Learning Dynamical Models of Motor Cognition

Libby Zhang Riley DeHaan

January 13, 2019

In this project we explore reflex- and state-based models of internal neural activity and external movement and examine motor cortex neural activity recorded during reaching tasks to extract salient features that can be used to describe the system. In this report, we describe 1) a linear state-based model of the system, 2) linear quadratic estimation of motion based on a linear model between kinematic and neural responses, 3) recurrent neural network model, and 4) error analysis and discussion of each of these models.

1 Related Work

The theory of the somatotopically organized cortex, which assumes a one-to-one correspondence of cortical area to behavioral function, was discarded in the mid-1990's and early 2000's after mounting evidence that cortical processes were far more distributed, both physically and functionally (Sanes et al. 1995) than somatotopical models would suggest. Even concrete cortical processes, such as motor output, for which there is a more easily quantifiable behavioral output (versus attention and decision-making, for example), were increasingly understood to be more complex than originally thought. Advances in multi-electrode intracortical arrays allowed for the simultaneous measurement of the spiking activity of neuronal populations, and the temporal complexity and heterogeneity of single-neuron and multi-neuron responses (Churchland and Shenoy 2007) supported the view of neural responses driving movement through their own dynamics.

One approach to investigating the neural dynamics of movement is to build computational models of how neural population responses relate to kinematic movements. Initial attempts focused on linear decoders, and the more successful of these attempts employed different types of Bayesian inference. Wu et al. pioneered the use of a Kalman filter to decode neural responses in 2003. This approach has been expanded and further refined for neural decoding in the subsequent years and have been applied with success to patients with neuroprosthetics/brain-machine interfaces (BMI) (e.g. Hochberg et al. 2006, Gilja et al. 2012).

To better model the complex non-linear dynamics and feedback believed to be present in biological neural networks, we have additionally developed a recurrent neural network (RNN) model with and without feedback of the current prediction of kinematic state. Other authors have applied variants of RNNs successfully to the subject of BMI reach task decoding (Sussillo, 2012), demonstrating the potential for this model in our task. The RNN is an artificial neural network commonly employed for complex sequential modeling tasks such as natural language translation and handwriting recognition (Lipton et. al., 2015). In the RNN, observations are sequentially input to the network and used to generate an output of a hidden context vector representing the state of the network. This hidden context vector is recursively taken as an additional input to the network in calculating future hidden contexts, thereby producing an internal network feedback connection allowing for complex sequential modeling. Of particular interest is the LSTM, an RNN variant developed by (Hochreiter et. al. 1997) to address problems of vanishing and exploding gradients. The LSTM has two primary features. First, a recurrent "cell" (representing the internal state of the network) is updated by a combination of the input and the cell values of the previous time step. The cell is essentially directly connected to the computation of future cell values to allow for gradient information to pass back through the network. Second, the LSTM has "gates" for allowing the network to control information flows throughout the network. Most notably, the forget gate controls the degree to which previous cell state values are passed to future cell states, allowing the network to "forget" previous states over time (Lipton et. al., 2015). The LSTM then was chosen for our modeling task as a standard RNN with relatively easy training.

2 Data Collection and Preprocessing

Kinematic position and neural spiking data were simultaneously collected from a behaving rhesus macaque while the subject performed a trained reaching task (Fig. 1). During this task, known as center-and-out, the

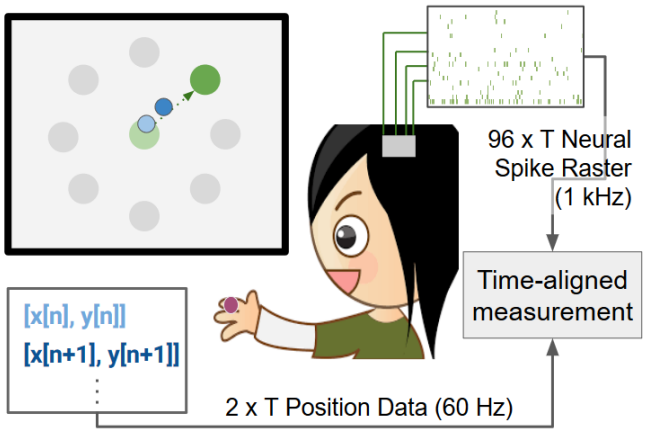

Figure 1: Schematic of experimental setup. Subject controlled on-screen cursor position (blue) with infrared bead on finger (magenta). Position data was collected at a 60 Hz sampling rate. Task objective was to move cursor from old target location (light green) to new target location (dark green). Other possible target locations are shown in dark grey. An intracortical electrode array collected neural spiking data at a 1 kHz sampling rate.

subject sits in front of a monitor and uses his hand position in free space to control a cursor on the monitor. Kinematic position is sampled at 60 Hz. Targets are presented in the center and then in one of eight radial locations. The subject must hold the cursor over the target for 500 ms. Neural population activity was recorded during these activities with a chronically-implanted 96 microelectrode array situated in the motor region of the cortex. The electrodes sampled local neural activity passed through a low-pass filter at a 1 kHz rate to determine spiking activity. Each trial consists of a single reach to the new target. There are 527 trials, with at least 32 trials per trajectory (average trial time to target acquisition was 1.2 0.2s). Data was collected by the Shenoy lab.

Kinematic and neural data were binned into 50 ms intervals prior to model training. This bin width was chosen to avoid data aliasing (see Appendix C, Fig. 8) and was limited by the kinematic position sampling rate. Velocity was calculated as first-order difference between positions at designated bin width. Neural data was binned by counting the number of spikes that occurred within the window. Data may also be binned by using a sliding window (e.g. width = 50 ms, step = 25 ms) to include more data points and to account for the continuous nature of the data, but this processing method was not explored in this project.

3 Linear Dynamical Model with Gaussian Noise

3.1 Hidden Markov Model

The first model that we explore is a hidden Markov model relating motion kinematics and neural observations (Fig. 2. Since our objective is to decode intended cursor position from neural responses, the kinematic state is the hidden variable and the neural response are the measured observations from the model. We define our state $x \in \mathbb{R}^n$ as the kinematic values of position and velocity and observations $y \in \mathbb{R}^m$ as neural spiking measurements from the 96 electrode array at timestep k to be:

$$\begin{aligned} x[k] &= \left(p_x,\ p_y,\ v_x,\ v_y\right) \\ y[k] &= \left(n_1,\ n_2,\ \ldots\ , n_m\right) \end{aligned} \tag{1}$$

for $n = 4$ states and $m = 96$ neural response measurements. We propose an undriven linear state-space model subject to Gaussian noise of the form:

$$\begin{aligned} x[k+1] &= A\,x[k] + w[k] \\ y[k] &= C\,x[k] + q[k] \end{aligned} \tag{2}$$

with state evolution matrix $A \in \mathbb{R}^{n\times n}$, observation matrix $C \in \mathbb{R}^{m\times n}$, state noise process $w[k] \sim \mathcal{N}(0, W)$, and observation noise process $q[k] \sim \mathcal{N}(0, Q)$. $W \in \mathbb{R}^{n\times n}$ and $Q \in \mathbb{R}^{m\times m}$ are the respective covariance matrices of the signals. Our modeling objective is therefore to extract the features of our neural-kinematic system, namely, the matrices A, C, W, and Q.

Given time-aligned neural data $Z \in \mathbb{R}^{m\times T}$ and kinematic motion data $X \in \mathbb{R}^{n\times T}$, where T is trial length

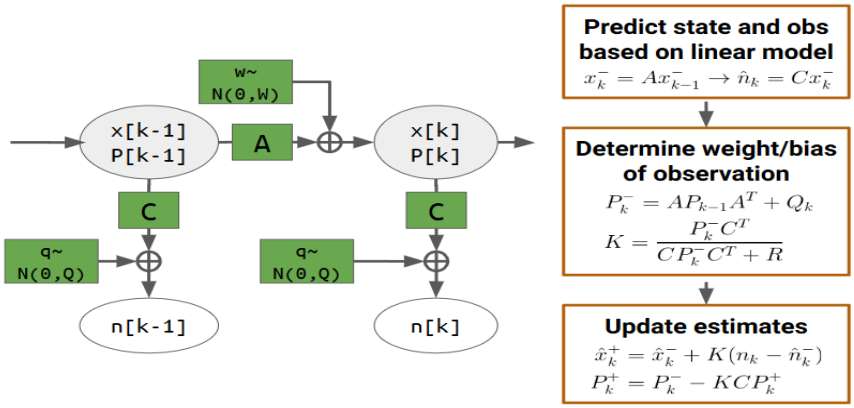

Figure 2: (*left*) Hidden Markov model of motor cognition. We would like to infer the hidden kinematic state (shaded oval) and associated covariance matrix at each timestep given the observed neural state (white oval). We assume a time-invariant linear-Gaussian model which governs the hidden state evolution with time and which relates state to neural observation. Matrices of the linear-Gaussian model (in green rectangles) are extracted from training data. (*right*) One-timestep snapshot of Kalman filter algorithm for predicting hidden states

in unit timesteps, we can extract the salient features by solving for the noiseless system, and then account for the difference as our sample noise:

$$\begin{aligned} A = x[k+1] \backslash x[k] \;&\Rightarrow\; A = X[:, 1:] \backslash X[:, 0:-1] \\ C = y[k] \backslash x[k] \;&\Rightarrow\; C = Z\, X^{+} = Z\, X^{T}(XX^{T})^{-1} \\ \mathbf{w} = X[:, 1:] - A\, X[:, 0:-1] \;&\rightarrow\; W = \frac{1}{T-1}\, \mathbf{w}\, \mathbf{w}^{T} \\ \mathbf{q} = Y - C\, X \;&\rightarrow\; Q = \frac{1}{T}\, \mathbf{q}\, \mathbf{q}^{T} \end{aligned} \tag{3}$$

3.2 Linear Quadratic Estimation

We now use linear quadratic estimation to infer kinematic movement from neural responses using our linear system model. The Kalman filter causally estimates the current state $\hat{x}_{k\,|\,k}$ from the current measurement z_k. The algorithm consists of three steps: 1) *a priori* prediction, 2) compute optimal gain given residual between measurement and estimation, and 3) update state estimation to minimize mean square error. Figure 2b provides a diagram of the algorithm, and a more detailed treatment of the algorithm is given in Appendix A.

3.3 Model Variations

Specifically, three different models were used: the position Kalman filter (PKF), the velocity Kalman filter (VKF), and the position-velocity Kalman filter (PVKF). These models differ in the underlying linear-Gaussian model used.

The constituent matrices seen below are calculated as in Eqn. 3 from position and/or velocity data. The same neural data was used. Specifically, $A_p \in \mathbb{R}^{2\times 2}$, $W_p \in \mathbb{R}^{2\times 2}$, $C_p \in \mathbb{R}^{96\times 2}$, and $Q_p \in \mathbb{R}^{96\times 96}$ were calculated from the position data and the neural data. A_v, W_v, C_v, and Q_v, which have the same respective dimensions, were calculated from the velocity data and the neural data.

Position Kalman Filter

$$A_{pkf} = \left(\begin{array}{c|c} A_p & 0 \\ \hline & 0 \end{array}\right), \qquad W_{pkf} = \left(\begin{array}{c|c} W_p & \\ \hline & 0 \end{array}\right)$$

$$C_{pkf} = \left(\begin{array}{c|c} C_p & 0 \end{array}\right), \qquad Q_{pkf} = Q_p$$

Velocity Kalman Filter

$$A_{vkf} = \left(\begin{array}{c|c} I & \begin{matrix}\Delta t & \\ & \Delta t\end{matrix} \\ \hline & A_v \end{array}\right), \qquad W_{vkf} = \left(\begin{array}{c|c} 0 & \\ \hline & W_v \end{array}\right)$$

$$C_{vkf} = \left(\begin{array}{c|c} 0 & C_v \end{array}\right), \qquad Q_{vkf} = Q_v$$

Position-Velocity Kalman Filter

$$A_{pvkf} = \left(\begin{array}{c|c} A_p & \begin{matrix}\Delta t & \\ & \Delta t\end{matrix} \\ \hline & A_v \end{array}\right), \qquad W_{pvkf} = \left(\begin{array}{c|c} W_p & \\ \hline & 0 \end{array}\right)$$

$$C_{pvkf} = \left(\begin{array}{c|c} C_p & C_v \end{array}\right), \qquad Q_{pvkf} = Q_p$$

4 Long-short Term Memory Model

The RNN model was chosen to model the proposed non-linear dynamics and feedback of the neural population activity. Our networks were implemented in Keras and consist of an LSTM layer followed by a fully-connected layer to output our kinematic state predictions of position and velocity. Two of these networks were developed. The first network predicts the cursor position and velocity from input of the 96-dimensional neural population activity observations and is shown in Figure3. The second LSTM network was additionally given inputs of the current kinematic predictions of the network at the current time step in predicting the kinematic state at the next time step (in addition to the usual recurrent connections of LSTM layers). For the first time step (prior to any predictions being made), this second LSTM with kinematic feedback was given the true initial cursor position and velocity for its kinematic inputs. Beyond this additional feedback, these two networks had identical features. Both LSTM layers contained 250 neurons. The hyperbolic tangent was used for the output and recurrent activations of both LSTM layers, and a linear activation was used for the activation of both fully-connected layers for regression. The networks were trained with mean squared error and the ADAM optimizer, which were found to optimize the network consistently during the model development process for this particular application.

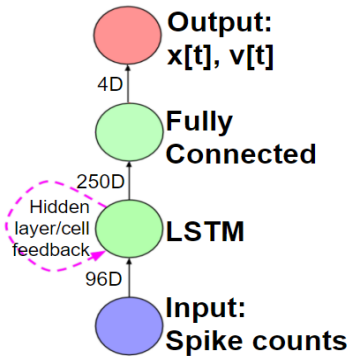

Figure 3: RNN architecture without kinematic feedback. Spike counts are input to the LSTM layer, which outputs a 250-dimensional hidden context vector to a fully-connected layer for prediction of cursor position and velocity. Modified from (Lipton et. al., 2015).

Several hyperparameters were tuned during the development process (discussed further in the Analysis section alongside the validation process for the Kalman filters). The RNN without kinematic feedback achieved optimal performance on the validation set with early stopping at eighteen epochs while the RNN with kinematic feedback achieved optimal performance with sixty epochs. One feature to note is that while the RNN without feedback was able to be trained on batch sequences of complete reach tasks, the RNN with feedback was trained on individual time samples (i.e. each observed point within a reach task) due to Keras

API limitations. This feature likely contributed additional noise to the training process for the RNN with feedback. No dropout was found to improve performance for the RNN without feedback, however dropout of 0.5 on the input to the fully-connected layer was necessary for stable training of the RNN with kinematic feedback.

5 Analysis

The output data consisted of predicted positions and velocities at each timestep, given a sequence of neural spike counts. Inference algorithms were initialized with the start position (old target position) with each new test. Position error was calculated at each timestep as the least-squares distance from the target position. The center-out task used in this experiment placed peripheral targets at a distance of 120 mm from the center target. Velocity angle error was also calculated at each timestep, using the absolute difference between the current angle and the target base angle. The target base angle is defined as the angle from the old target position to the new target position. For example, if the old target position was at the center and the new target position northeast of the center, the base angle would be 45^o. If the old target position was in the south position and the new target position was in the center, the base angle would be 90^o. The position and velocity angle errors were calculated for both the subject (true) data and estimated data, and subsequently compared.

Note that the estimated kinematics were not directly compared to the true motion kinematics (especially with respect to position) because we are not trying to directly reproduce motion trajectory. As mentioned in Section 1, it is now understood that neural responses do not directly represent kinematic variables (e.g. position or velocity) but instead produce and are governed by complex dynamics that drive physical movement downstream. Additionally, subjects typically do not move the cursor to the new target in a straight line, due to musculoskeletal preferences. A concrete example is as follows: if moving from the center to the northeast target (45^o position), and the subject is controlling the cursor with their right limb, it may be more efficient to pivot from the elbow joint (generating an arc movement at the finger tips, where the sensor is located), rather than shifting the whole arm from the shoulder joint (generating a straight line). These considerations led us to calculate position error (for analysis) as distance to target and velocity angle as difference from base angle.

Due to the limited size of the dataset (526 total trials), k-folds cross-validation was used for the error analysis. Input data was truncated to the shortest input sequence so that mean and standard error at each time step could be calculated across trials. K-folds cross-validation was used for all Kalman filter testing. Cross-validation was not used in RNN models due to the larger training time required ($\sim$82 minutes for one simulation of LSTM with kinematic feedback, 60 epochs, 320 training trial examples). Instead, approximately 60% of the available dataset (320 of 526 samples) was used for training, 80 samples (20% of the total data used for model training and development) were used for a development set, and 125 samples were used to test the model.

6 Results

6.1 Kalman Filter

The linear, undriven model of motor dynamics performs surprisingly well considering its simplifying assumptions and model. The various Kalman filter variations have differing degrees of complexity to allow for assessment of the linear model. As expected, the PKF, which assumed that neural responses were linearly correlated with hand position, performed poorly both qualitatively and quantitatively (figures shown in Appendix C). The VKF and PVKF, which assumed some neural response correlation with hand velocity, approximately captured the task objective.

Examining the model errors (Fig. 4), the VKF minimizes the mean square error of velocity angle, and predicted velocity angle (orange) closely follows the velocity angle error of typical subject movements (blue). The PVKF, which incorporates observation correlation and process noise source derived from position data into the linear-Gaussian model, is able to decrease position error initially, relative to the position error the VKF, but at the cost of velocity angle error during the middle portion of the trajectory.

Qualitatively, both the VKF and PVKF generally predict motion in the correct direction of the target. The PVKF, when able to move towards the target, is able to approach the target position more accurately than the VKF (Figure 5, bottom panels (a) and (c), selected high-performing example trajectories shown in Appendix C, Figure 11). However, at times the initial velocity angle error is greater than 90^o and the model

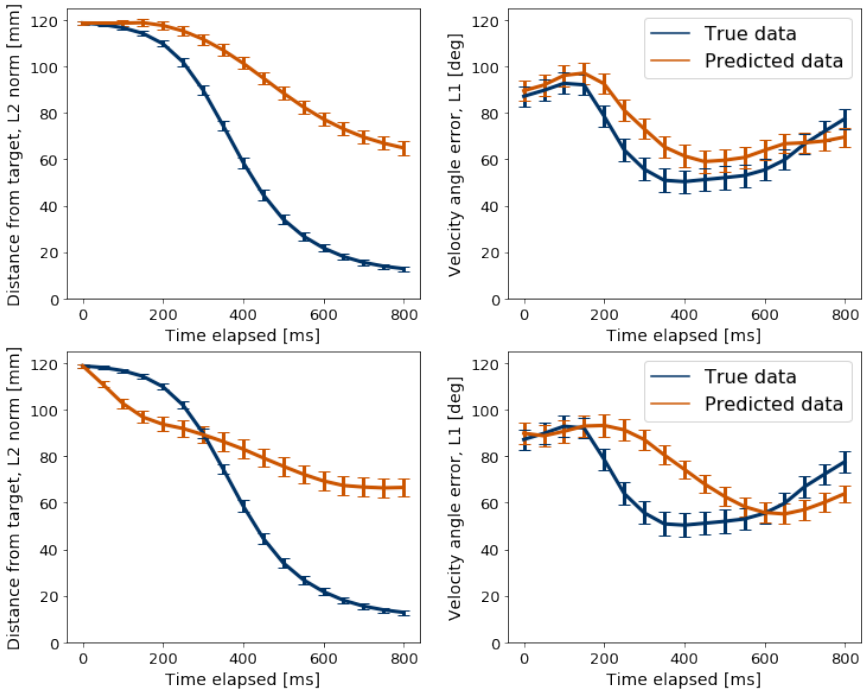

Figure 4: Position and velocity angle error of the VKF (*top*) and the PVKF (*bottom*).

is unable to correct (Figure 5, bottom panels (b) and (d)). While the VKF does not perform as well as the PVKF (Figure 5, bottom panels (c) and (b)), it also does not perform as poorly (Figure 5, bottom panels (a) and (d)). The final mean position error for the subject is 12.83 ± 0.98, but the final mean position errors for the Kalman filters are significantly greater (VKF: 64.89 ± 2.92, PVKF: 66.55). Note that this final mean position error is half the initial error, which is consistent with its binary performance.

Averaging the error across the entire motion trajectory (Appendix B, Table 2) obscures these parametric dependencies, but provides an efficient summary of each model's performance. Briefly, we find that the estimated VKF and PVKF velocity angle to base angle errors, averaged across the entire trajectory, are not significantly different from the subject angle error. The position error produced by all Kalman filters are significantly greater than the subject position error across the trajectory, but both the PKF and PVKF have significantly less position error than the VKF.

The Kalman filters presented here assume a time-invariant linear correlation between neural observations and kinematics. We did not attempt to model some features that likely played a large role in the subject's behavioral prevision and the neural responses. For example, subjects were required to hold the cursor at the target position for 500 ms, which is a goal-oriented action distinct from moving the cursor towards the target. Additionally, subjects had visual feedback of cursor position throughout the duration of each trial, enabling them to more precisely control cursor position. While one might suggest that the training data encodes a degree of this feedback, the linear model as described in Eqn. 2 is not able to capture this feature.

6.2 LSTM Networks

The LSTM RNN was chosen as a non-linear sequential model with the expressivity to capture these features of feedback. Our LSTM network kinematic predictions were generally noisier than those of the Kalman filters, as may be seen in the plots of position predictions given in Figure 6 for the RNN without kinematic feedback (top) and with kinematic feedback (bottom). In particular, the predictions of position by the RNN with no feedback show significant noise, with predicted position varying widely around the path of true motion. The predictions of position of the RNN with feedback are significantly less variable, but still far noisier than the predictions made by the Kalman filters. This noise can be attributed to the non-linear complexities of RNNs and the difficulties associated with training these models. The LSTM was chosen as a standard model that avoided many of the common issues associated with RNNS. However, the training

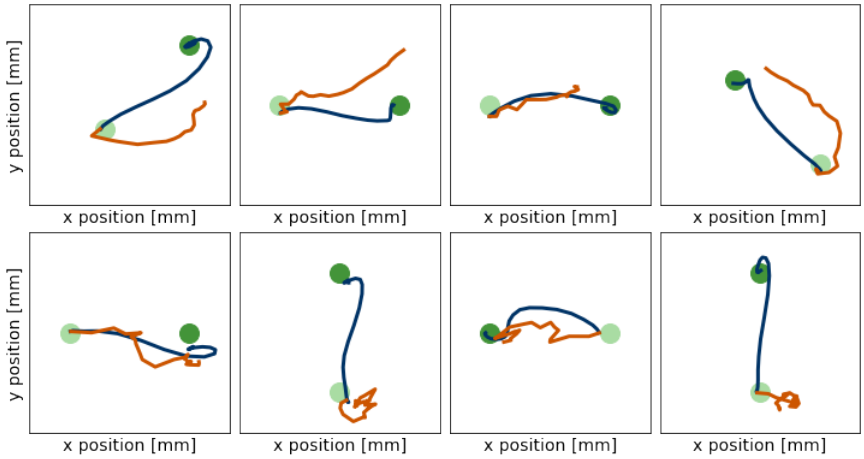

Figure 5: Randomly selected examples of subject trajectory (blue) reaching from old target (light green) to new target (dark green), and of simulated trajectories (orange) produced by VKF *top* and PVKF *bottom*

process for these networks is challenging and still an active area of research. Interestingly, even given the variability present in the individual motion predictions, the distance from target and mean velocity angle error over time of the RNN without feedback follow those of the true motions fairly closely (Figure 7). The RNN with feedback actually has generally higher angle error than the true results and the RNN without feedback, and later on in the motion trajectories, the RNN with feedback tends to undershoot the target as seen in the example position predictions although the overall quality of the motions predicted with feedback are significantly better than those predicted without feedback (Figure 6). Generally the predictions of both RNN networks show significant variability, but are still highly correlated to the true reach motions.

Another interesting feature which may be seen in the reach motion plots of the RNN with feedback is the tendency for the predictions at the end of the motions to turn and cluster, in line with the way the true motions tend to slow down as the target is approached. We hypothesize that these features of the RNN predictions show evidence of the RNN having learned features of non-linear feedback in the neural population activity to some degree, with the RNN "reacting" in a qualitatively similar way as the test subject did during the reach tasks in correcting its motion as the target is approached by turning and then attempting to hover around the target once reached. Thus it is possible the RNN approach we have taken has partially encompassed the non-linear feedback dynamics of interest in the neural population activity.

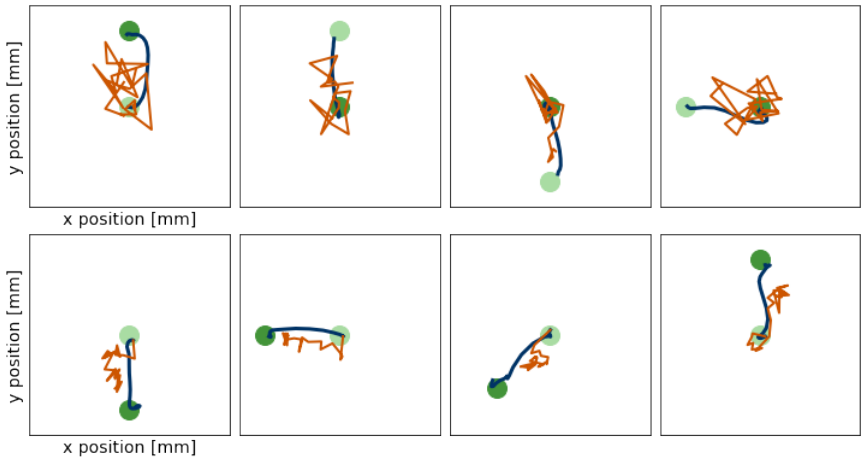

Figure 6: Examples of cursor position predictions produced by RNN model (orange) with no kinematic feedback (top) and with kinematic feedback (bottom, relatively high performance), compared to true trajectory (blue). Trajectories are directed from the light green circle to the dark green circle.

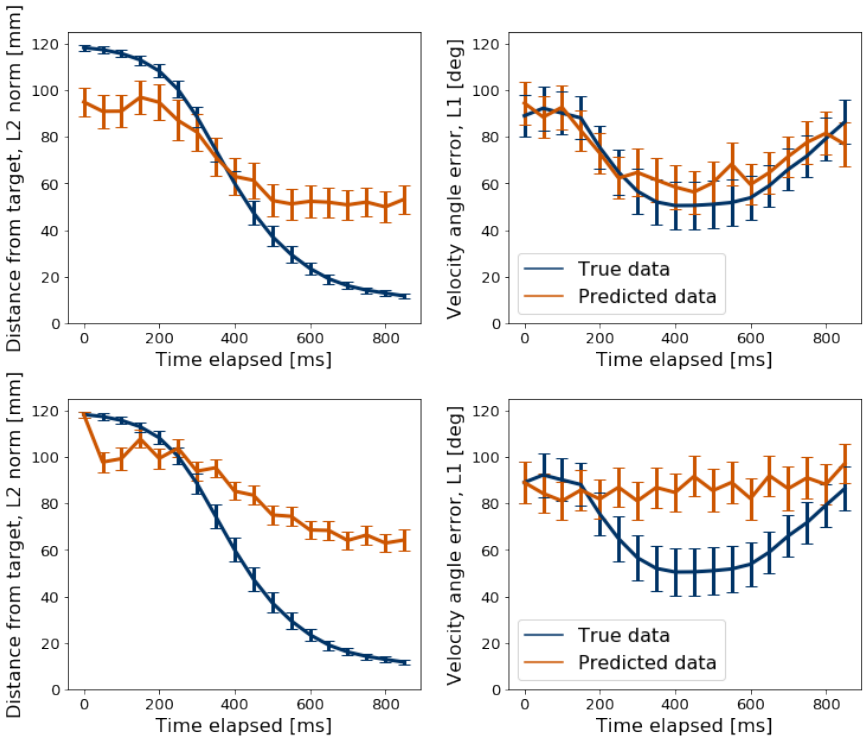

Figure 7: Mean distance from target (left) and mean velocity angle error (right) for the RNN model without kinematic feedback (top) and with kinematic feedback (bottom).

7 Conclusion

We have presented a linear model and a RNN model of the neural dynamics underlying motor cognition. Our linear model performs best when neural observations are correlated with velocity, and perform well on about half of the motion trials with position correlation. However, this model overlooks the feedback and task goal components that play a significant component in the subject's (true system) performance. The RNN model was chosen to model these drivers of the reach motions, and with kinematic feedback was found to have learned some qualitative features of the feedback underlying the motion.

8 References

Sanes, Jerome N., John P. Donoghue, Venkatesan Thangaraj, Robert R. Edelman, and Steven Warach. "Shared neural substrates controlling hand movements in human motor cortex." Science 268, no. 5218 (1995): 1775-1777.

Churchland, M.M. and Shenoy, K.V. (2007) Temporal complexity and heterogeneity ofsingle-neuron activity in premotor and motor cortex. J. Neurophysiol. 97, 42354257

Wu, W., Black, M. J., Gao, Y., et al. Neural decoding of cursor motion using a Kalman filter. (2003) Advances in neural information processing systems (pp. 133-140).

Hochberg, Leigh R., Mijail D. Serruya, Gerhard M. Friehs, Jon A. Mukand, Maryam Saleh, Abraham H. Caplan, Almut Branner, David Chen, Richard D. Penn, and John P. Donoghue. "Neuronal ensemble control of prosthetic devices by a human with tetraplegia." Nature 442, no. 7099 (2006): 164

Gilja, Vikash, Paul Nuyujukian, Cindy A. Chestek, John P. Cunningham, M. Yu Byron, Joline M. Fan, Mark M. Churchland et al. "A high-performance neural prosthesis enabled by control algorithm design." Nature neuroscience 15, no. 12 (2012): 1752.

Jolliffe I.T. and Cadima, J. Principal component analysis: a review and recent developments. Phil. Trans. R. Soc. A 374: 20150202. (2016).

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Marks, R.J. and Reed R. Neural Smithing. Supervised Learning in Feedforward Artificial Neural Networks. MIT Press. (1999).

Lipton Z., et al. A Critical Review of Recurrent Neural Networks for Sequence Learning. 2015.

Hochreiter S., et al. Long short-term memory. Neural Computation, 1997.

A Linear Quadratic Estimation Algorithm

Algorithm 1: Kalman Filter

input : z_k, $\hat{x}_{k-1\,|\,k-1}$, $P_{k-1\,|\,k-1}$
output: $\hat{x}_{k|k}$, $P_{k\,|\,k}$

Predict *a priori* state estimate and error covariance given on system model and noise
$\hat{x}_{k\,|\,k-1} \leftarrow A\,\hat{x}_{k-1\,|\,k-1}$
$P_{k\,|\,k-1} \leftarrow A\,P_{k-1\,|\,k-1}\,A^T + W$

Compute *a priori* residual and covariance and optimal gain
$r_{k\,|\,k-1} \leftarrow z_k - C_k\,\hat{x}_{k\,|\,k-1}$
$S_{k\,|\,k-1} \leftarrow C\,P_{k\,|\,k-1}\,C^T + Q$
$K \leftarrow P_{k\,|\,k-1}\,C^T\,S^{-1}_{k\,|\,k-1}$
$\Delta \leftarrow (\mathbf{I} - K\,C)$

Update state estimate and error covariance
$\hat{x}_{k\,|\,k} \leftarrow \hat{x}_{k\,|\,k-1} + K\,r_{k\,|\,k-1}$
$P_{k\,|\,k} \leftarrow \Delta P_{k\,|\,k-1}\,\Delta^T + K\,W K^T$

B Summary statistics of error analysis

Table 1: Summary error statistics of different Kalman filter variations operating on test data. Position error is the L2 norm of distance to target position. Velocity angle error is L1 norm from base angle to target position. Values are (mean ± 2 x SE) across all k-folds and across trajectory.

Model	Position Error (L2)	Velocity Angle Error (L1)
True	64.16 ± 1.43	67.52 ± 4.64
PKF	86.85 ± 3.19	90.17 ± 4.72
VKF	98.83 ± 2.23	74.28 ± 4.48
PVKF	84.56 ± 3.30	74.75 ± 4.29

Table 2: Summary error statistics of different Kalman filter variations operating on test data. Position error is the L2 norm of distance to target position. Velocity angle error is L1 norm from base angle to target position. Values are (mean ± 2 x SE) across all k-folds and across trajectory.

Model	Position Error (L2)	Velocity Angle Error (L1)
True	61.54 ± 2.77	68.27 ± 9.46
RNN (no fb)	69.31 ± 7.09	71.94 ± 9.16
RNN (w/ fb)	84.84 ± 3.95	86.96 ± 8.72

C Additional figures

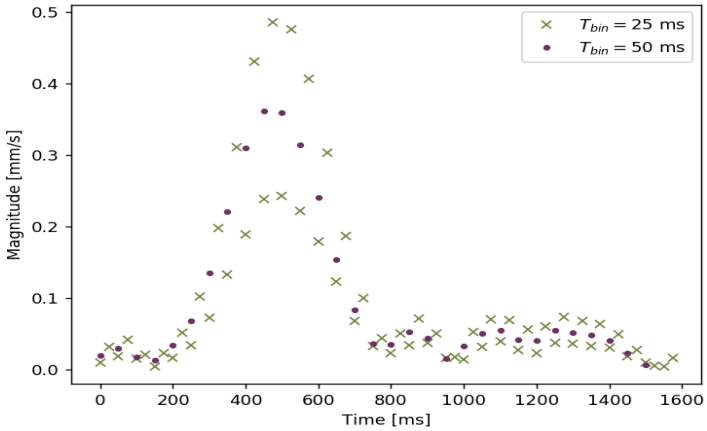

Figure 8: Aliasing of kinematic data seen at $tbin = 25$ ms but not at $tbin = 50ms$

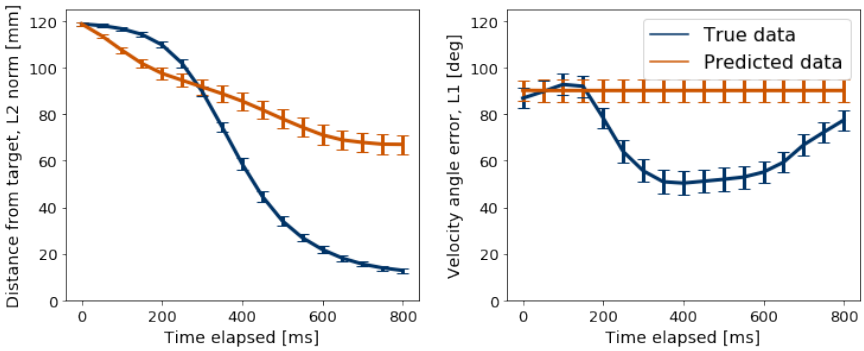

Figure 9: Position *(left)*and velocity angle *right* error of PKF simulated trajectories.

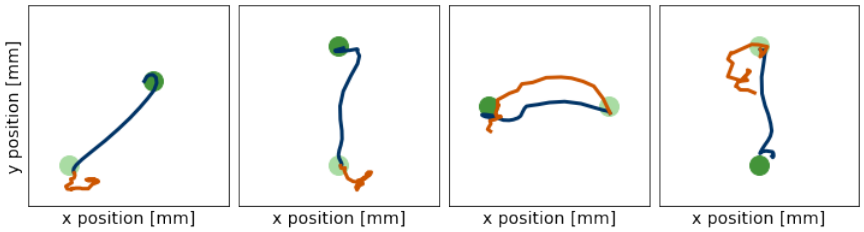

Figure 10: Examples of subject trajectory (blue) and PKF simulated trajectories (orange).

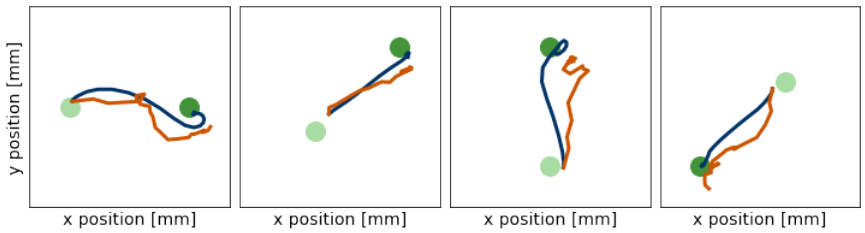

Figure 11: Examples of PVKF simulated trajectories (orange) performing precisely with respect to reaching target position (dark green).

D Lower-Dimensional Representation

An earlier component of our project involved determining meaningful reduced representations of the neural data. In the end, this direction of the project was dropped in favor of exploration of the RNN models. However, we include our previous work on PCA dimensionality reduction for neural signal representation here.

Biological and neural responses tend to be redundant and distributed, so we sought to explore whether lower-dimensional representations would still allow inference algorithms such as the Kalman filter to accurately predict cursor positions. Principle component analysis (PCA) is one such technique for dimensionality reduction. It reduces n-dimensional data to $K < n$ set of vectors which preserve the greatest source of variance in the data. PCA projects a given dataset onto an orthogonal basis of eigenvectors of the empirical covariance matrix of the data. More generally, singular value decomposition is similarly used to produce the principle components (PCs) instead of matrix diagonalization.

PCA receives a concatenation of observation vectors, $X \in \mathbb{R}^{m \times T}$, where this project has $m = 96$ recording electrodes and T is trial length. We desire to represent each observation x as a linear combination of the unit vectors p_j:

$$x_i = \sum_{j=1}^{k} \alpha_{ij} p_j \in \mathbb{R}^n \tag{4}$$

Additionally, we desire the first PC to preserve the maximal variance in the data:

$$p_1 = arg \max_{p:||p||=1} \frac{1}{m} p^T X^T X p = arg \max_{p:||p||=1} p^T \Sigma p \tag{5}$$

Using Lagrangian multipliers, we can show this is an eigenvalue problem $\Sigma p_1 = \lambda p_1$. This problem may be solved by the randomized power method for computing eigenvectors. Initialize p_1 to a random vector $p_1^{(0)}$ and iterate until convergence:

$$p_1^{(t+1)} \leftarrow \frac{\Sigma p_1^{(t)}}{||\Sigma p_1^{(t)}||} \tag{6}$$

which may be intuitively seen to converge given multiplication of p_1 by Σ iteratively amplifies the principal eigenvector component of Σ in p_1. Further PCs p_k can be computed by subtracting out the previously computed components and applying the power method again.

The main parameter to tune for PCA is the number of PCs to include in the reduced representation. Each PC "explains" some of the variance of the observations in the data set. Removing PCs reduces the data dimensionality, but also reduces the amount of information represented. Typically, data can be fiducially represented by a fraction of its dominant PCs.

We applied PCA (as implemented in the scikit-learn library) to the experimental neural spike counts. The resulting PCs were input to the VKF model described above, and estimated cursor position was compared to the true positions (Fig. 13). Dimensionality of 6 was found to be optimal both with a sum of square errors analysis $\sum(x_i - \hat{x}_i)^2$ computed on a 30 trial test set as a function of dimensionality, $k : \{1, \ldots, 96\}$ and with a direct examination of the PCA spectrum or proportion of explained variance per PC (magnitude of normalized eigenvalues) (Fig. 12).

Overall, the positions inferred from lower dimensional neural data were significantly less accurate than with full measurements. Commonly, the resulting position would have a similar qualitative form but veered off at either 90^o or 180^o from the true motion. These results suggest that PCA dimensionality reduction is removing some critical non-linear relations or else may be adding ambiguity to the direction computed by the Kalman filter (the PCs are not unique and in particular, may be negated without affecting the variance explained by the components). Higher dimensional representations (~ 60) were found to often give better qualitative results than 6 components.

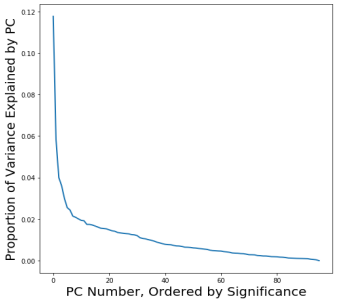

Figure 12: Proportion of variance *vs.* PC number.

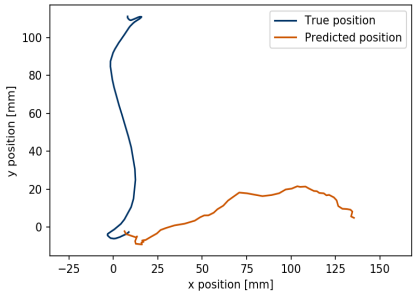

Figure 13: Representative plot of the PCA-reduced state estimate *vs.* actual motion.

PCA may not have been the correct representational model to choose for this particular data set and reduction problem, but it is also possible that (given the successful use of PCA in other similar applications within neuroscience) our application of PCA needs further optimization. One idea for improvement may include a better means of representing the dynamics of the neural data (the data does not represent a static distribution, but the neural activity over the course of an action in time) as discussed in Churchland et al. in the extension to PCA of jPCA, which may give a better dynamic representation of the data.