog.mbedded.ninja/programming/signal-processing/digital-filters/exponential-moving-average-ema-filter/

[4] S. Zhang, X. Li, M. Zong, X. Zhu, and R. Wang, "Efficient knn classification with different numbers of nearest neighbors," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, 2018.

[5] Z. Wei, W. Zhang, and C. Zhong, "A distance-weighted k-nearest neighbor algorithm for multi-label classification," *Pattern Recognition Letters*, vol. 118, pp. 115–121, 2019.

[6] S. García, J. Luengo, and F. Herrera, "Data preprocessing in data mining," in *Data Preprocessing in Data Mining*. Cham: Springer, 2020.

[7] M. R. Putri, I. G. P. S. Wijaya, F. P. A. Praja, A. Hadi, and F. Hamami, "The comparison study of regression models (multiple linear regression, ridge, lasso, random forest, and polynomial regression) for house price prediction in west nusa tenggara," in *2023 International Conference on Advancement in Data Science, E-learning and Information System (ICADEIS)*, Bali, Indonesia, 2023, pp. 1–6.

[8] E. Walach and B. Widrow, "The least mean fourth (lmf) adaptive algorithm and its family," *IEEE Transactions on Information Theory*, vol. 30, no. 2, pp. 275–283, 1984. [Online]. Available: https://doi.org/10.1109/TIT.1984.1056886