Dimensionality Reduction of Motor-related Neural Activity

Joline Fan, Jonathan Kao, Paul Nuyujukian

1 Introduction

Brain-computer interfaces (BCI) translate neural activity from a subject's cortex into useful control signals. In an online, closed-loop framework, the subject can therefore control a computer cursor by instructing movements, since this instruction modulates neural spiking in the brain. In this project, we achieve two goals. First, we implement a new real-time neural network decoder that learns the relationship between cursor kinematics and spiking. Second, we analyze neural activity under BCI control using dimensionality reduction techniques and analytical modifications.

Advances in decoder design have pushed prosthetic-cursor performance towards that of native arm movement. Most recently, these systems have been based on the Kalman filter [1]. Because neural spiking is non-linear with respect to kinematics, we investigate the feasibility of a non-linear network – a recurrent and randomly connected neural network – for real-time decoding [2].

Understanding the dynamics of the neural activity under brain-control is also important for informing better decoder designs. Typically, the analysis of neural activity involves the averaging of firing rates across many trials in order to reduce spiking variability and unveil the true firing rate of a neuron. However, this operation may remove important temporal information such as meaningful firing rate changes during a trial. To this end, dimensionality reduction techniques can be used to find structure in the neural data while maintaining temporal information [3]. We therefore examine neural data using unsupervised learning algorithms including principle components analysis (PCA), factor analysis (FA), and Gaussian process factor analysis (GPFA).

2 Methods

2.1 Task setup

Experiments were conducted using rhesus macaques following a Stanford IACUC approved protocol. The monkeys perform a center-out task, where 8 targets are positioned uniformly around a circle with a radius of 8 cm. One of the 8 targets around the circle is selected randomly and lit on the screen. The monkey then reaches to the target and holds the cursor for 500 ms within a 2.5 cm window. During training, the monkey's hand position determines the kinematics of an onscreen cursor. Signals in motor cortex are simultaneously measured from an electrode array while the monkey makes reaches to the instructed targets. These two sources of data are passed to a supervised learning algorithm to fit a Kalman filter (least squares) and/or neural network decoder (FORCE learning/recursive least squares). These decoders use the monkey's neural state to predict the on-screen cursor position. After fitting a model, the monkey engages in the same reach task; this time, however, the cursor position is controlled by spiking activity in motor cortex instead of hand position.

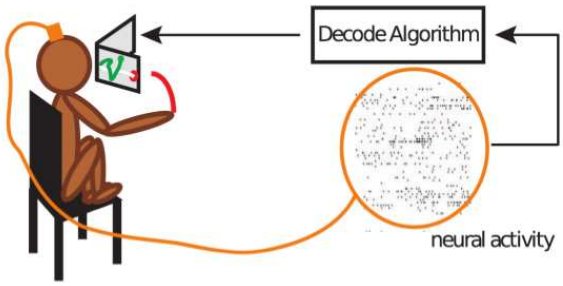

Figure 1: A monkey makes reaches while neural activity is recorded from motor cortex. In BCI mode (shown above) the monkey's neural activity is then passed to a decode algorithm to update the cursor position on the screen.

2.2 Decoders

In this section, we present background on the traditional Kalman filter used in the BCI literature and present a decoder novel to this work – the recurrent neural network (in collaboration with David Sussillo).

2.2.1 Kalman Filter

The Kalman filter is a recursive and efficient solution to estimate states of linear dynamical systems with noise and is the minimum mean square error estimator when the noise is Gaussian. The canonical discretized linear

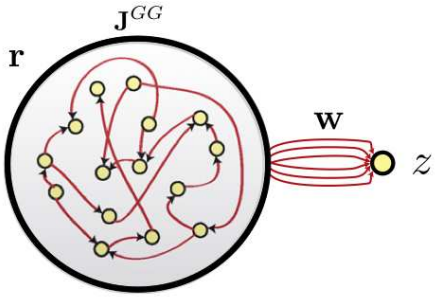

Figure 2: A recurrent neural network with randomized initial connections. Their outputs are fed to a vector of weights $\mathbf{w}$ which are linearly summed to produce the output z.

dynamical system is

$$x_{t+1} = Ax_t + v_t \quad (1)$$
$$y_t = Cx_t + w_t \quad (2)$$

where x_t – the state – are the cursor kinematics, specifically position and velocity, and y_t – the output – is the neural spiking data. The noise terms are Gaussian noise $v_t \sim \mathcal{N}(0, \Sigma_v)$ and $w_t \sim \mathcal{N}(0, \Sigma_w)$. A, C, and the noise covariances can all be fit via least squares using observations of the data.

To perform state estimation, we employ Ricatti recursion to estimate the covariance of the state prediction error, $\hat{\Sigma}_{t|t-1}$ and update the predicted state $\hat{x}_t$. The state prediction covariance recursion and state estimation are

$$\hat{\Sigma}_{t+1|t} = A\hat{\Sigma}_{t|t-1}A^T - L_tC\hat{\Sigma}_tA^T + \Sigma_v \quad (3)$$
$$\hat{x}_{t+1} = A\hat{x}_t + L_t(y_t - \hat{y}_t) \quad (4)$$

where $L_t = A\hat{\Sigma}_{t|t-1}C^T\left(C\hat{\Sigma}_{t|t-1}C^T + \Sigma_w\right)^{-1}$ is often termed the *observer gain* and $\hat{y}_t = C\hat{x}_t$ is the predicted output. Thus our estimated state $\hat{x}_{t+1}$ can be viewed as the superposition of a state update $A\hat{x}_t$ and a linear function of the output prediction error.

2.2.2 Recurrent Neural Network

The recurrent neural network shown in Figure 2 is initialized to have random synaptic connections and strengths. In Figure 2, r_i is a measure of neuron i's firing rate (x_i) after the hyperbolic tangent nonlinearity, i.e. $r_i = \tanh(x_i)$, J^{CG} is a matrix denoting the synaptic strengths, $\mathbf{w}$ are the output weights, and z is the output. The dynamics are governed by equation 5, where g is a scalar gain.

$$\tau\dot{x}_i = -x_i + g\sum_k J^{CG}_{ik}r_k \quad (5)$$

Due to the recurrent nature of the network, it is not immediately intuitive how backpropagation would be used to provide error signals to each neuron, which are used to optimize its weights via gradient descent. Assuming the network output is desired to be $f(t)$, we define the *output* error signal $e(t) = z(t) - f(t)$. The novelty of FORCE learning [2] is to use the *output* error signal as the *neuron* error signal for every neuron. In this framework, the network feeds back the overall output error and can use recursive least squares to train the weights $\mathbf{w}$. Learning the weights $\mathbf{w}$ effectively trains the entire network because it can be shown that the network synaptic strengths J in equation (5) can be re-written as

$$J_{eff} = gJ^{CG} + w_y\mathbf{w}^T \quad (6)$$

where w_y are neuron y's weights. Because the addition of a one rank matrix to a rank n matrix can change all n eigenvalues, the *overall* network is trained. The details of the supervised recursive least squares training are in [2] and are omitted from this paper for brevity.

2.3 Dimensionality Reduction

On a per-trial basis, it is useful to visualize how the neuron firing rates evolve over time. However, to capture the ensemble activity, this would require visualizing a 96 dimensional space, which is not tractable. Moreover, it has been reported that most of the variance of motor neural data for reaching tasks like ours can be captured in approximately 10 dimensions [4]. Thus, dimensionality reduction algorithms can be used to project neural data onto 3 dimensions and visualize trajectories trial-by-trial in a $3D$ space.

2.3.1 Principal components analysis

PCA analysis is performed as formulated in CS 229 lectures [5]. The data, X, have rows corresponding to the channel number and columns corresponding to smoothed firing rates over time in 20 ms bins. The firing rates are binned by counting the number of spikes occurring in non-overlapping 20 ms intervals and are square rooted to stabilize the firing rate variance. Smoothing is achieved by convolving the binned firing rates with a Gaussian kernel of standard deviation 30 ms. Thus, dimensionality reduction using PCA requires two stages: (1) binning and smoothing of firing rates and (2) dimensionality reduction.

2.3.2 Factor analysis

FA analysis requires both binning and smoothing of firing rates. As formulated in CS 229 lectures [5], $x_{i,t}$ is the firing rate of channel i at time t, z_{jt} is the latent state dimension j at time t, and the overall z represents the reduced dimensionality of the data. The data is formatted in matrix form, i.e. $x = \left[x_{:,1}^T \ldots x_{:,T}^T\right]$ where T

is the length of a trial, and $z = \left[z_{:,1}^T \dots z_{:,T}^T\right]$. The output, $x = \mu + \Lambda z$ is the key to visualization. In effect, z can be viewed as coefficients which determine the contribution of a column of Λ to x. However, the columns of Λ are not orthonormal. Thus, we perform a singular value decomposition (SVD) on $\Lambda = U\Sigma V^T$ and transform coordinates into $\tilde{z} = \Sigma V^T z$ so that $\Lambda z = U\tilde{z}$. U is an orthogonal matrix comprising the left singular vectors of Λ and $\tilde{z}$ is the rotated (by V^T) and scaled (by Σ) version of z. To visualize trajectories, we calculate $E[z|x] = \Lambda^T\left(\Lambda\Lambda^T + \Psi\right)^{-1}(x - \mu)$ as derived in CS 229 lectures using jointly Gaussian random variables [5]. Then, $E[\tilde{z}|x] = \Sigma V^T E[z|x]$ is the *orthonormalized* trajectory which can be projected onto the bases, U.

2.3.3 Gaussian process factor analysis

Gaussian process factor analysis (GPFA) is similar to factor analysis, but assumes that the latent states are related through time via a Gaussian process (GP). Thus, if $z_{i,j}$ is the latent state for dimension i at time j then, $z_{i,:} \sim \mathcal{N}(0, K_i)$ where covariance matrix K_i is element-wise

$$K_i(t_1, t_2) = \sigma_{f,i}^2 \exp\left(-\frac{(t_1 - t_2)^2}{2\tau_i^2}\right) + \sigma_{n,i}^2\delta_{t_1,t_2} \quad (7)$$

where δ_{ij} is the Kronecker delta, and $\sigma_{f,i}^2$ and $\sigma_{n,i}^2$ are constant pre-determined variances. The parameters μ, Λ, Ψ can be estimated using maximum likelihood and expectation maximization as derived in [4] and the τ_i can be optimized via gradient descent. The effect of relating the latent states through a Gaussian process is to effectively smooth the data. Thus, GPFA is not a two-stage algorithm, but rather an algorithm which simultaneously smooths the data and performs dimensionality reduction.

2.3.4 Algorithmic Choices

One shortcoming of PCA is the lack of a noise model. Neurons can be well-characterized as a Poisson process, indicating that a neuron's firing rate variance is approximately equal to its firing rate mean. Since PCA seeks dimensions which maximize variance, it favors channels with higher firing rates and incorrectly skews the bases. Upon implementing PCA, we observed separated trajectories across different reach directions; however, the trajectories were inherently more noisy on a trial-by-trial basis as compared to the latter two algorithms. These trajectories are not shown for brevity.

FA provides a major advantage over PCA in that it allows individual neurons to have their own variance and can therefore model neuron noise. Thus, unlike PCA, it will not incorrectly weight neurons that have high firing rates but are not necessarily informative. Rather, FA seeks directions that maximize *covariance* between neurons. We therefore observed that neural trajectories using FA had lower across trial variance as compared to that of PCA.

In both PCA and FA, smoothing and the dimensionality reduction are performed separately and cannot inform each other. A major advantage of GPFA is that smoothing and dimensionality reduction are done simultaneously and can thereby be jointly optimized, allowing for optimal smoothing. Moreover, GPFA has been shown to be superior to FA and PCA in predicting a leave-one-out-CV neuron firing rate [4]. In the neural trajectories constructed using GPFA, we noticed that the path curvature was smoother and had less across-trial variance as compared to the two-step algorithms. We therefore utilize GPFA in the following analyses.

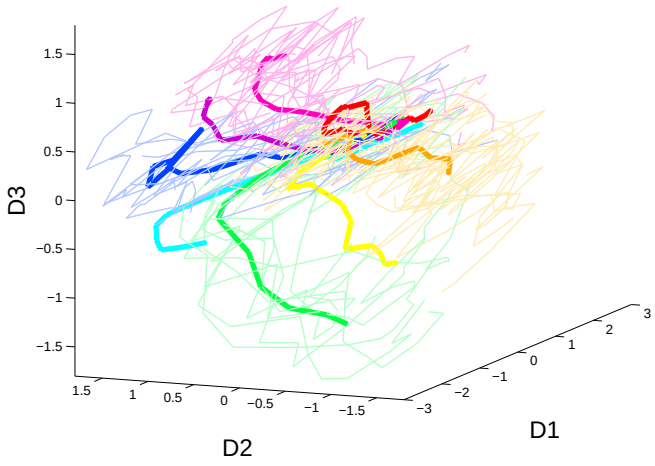

Figure 3: The neural trajectories (GPFA) for 96 dimensional data (96 channels of neural data) reduced to 3 dimensions under brain control with a modified Kalman filter decoder. The trajectories are colored based on target location and the bold trajectories are mean trial trajectories. It is clear that reaches in different directions follow separate trajectories in space.

3 Results

3.1 Behavioral-dependent visual space trajectories

Figure 3 shows the neural trajectories on a trial-by-trial basis across 8 different reach directions, denoted by separate colors. There is clear separation of the trajectories, and they occupy different regions of the 3 dimensional space. Furthermore, the order of the trajectories mirror the relative position of the targets

acquired for actual hand reaches – red maps to a downward left reach, orange to a downward reach, yellow to a downward right reach, and so on.

3.2 Channel count

Signals received on each of the 96 channels contain varying amounts of information. For example, a channel may be far away from any neurons and therefore receive small signal, or it may be measuring action potentials of a neuron that is not informative of reach direction. Therefore, it is useful to have a way to rank channels in order of importance, and observe how neural trajectories differ by using only a subset of channels. Below, we investigate a ranking method to select a subset of channels that are most informative.

3.2.1 Variance normalized maximum depth

The ranking metric we develop, variance normalized maximum depth (MDV), was inspired by previous work in neuroscience. The MDV of a single channel is defined as

$$MDV = \frac{\max_{y \in Y} \bar{x}(y) - \min_{y \in Y} \bar{x}(y)}{\sigma_x^2} \quad (8)$$

where $\bar{x}(y)$ is the average firing rate of the channel when a reach in direction y is made and σ_x^2 is the firing rate variance for that channel. The metric is a measure of normalized "tuning depth," or roughly, the maximum difference in a channel's firing rate across reach directions.

It is worth noting that it is possible to calculate the mutual information between channel i and a reach direction y by empirically determining the firing rate probability mass function (pmf) of that channel, $p_{X_i}(x_i)$. We also determine the same firing rate pmf conditioned on a reach direction $Y = y$, giving $p_{X_i|Y}(x_i|y)$. With these pmfs, we can calculate the mutual information between neuron i and reach direction via $I(X_i;Y) = H(X_i) - H(X_i|Y)$. A ranking of channels using mutual information yields similar results to MDV.

3.2.2 Visualization with different channel counts

When the number of channels used for dimensionality reduction is decreased from 96 channels (Figure 3) to 24 channels (Figure 4) there is an obvious compression of trajectory volume. We believe that by including fewer noisy channels, the lower dimensional space is expanded in a non-informative way. This is because even after a lower dimensional basis is found, noisy channels are projected into this lower dimensional space and thus corrupt the trial-by-trial trajectories.

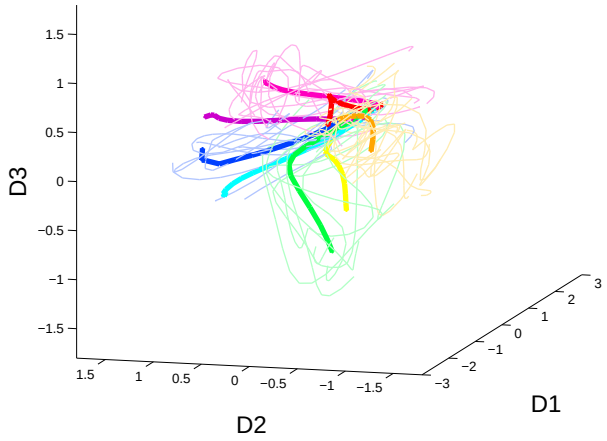

Figure 4: The neural trajectories (GPFA) for 24 dimensional data (24 channels of neural data) reduced to 3 dimensions under brain control with a modified Kalman filter. These traces occupy a smaller volume than the trajectories of Figure 3.

3.3 Decoder variation

In Figure 3, 4, and 5, the neural trajectories are built off a set of bases representing all reach directions for the same mode of control even though only two reach directions are visualized. In each mode of control, there is clear separation of the upward (blue) and downward-reaching (orange) trajectories. By analyzing differences in the curvature and the across-trial-variance of the trajectories, we can make hypotheses about neural control strategy when operating under different modes of BCI. For instance, the increase in across-trial-variance between different models may reflect robustness or sensitivity of a decoding algorithm. It may also indicate whether a reach is mostly ballistic or subject to online corrections. Alternatively, it could simply indicate increased variation in the dynamics of control. These hypotheses will necessitate further scientific inquiry to better understand. Because these three dimensions represent only a fraction of the variability in the dataset, projections to a higher dimensional space would be required to fully quantify metrics.

4 Conclusions

In this project, we explored neural relationships between various control modalities and decoders utilizing several dimensionality reduction techniques. We carried out behavioral experiments in monkeys to gather data to test these hypotheses, using both hand kinematics and real-time brain control tasks. Two BCI decoders were examined: a feedback-modified Kalman fil-

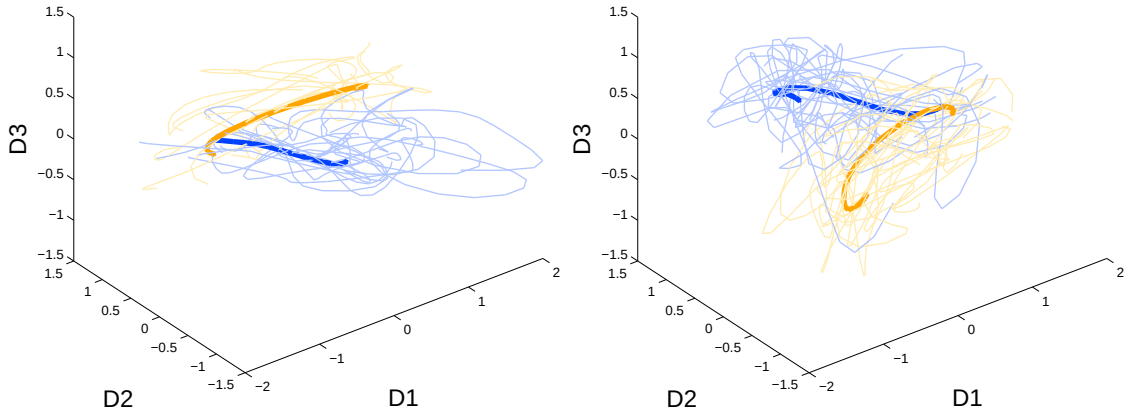

Figure 5: (left) The GPFA trajectories for downward (orange) and upward (blue) hand reaches, using only 24 channels. (right) The GPFA trajectories for downward (orange) and upward (blue) neural network decodes, using only 24 channels.

ter and a novel FORCE neural network decoder. We demonstrated the feasibility of the neural network decoder for online real time BCI. The differences between these decoders in contrast to native limb control were explored with PCA, FA, and GPFA. We ultimately settled on GPFA as the final tool for analysis because of its ability to maximize separation in the neural state representation. Additionally, the impact of non-informative electrode exclusion on the smoothness of the low-dimensional representation was examined. Further detailed analysis is necessary to characterize specific changes to decoders to improve performance. However, one clear finding from this work that would improve decoder robustness would be to preemptively screen and remove electrodes that are not as informative and may be mis-modeled by the decoder. Taken together, these findings can help inform subsequent decoder design and potentially improve clinical BCI utility in translation.

5 Acknowledgments

We thank D. Sussillo for collaborating on the neural network decoder. We are grateful to M. Risch and J. Aguayo for veterinary care, and S. Eisensee for administrative support. We thank K. Shenoy for his guidance, support, and resources.

6 References

1. Gilja V*, Nuyujukian P*, Chestek CA, Cunningham JP, Yu BM, Ryu SI, Shenoy KV (2010, poster) *High-performance continuous neural cursor control enabled by a feedback control perspective.* Frontiers in Neuroscience. Conference Abstract: Computational and Systems Neuroscience (COSYNE), Salt Lake City, UT.

2. Sussillo, D, Abbott, LF (2009) *Generating Coherent Patterns of Activity from Chaotic Neural Networks.* Neuron. 63:544-557.

3. Churchland MM*, Yu BM*, Cunningham JP, Sugrue LP, Cohen MR, Corrado GS, Newsome WT, Clark AM, Hosseini P, Scott BB, Bradley DC, Smith MA, Kohn A, Movshon JA, Armstrong KM, Moore T, Chang SW, Snyder LH, Lisberger SG, Priebe NJ, Finn IM, Ferster D, Ryu SI, Santhanam G, Sahani M, Shenoy KV (2010) *Stimulus onset quenches neural variability: a widespread cortical phenomenon.* Nature Neuroscience. 13:369-378.

4. Yu BM, Cunningham JP, Santhanam G, Ryu SI, Shenoy KV*, Sahani M* (2009) *Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity.* Journal of Neurophysiology. 102:614-635.

5. Ng, A. *CS229 Lecture Notes.* Stanford University, Autumn 2010.