4G10 Assignment 2
Predicting Hand Kinematics from Neural Data

Candidate Number: 5548E

December 24, 2024

Introduction

Decoding neural signals to predict movement is a key challenge in brain-machine interface (BMI) research. BMIs require accurate decoding models to decipher intended movements from neuromotor activity and generate appropriate command signals to control a computer cursor, prosthetic limb, functional electrical stimulation device, or other assistive devices. This coursework investigates methods for predicting hand velocities from neural data recorded from a reaching monkey. Two approaches are explored: a simple linear decoder and a more sophisticated Kalman filter. The performance of these methods is compared in terms of accuracy and suitability for BMI applications, with a focus on their strengths, limitations, and real-world feasibility.

1 Simple Decoder

A simple linear decoder was implemented to predict hand velocities $\hat{v}_{k,t}$ from smoothed neural signals $\tilde{x}_{k,t}$ in trial k. This mapping was achieved through a decoding matrix W:

$$\hat{v}_{k,t} = W\tilde{x}_{k,t}$$

The decoding matrix W was determined using ridge regression, a variation of linear regression that incorporates regularisation. To preprocess the data, the neural signals were smoothed using a Gaussian filter. The ridge regression weights were computed as:

$$W^\star = V\tilde{X}^\top(\tilde{X}\tilde{X}^\top + \lambda I_N)^{-1}$$

where λ is the regularisation parameter, V is the matrix of hand velocities, and X is the matrix of smoothed neural signals.

1.1 Implementation

The regularisation parameter λ controls the trade-off between minimising the loss and penalising large weights. A large λ imposes a stronger penalty, leading to smaller weights and a simpler model that may underfit the data. A small λ imposes a weaker penalty, allowing the model to fit more closely to the data but increasing the risk of overfitting. To select an optimal λ, I hypothesised that it should be on the same order as the terms in $\tilde{X}\tilde{X}^\top$, which have an average value of 418. I validated this hypothesis using cross-validation, splitting the neural training data into three subsets: 60% for training, 20% for validation, and 20% for testing. For each candidate λ, $W^\star$ was fitted using the training set, and the model was evaluated on the validation set. The λ that yielded the lowest mean MSE was selected as optimal.

The cross-validation results, shown in Figure 1 (left), identified an optimal λ of 112.883, achieving a mean validation MSE of 0.539. To confirm the model's generalisation ability, I retrained it on the combined training and validation sets using this λ and evaluated it on the test set. The test MSE was 0.562, closely matching the validation MSE, indicating that the model generalised well and did not overfit the data.

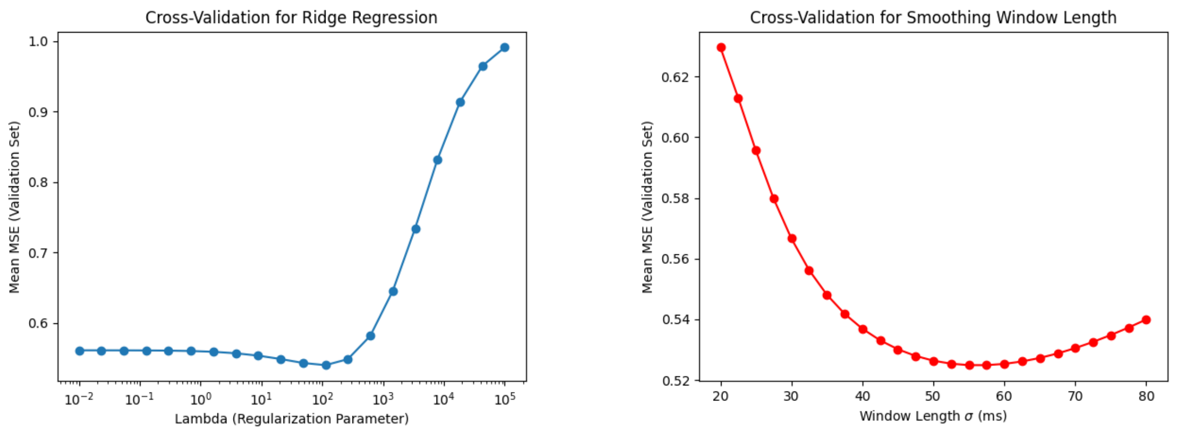

Figure 1: Cross Validation of λ (left) and Smoothing Window Length σ (right)

1.2 Smoothing Factor

The choice of smoothing factor significantly impacts the performance of the decoder. Cross-validation was used to determine the optimal smoothing window length σ. The results, shown in Figure 1 (right), indicate that a smoothing window of 57.5 ms produced the lowest mean MSE of 0.516. The test MSE was found to be 0.524, demonstrating that the model generalises well and is not overfitting.

Figure 1 (right) also illustrates how prediction accuracy varies with the smoothing window length σ. A very short smoothing window (e.g., 20 ms) retains noise in the spike counts, resulting in poor accuracy despite capturing fine-grained temporal details. Intermediate window lengths (45 ms to 65 ms) achieve the best prediction accuracy by balancing noise reduction while also retaining enough of the signal. Conversely, overly long smoothing windows result in over-smoothing, leading to a decline in prediction accuracy due to excessive averaging.

1.3 Suitability of the Simple Decoder

The suitability of the simple decoder for use in BMI applications can be assessed across three key factors:

1. **Feasibility:** The Gaussian smoothing and linear decoding processes are computationally lightweight, making the approach feasible for real-time implementation in BMI systems. However, the Gaussian smoothing filter used here is non-causal, introducing a delay of $\frac{\sigma}{2}$ (approximately 30 ms in this case) before estimated velocities can be obtained. While this delay may be acceptable for some BMI applications, it could be problematic for tasks requiring near-instantaneous responsiveness.

2. **Computational Tractability:** The simple decoder is computationally tractable, as it involves only basic matrix operations such as multiplication and inversion, which can be efficiently computed. It involves Gaussian smoothing (a convolution operation) followed by a single matrix multiplication to map neural activity to hand velocity, making it very lightweight computationally. It takes 0.114 seconds for the simple decoder to predict the hand velocities from the test neural data.

3. **Accuracy:** The simple decoder achieved an R^2 of 0.444 when predicting hand velocities from neural data in the test trials. This indicates that the decoder explains only 44% of the variance in the data, which is relatively low. While the simplicity of the approach contributes to its computational feasibility, the accuracy may be insufficient for applications requiring high precision. The inherent limitations of linear models and the lack of temporal dynamics handling contribute to this lower accuracy.

The suitability of the decoder could be enhanced by addressing the temporal lag introduced by the smoothing process. This lag occurs because the non-causal Gaussian filter considers both past and future data points during smoothing. A causal Gaussian filter, which only considers current and past data points, was implemented by truncating the Gaussian kernel to include only non-negative time points. Using this filter, the optimal smoothing window length was determined to be 80 ms. Surprisingly, the model trained on data smoothed with this filter achieved an R^2 of 0.458 on the test set, slightly higher than the non-causal filter's R^2 of 0.444. This unexpected improvement could stem from the causal filter better aligning the smoothed signals with real-time neural dynamics, thereby reducing potential artifacts introduced by future data points in the non-causal filter.

1.4 Preprocessing of Hand Velocity Data

The hand velocity data was shifted back by 120 ms relative to the neural data to account for the natural delay between neural activity and physical movement. This delay, typically ranging from 100 ms to 200 ms, reflects the time required for the brain to process sensory information, plan movements, transmit motor commands, and activate muscles. The 120 ms shift likely represents an average motor delay for the studied population of monkeys. By shifting the velocity data, the neural signals at time t were better aligned with the corresponding hand movements occurring at $t + 120$ ms.

2 Kalman Filter-based Decoder

A more sophisticated approach was implemented using the Kalman filter. First, a latent linear dynamical system (LDS) model was pre-trained using the training data, leveraging a smoothing process to extract latent kinematic signals. This established the autoregressive prior that captured the temporal dynamics of these signals. Next, a generative model was constructed by combining this latent prior with a neural likelihood, linking the latent states to observed neural spike counts. Finally, the Kalman filter was used to invert this generative model, allowing for the estimation of the posterior distribution of the latent states given the observed neural data. These posterior estimates were then substituted into the observation equation for hand velocities, enabling prediction of velocities from the neural data.

2.1 Parameters of the Generative Model

To construct the generative model, it was necessary to estimate the likelihood parameters D and S in the observation model:

$$x_{k,t} = Dz_{k,t} + \xi_{k,t}, \quad \xi_{k,t} \sim \mathcal{N}(0, S)$$

Given the estimated latent states $\hat{z}_{k,t}$, the joint log-likelihood for a single trial k across all time points t is expressed as:

$$\log p(x_{k,1:T}|\hat{z}_{k,1:T}) = -\frac{1}{2}\sum_{t=1}^{T}\left[(x_{k,t} - D\hat{z}_{k,t})^\top S^{-1}(x_{k,t} - D\hat{z}_{k,t}) + \log|S| + N\log(2\pi)\right]$$

For all trials $k = 1, \ldots, K$, the average joint log-likelihood becomes:

$$\mathcal{L} = -\frac{1}{2}\sum_{k=1}^{K}\sum_{t=1}^{T}\left[(x_{k,t} - D\hat{z}_{k,t})^\top S^{-1}(x_{k,t} - D\hat{z}_{k,t}) + \log|S| + N\log(2\pi)\right]$$

To maximize $\mathcal{L}$ with respect to D, the terms involving D are isolated, and $\xi_{k,t} = x_{k,t} - D\hat{z}_{k,t}$ is substituted:

$$\mathcal{L}(\mathcal{D}) = \sum_{k=1}^{K}\sum_{t=1}^{T}(x_{k,t} - D\hat{z}_{k,t})^\top S^{-1}(x_{k,t} - D\hat{z}_{k,t}) = \sum_{k=1}^{K}\sum_{t=1}^{T}\xi_{k,t}^\top S^{-1}\xi_{k,t}$$

The gradient of $\mathcal{L}$ with respect to D is computed using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial D} = \frac{\partial(\sum_{k,t} \xi^\top S^{-1} \xi)}{\partial \xi} \frac{\partial \xi}{\partial D} = -\sum_{k=1}^{K}\sum_{t=1}^{T} S^{-1}\xi_{k,t}\hat{z}_{k,t}^\top = -S^{-1}\sum_{k=1}^{K}\sum_{t=1}^{T}(x_{k,t} - D\hat{z}_{k,t})\hat{z}_{k,t}^\top$$

Setting $\frac{\partial \mathcal{L}}{\partial D} = 0$ yields the optimal D:

$$D^\star = \left(\sum_{k=1}^{K}\sum_{t=1}^{T} x_{k,t}\hat{z}_{k,t}^\top\right)\left(\sum_{k=1}^{K}\sum_{t=1}^{T} \hat{z}_{k,t}\hat{z}_{k,t}^\top\right)^{-1}$$

The residuals $\xi_{k,t}$ are updated as:

$$\xi_{k,t} = x_{k,t} - D^\star \hat{z}_{k,t}$$

To optimise $\mathcal{L}$ with respect to S, terms involving S are isolated:

$$\mathcal{L}(\mathcal{S}) = \sum_{k=1}^{K}\sum_{t=1}^{T} \xi_{k,t}^\top S^{-1}\xi_{k,t} + KT\log|S|$$

The first term is expressed as a trace (since it is a scalar), enabling the use of cyclic properties of the trace:

$$\sum_{k,t}^{K,T} \xi_{k,t}^\top S^{-1}\xi_{k,t} = \sum_{k,t}^{K,T} Tr(\xi_{k,t}^\top S^{-1}\xi_{k,t}) = \sum_{k,t}^{K,T} Tr(S^{-1}\xi_{k,t}\xi_{k,t}^\top) = Tr(S^{-1}\sum_{k,t}^{K,T}\xi_{k,t}\xi_{k,t}^\top) = Tr(S^{-1}\Xi)$$

where $\Xi = \sum_{k,t}^{K,T}\xi_{k,t}\xi_{k,t}^\top$. Using the derivatives of $Tr(A^{-1}B)$ and $\log|A|$ (Appendix), and setting $\frac{\partial \mathcal{L}}{\partial S} = 0$:

$$\frac{\partial \mathcal{L}}{\partial S} = -S^{-1}\Xi S^{-1} + KTS^{-1} = S^{-1}(SKT - \Xi)S^{-1} = 0$$

$$S^\star = \frac{1}{KT}\Xi = \frac{1}{KT}\sum_{k=1}^{K}\sum_{t=1}^{T}\xi_{k,t}\xi_{k,t}^\top$$

Expanding $\xi_{k,t}\xi_{k,t}^\top$:

$$\xi_{k,t}\xi_{k,t}^\top = x_{k,t}x_{k,t}^\top - 2D^\star\hat{z}_{k,t}x_{k,t}^\top + D^\star\hat{z}_{k,t}\hat{z}_{k,t}^\top(D^\star)^\top$$

Substituting $D^\star$ simplifies the term:

$$S^\star = \frac{1}{KT}\left(\sum_{k=1}^{K}\sum_{t=1}^{T} x_{k,t}x_{k,t}^\top - D^\star\sum_{k=1}^{K}\sum_{t=1}^{T}\hat{z}_{k,t}x_{k,t}^\top\right)$$

2.2 Performance and Suitability of the Kalman filter-based Decoder

The Kalman filter-based decoder achieves a significantly higher R^2 value (0.725) compared to the simple ridge regression decoder (0.444). One reason for this is the Kalman filter's autoregressive prior, which incorporates temporal dependencies by using the current latent state to predict the next state. This allows the Kalman filter to model the smooth, evolving dynamics of time-series data, such as hand velocity trajectories, where each point is influenced by its predecessors. In contrast, ridge regression treats each time point independently and cannot leverage these temporal relationships.

The Kalman filter further benefits from its ability to explicitly model noise in both the dynamics and observations, allowing it to balance uncertainty in the neural data with prior knowledge of the system's behaviour. This leads to more reliable predictions compared to ridge regression, which only minimises a penalised error term without considering the complex noise structure. Finally, the Kalman filter's generative approach assumes that neural activity is generated by latent states related to hand velocity, capturing key aspects of the hand motion dynamics. Ridge regression, being a purely discriminative model, is less effective at generalising under noisy conditions. Thus, the Kalman filter's ability to model temporal dependencies, smooth trajectories, and account for noise results in its superior performance over ridge regression.

The suitability of the Kalman filter-based decoder for use in BMI applications can be assessed across the same three key factors:

1. **Feasibility:** The Kalman filter-based decoder is feasible for real-time BMI applications due to its recursive nature, which allows for continuous updates to predictions as new neural data becomes available. However, the need to pre-train a latent dynamical system and estimate model parameters adds some complexity compared to the simple decoder which could be a barrier to rapid deployment to new tasks or users.

2. **Computational Tractability:** The Kalman filter operates efficiently with recursive updates, making it computationally tractable for online use. Its reliance on a low-dimensional latent state ($z_{k,t} \in R^{10}$) ensures that matrix operations such as inversion remain manageable. It takes 15.55 seconds for the Kalman filter-based decoder to predict the hand velocities from the test neural data. It uses matrix multiplication, addition, and inversion to compute the posterior distribution of the latent state, which are more intensive than the single matrix multiplication required by the simple decoder.

3. **Accuracy:** The Kalman filter-based decoder achieved an R^2 of 0.725 when predicting hand velocities from neural data in the test trials, significantly outperforming the simple decoder. This improvement is due to the autoregressive prior in the LDS, which captures temporal dependencies and smooth, biologically plausible hand trajectories. The improved accuracy makes it highly suitable for tasks requiring precise movement decoding, although the increased complexity might not always justify the gains in simpler use cases.

2.3 Further Improvements

The linear Gaussian model of neural coding is an approximation, and while it provides a solid foundation for decoding, there may be potential to improve performance by using more complex models. More sophisticated likelihood models can capture non-linear relationships and account for more intricate patterns in the data, which may result in higher decoding accuracy. For example, generalised linear models (GLMs) or generalised additive models (GAMs) can offer a more flexible framework to model the neural data, potentially improving the fit to the observed neural activity.

However, more complex models come with drawbacks. They typically require greater computational resources, as they involve more parameters and are more computationally intensive to fit. Also, the risk of overfitting increases, especially with high-dimensional data or small sample sizes, where the model may capture noise rather than the true signal. These models also require careful tuning to ensure they generalise well to new data, as they can become overly specialised to the training set. Thus, while they can improve accuracy, their use must be balanced with considerations of computational cost and how well they generalise.

Another potential area for improvement is the noise model used to represent variability in neural firing. In this analysis, a standard Gaussian model was used, but exploring alternative models, such as the Poisson distribution, could enhance performance. Neural activity is often measured in terms of spike counts, which are discrete, non-negative values representing the number of times a neuron fires within a specific time window. Since the Poisson distribution is naturally suited for modelling count data, it could better capture the discrete and non-negative nature of neural firing. Incorporating a Poisson model might improve the accuracy of the decoder by providing a more appropriate representation of the underlying neural activity.

Appendix

Compute the partial derivative of tr$(A^{-1}B)$ with respect to A. Since B is constant with respect to A, we apply the derivative of the inverse of A, which is known to be:

$$\frac{\partial}{\partial A}A^{-1} = -A^{-1}A^{-1}$$

This result can be derived by implicitly differentiating the identity $AA^{-1} = I$, where I is the identity matrix. Differentiating both sides with respect to A:

$$\frac{d}{dA}(AA^{-1}) = \frac{d}{dA}I$$

The right-hand side is zero, and applying the product rule to the left-hand side, we get:

$$\frac{d}{dA}(AA^{-1}) = \frac{dA}{dA}A^{-1} + A\frac{d}{dA}A^{-1}$$

Since $\frac{dA}{dA} = I$, this simplifies to:

$$A^{-1} + A\frac{d}{dA}A^{-1} = 0$$

Solving for $\frac{d}{dA}A^{-1}$, we obtain:

$$\frac{d}{dA}A^{-1} = -A^{-1}A^{-1}$$

Using this, we compute the derivative of $\mathrm{tr}(BA^{-1})$ with respect to A. Applying the chain rule:

$$\frac{\partial}{\partial A}\mathrm{tr}(A^{-1}B) = \frac{\partial}{\partial A}\mathrm{tr}(BA^{-1}) = -A^{-1}BA^{-1}$$

Compute the derivative of $\log(\det(A))$ with respect to the matrix A. We apply the chain rule for the derivative of the logarithm. First, the derivative of $\log(x)$ with respect to x is:

$$\frac{d}{dx}\log(x) = \frac{1}{x}$$

Thus, the derivative of $f(A) = \log(\det(A))$ with respect to A is:

$$\frac{\partial}{\partial A}\log(\det(A)) = \frac{1}{\det(A)} \cdot \frac{\partial}{\partial A}\det(A)$$

From matrix calculus, it is known that the derivative of $\det(A)$ with respect to A is:

$$\frac{\partial}{\partial A}\det(A) = \det(A) \cdot A^{-1}$$

Substituting the result for $\frac{\partial}{\partial A}\det(A)$ into the chain rule expression, we get:

$$\frac{\partial}{\partial A}\log(\det(A)) = \frac{1}{\det(A)} \cdot \det(A) \cdot A^{-1}$$

The $\det(A)$ terms cancel out, leaving:

$$\frac{\partial}{\partial A}\log(\det(A)) = A^{-1}$$