ARTICLE Communicated by Vasilis Marmarelis

Design and Analysis of Closed-Loop Decoder Adaptation Algorithms for Brain-Machine Interfaces

Siddharth Dangi*
sdangi@eecs.berkeley.edu
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA 94720, U.S.A.

Amy L. Orsborn*
amyorsborn@berkeley.edu
UC Berkeley–UCSF Graduate Group in Bioengineering, University of California, Berkeley, Berkeley, CA 94720, U.S.A.

Helene G. Moorman
helenem@berkeley.edu
Helen Wills Neuroscience Institute, University of California, Berkeley, Berkeley, CA 94720, U.S.A.

Jose M. Carmena
carmena@eecs.berkeley.edu
Helen Wills Neuroscience Institute, and Department of Electrical Engineering and Computer Sciences, UC Berkeley–UCSF Graduate Group in Bioengineering, University of California, Berkeley, Berkeley, CA 94720, U.S.A.

Closed-loop decoder adaptation (CLDA) is an emerging paradigm for achieving rapid performance improvements in online brain-machine interface (BMI) operation. Designing an effective CLDA algorithm requires making multiple important decisions, including choosing the timescale of adaptation, selecting which decoder parameters to adapt, crafting the corresponding update rules, and designing CLDA parameters. These design choices, combined with the specific settings of CLDA parameters, will directly affect the algorithm's ability to make decoder parameters converge to values that optimize performance. In this article, we present a general framework for the design and analysis of CLDA algorithms and support our results with experimental data of two monkeys performing a BMI task. First, we analyze and compare existing CLDA algorithms to highlight the importance of four critical design elements: the adaptation timescale, selective parameter adaptation, smooth decoder updates, and

*SD and ALO contributed equally to this work.

intuitive CLDA parameters. Second, we introduce mathematical convergence analysis using measures such as mean-squared error and KL divergence as a useful paradigm for evaluating the convergence properties of a prototype CLDA algorithm before experimental testing. By applying these measures to an existing CLDA algorithm, we demonstrate that our convergence analysis is an effective analytical tool that can ultimately inform and improve the design of CLDA algorithms.

1 Introduction

Brain-Machine Interfaces (BMIs) are an emerging technology designed to restore motor function to those with spinal cord injury, stroke, amyotrophic lateral sclerosis, and other disabling conditions. Researchers have exhibited compelling proof-of-concept BMI experiments, with multiple groups showing impressive demonstrations—in rodents (Chapin, Moxon, Markowitz, & Nicolelis, 1999; Gage, Ludwig, Otto, Ionides, & Kipke, 2005; Koralek, Jin, Ii, Costa, & Carmena, 2012), nonhuman primates (Serruya, Hatsopoulos, Paninski, Fellows, & Donoghue, 2002; Taylor, Tillery, & Schwartz, 2002; Carmena et al., 2003; Musallam, Corneil, Greger, Scherberger, & Andersen, 2004; Santhanam, Ryu, Yu, Afshar, & Shenoy, 2006; Jarosiewicz et al., 2008; Moritz, Perlmutter, & Fetz, 2008; Velliste, Perel, Spalding, Whitford, & Schwartz, 2008; Ganguly & Carmena, 2009; O'Doherty, Lebedev, Hanson, Fitzsimmons, & Nicolelis, 2009; Suminski, Tkach, Fagg, & Hatsopoulos, 2010; Ethier, Oby, Bauman, & Miller, 2012), and humans (Hochberg et al., 2006; Kim, Simeral, Hochberg, Donoghue, & Black, 2008; Hochberg et al., 2012)—using neural activity to control both real and artificial actuators. However, significant improvements in both reliability (lifetime usability of the interface) and performance (achieving control and dexterity comparable to natural movements) are still needed to achieve clinically viable neuroprostheses for humans (Millán & Carmena, 2010; Gilja et al., 2011).

A fundamental component of any BMI is the decoding algorithm, or "decoder," that translates recorded neural activity into control signals for a prosthetic device. Decoders are often initialized offline by recording neural activity while a subject performs real movements (Serruya et al., 2002; Taylor et al., 2002; Carmena et al., 2003; Musallam et al., 2004; Santhanam et al., 2006) or observes or imagines movements (Hochberg et al., 2006; Wahnoun, He, & Helms-Tillery, 2006; Kim et al., 2008; Velliste et al., 2008; Suminski et al., 2010), and fitting a decoder to predict these movements from the neural activity. However, BMIs are fundamentally closed-loop systems, since BMI users receive performance feedback (e.g., by visual observation of the prosthetic's movements). Recent research shows that the prediction power of decoders trained offline does not directly correlate with closed-loop performance, perhaps due to the inherent feedback-related differences between open- and closed-loop systems (Ganguly & Carmena, 2010; Koyama et al.,

2010). These results suggest that performance improvements cannot be achieved solely by finding an optimal open-loop decoding algorithm and therefore highlight the importance of treating BMIs as closed-loop systems.

In the closed-loop regime, two mechanisms can be used to achieve high-performance BMI systems. The first mechanism, neural plasticity or brain adaptation, is the ability of neurons to adapt their receptive fields and tuning properties to facilitate performance improvements. For instance, previous work by Ganguly and Carmena (2009) demonstrated that when monkeys practiced BMI control with a stable circuit consisting of a fixed decoder and stable neurons, they developed a stable neural map of the decoder that enabled performance improvements, was readily recalled across days, and was robust to interference from a second learned map. The second mechanism, closed-loop decoder adaptation (CLDA), is the process of adapting or updating the decoder's parameters during closed-loop BMI operation, that is, while the subject is using the BMI (see Figure 1). The goal of CLDA is to make the decoder's output accurately reflect the user's intended movements (Orsborn, Dangi, Moorman, & Carmena, 2012).

The concept of CLDA has shown great promise as a mechanism to either improve or maintain closed-loop BMI performance. A wide range of CLDA algorithms has been developed and tested for various purposes (see Table 1 for a sampling of prior work). Some of these algorithms strive to improve control accuracy when limited information is available to create an initial decoder (Orsborn et al., 2012), while others aim to maintain the control accuracy of already high-performing decoders (Li, O'Doherty, Lebedev, & Nicolelis, 2011). Although these algorithms all represent implementations of CLDA, they are designed to operate in characteristically different ways. Indeed, those developing a CLDA algorithm have many different design choices to make, including choosing the timescale of adaptation, selecting which decoder parameters to adapt, crafting the corresponding update rules, and designing CLDA parameters. These decisions may have a significant impact on an algorithm's performance. For instance, a CLDA algorithm's ability to improve closed-loop BMI performance may be particularly sensitive to the timescale at which it adapts decoder parameters (Orsborn et al., 2012; Orsborn, Dangi, Moorman, & Carmena, 2011). To date, little work has been done to explore how algorithms' design choices influence their performance. Similarly, few techniques have been established to assess algorithms' convergence properties. Ideally, CLDA algorithms should be designed to make the decoder's parameters converge rapidly to or maintain values that are optimal for high-performance BMI control. While conducting closed-loop BMI experiments is ultimately the only conclusive way to evaluate a CLDA algorithm's convergence properties, such experiments are lengthy and costly. Although closed-loop simulation methods (Cheng & Sabes, 2006; Héliot, Ganguly, Jimenez, & Carmena, 2010; Cunningham et al., 2011) could potentially be used to study convergence, these tools have not yet been used for this purpose. Aside from these tools, there are currently limited methods for exploring the parameter space and predicting

Table 1: Sampling of Prior Work on CLDA Algorithms.

CLDA Algorithm	Decoder	Form of CLDA Updates
Taylor et al. (2002)	Population vector algorithm	Iterative refinement of tuning properties based on current performance, errors from most recent block, and best weights from a past block
Gage et al. (2005)	Kalman filter	Batch maximum likelihood estimation using data from a trial-by-trial sliding block
Shpigelman, Lalazar, and Vaadia (2008)	Kernelized ARMA	Replacement of old examples in training set with new examples
Gilja et al. (2012)	Kalman filter	Batch maximum likelihood estimation
Héliot, Venkatraman, and Carmena (2010)	Wiener filter	Remapping of a lost neuron's decoder weights using next closest neuron's weights
Mahmoudi and Sanchez (2011)	Time-delayed neural network	Decoder adapted within a reinforcement learning (actor-critic) framework
Li et al. (2011)	Kalman filter	Bayesian regression self-training updates
Dangi, Gowda, Héliot, and Carmena (2011), Adaptive KF	Kalman filter	Stochastic gradient descent
Orsborn et al. (2012), SmoothBatch	Kalman filter	Weighted average of current parameters with new batch maximum likelihood estimates
Hochberg et al. (2012)	Kalman filter	Iteratively updated filter parameters during successive closed-loop calibration blocks

the convergence properties of a prototype CLDA algorithm before experimental testing. A better understanding of how different design elements of a CLDA algorithm influence its closed-loop performance and convergence will be critical to developing high-performance BMIs.

In this article, we present a general framework for the design and analysis of CLDA algorithms. First, we identify and explore a core set of important design considerations that frequently arise when designing CLDA algorithms. For instance, we evaluate the effects of an algorithm's timescale of adaptation, which can have a significant impact on both the subject's level of engagement and the rate of performance improvements. Next, we underscore the importance of selective parameter adaptation by demonstrating why adapting certain decoder parameters can lead to undesired effects on performance. We then illustrate how smooth parameter updates can help mitigate the impact of unreliable batches of data, especially when initial performance is limited. Finally, we stress the importance of designing CLDA parameters that are readily interpretable, a property that can help avoid conducting large parameter sweeps and simplify parameter selection in experimental work.

We then introduce mathematical convergence analysis, using measures such as mean-square error (MSE) or KL divergence (KLD), as a useful precursor to closed-loop experiments that can further inform CLDA design and constrain the necessary experimental testing. We choose the SmoothBatch CLDA algorithm (Orsborn et al., 2012) as a case study to demonstrate the utility of our analysis. Our analysis predicts that SmoothBatch's MSEs converge exponentially to steady-state values and that both these values and the rate of convergence are independent of the decoder's seeding method. Experimental data from two nonhuman primate subjects across 72 sessions serve to support our predictions and validate our convergence measures. Finally, guided by our convergence predictions, we propose a specific method to improve the SmoothBatch algorithm by allowing time-varying CLDA parameters. Overall, our study of CLDA design considerations sheds light on important aspects of the design process, while our mathematical convergence analysis serves as a useful tool that can inform the design of future CLDA algorithms prior to conducting closed-loop experiments.

2 Methods

2.1 Experimental Procedures. CLDA algorithms were experimentally tested using intracortically recorded neural activity from two nonhuman primates. Electrophysiology and behavioral protocols have been previously described in Orsborn et al. (2011, 2012). Briefly, 128-electrode microwire arrays (35 μm diameter, 500 μm wire spacing, 8×16 array configuration; Innovative Neurophysiology, Durham, NC) were implanted in both brain hemispheres of two adult male rhesus macaque monkeys. Arrays were positioned to target the arm areas of primary motor cortex (M1) and, due

to their size, extended rostrally into dorsal premotor cortex (PMd). Unit activity was recorded using a combination of two 128-channel MAP and OmniPlex recording systems (Plexon, Dallas, TX). Single and multiunit activity was sorted using an online sorting application (Plexon, Dallas, TX), and only units with well-identified waveforms were used for BMI control. Monkeys were head restrained in a primate chair and observed a task display via a computer monitor projecting to a semitransparent mirror parallel to the floor. They were trained to perform a self-paced 2D center-out reaching task to eight circular targets uniformly spaced around a circle. During BMI operation, the subjects' arms were confined within the primate chair as they performed the task by moving the cursor under neural control.

Figure 1 shows an illustration of the task setup and trial time line. Reach targets were presented in a block structure of eight targets with pseudo-randomized order within each block. Targets were spaced 7 cm (subject 1) or 6.5 cm (subject 2) away from the center. Target sizes of 1.2 to 1.7 cm radius, center and target holds of 250 to 400 ms, and reach times of 3 to 5 s were used. All procedures were conducted in compliance with the National Institutes of Health Guide for Care and Use of Laboratory Animals and were approved by the University of California, Berkeley Institutional Animal Care and Use Committee.

2.2 Decoding Algorithm. All experimental sessions used a Kalman filter (KF) as the decoding algorithm during closed-loop control. In this article, we focus on BMI cursor control, although our analysis and results still apply more generally to any BMI prosthetic device. Let x_t and y_t be vectors representing the kinematic state of a computer cursor and binned neural firing rates, respectively, at time t. The KF assumes the following models for how the state x_t evolves over time and how the observed firing rates y_t relate to the state:

$$x_t = Ax_{t-1} + w_t, \tag{2.1}$$

$$y_t = Cx_t + q_t, \tag{2.2}$$

where $w_t \sim \mathcal{N}(0, W)$ and $q_t \sim \mathcal{N}(0, Q)$ are gaussian noise terms. From the above equations, it is evident that the KF model can be parameterized by the matrices $\{A, W, C, Q\}$. In BMI control, the decoder's purpose at each time step t is to estimate the user's intended cursor state x_t based on the neural activity $\{y_t, y_{t-1}, y_{t-2}, \ldots\}$ observed so far. The KF is an efficient, linear algorithm that performs this estimation recursively. In other words, it estimates x_t using only y_t and its own previous estimate of x_{t-1}. We refer readers to Haykin (2001) and Wu et al. (2003) for the actual KF equations that perform the state estimation at each filter iteration.

We used a position-velocity KF operating in end-point coordinates where the KF state vector x_t is defined to include the position (p) and velocity (v)

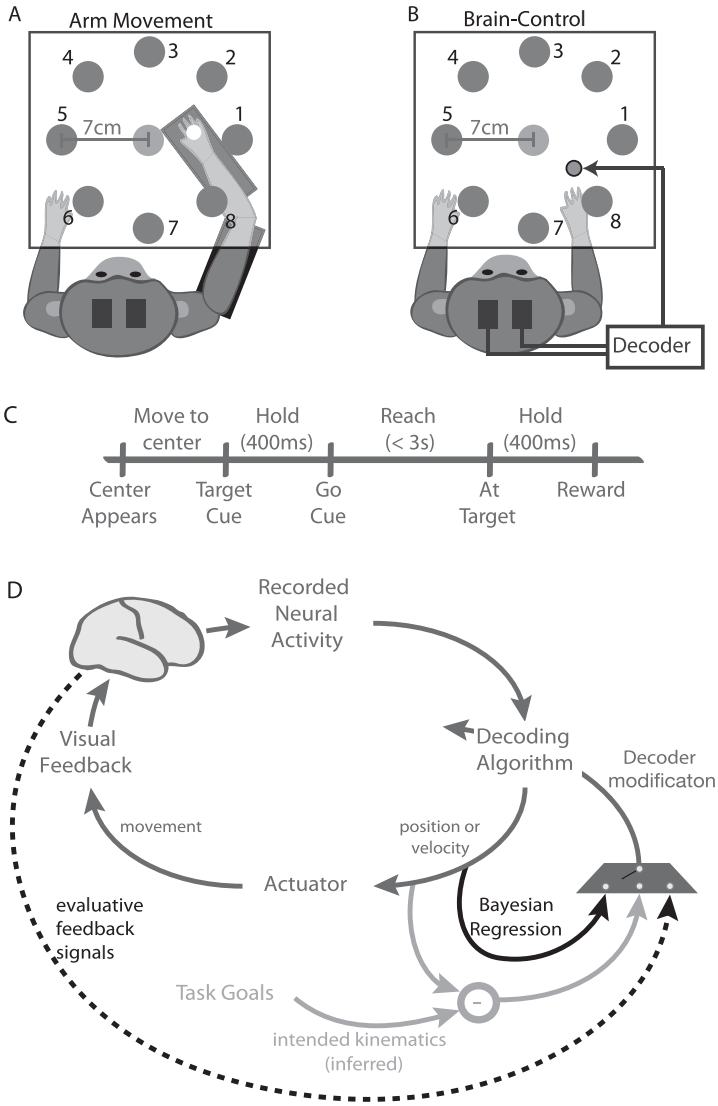

Figure 1: Experimental setup and closed-loop decoder adaptation (CLDA). Illustration of the (A) experimental task setup for arm movements and (B) closed-loop BMI experiments, and (C) trial time line for the center-out task. Panel D illustrates the concept of closed-loop decoder adaptation. The decoder is modified as a subject uses it in closed-loop control. The signals used to modify the decoder can be attained in several ways, including using (1) task goals to infer a subject's intentions (light gray), (2) Bayesian methods to self-train the decoder (solid black), or (3) neural signals (dashed black).

of the cursor at time t along both the horizontal and vertical directions of the screen:

$$x_t = \left[p_{\text{horizontal},t} \;\; p_{\text{vertical},t} \;\; v_{\text{horizontal},t} \;\; v_{\text{vertical},t} \;\; 1 \right]^T.$$

The constant term 1 and a corresponding extra column of C are added to account for observations y_t with nonzero means (e.g., the neurons' baseline firing-rates).

Online BMI control was implemented using PlexNet (Plexon, Dallas, TX) to stream neural data on a local intranet from the Plexon recording system to a dedicated computer running Matlab. Neural firing rates were estimated with a 100 ms bin width. Neural ensembles of 16 to 36 (26.5 ± 4.45; mean $\pm$ STD) neurons were used. Units were selected based on waveform quality.

Tests of CLDA algorithms typically used decoders initialized using five methods (Orsborn et al., 2012):

1. Recording neural activity as the subject passively watched a cursor moving through artificially generated center-out trajectories (visual feedback; *VFB*; $n = 32$)
2. Recording neural activity during contralateral arm movements (*contra*; $n = 2$)
3. Recording neural activity during ipsilateral arm movements (*ipsi*; $n = 8$)
4. Recording neural activity during quiet sitting (*baseline*; $n = 17$)
5. Arbitrary seeding by shuffling the coefficients of KF decoder matrices from previous sessions (*shuffled*; $n = 13$)

The structures of the KF state transition model parameters (A and W) were constrained during fitting to obey physical kinematics, such that integrating the velocity from one KF iteration perfectly explains the position at the next iteration (see Gilja et al., 2012, for further details). As discussed below, these parameters were typically held fixed during CLDA. In order to allow for BMI control that would mimic natural arm movements as much as possible, A and W were fit for all seeding methods using a 30 minute data set of arm movements collected while the subjects performed the center-out task in manual control. In experiments where A and W were adapted (see section 3; *VFB* and *contra* seeds only), these matrices were initialized to their maximum likelihood estimates calculated from the recorded kinematic data collected specifically for seeding. For seeding methods 1 to 4, the KF observation model parameters, C and Q, were seeded using batch maximum likelihood estimates based on 8 minutes of neural and kinematic data.

2.3 KF CLDA Algorithms. In the following sections, we review some existing CLDA algorithms for KF decoders that will be compared within our analyses. These algorithms update the decoder as the subject performs the center-out task in closed-loop BMI control. In closed-loop experiments with

each algorithm, the subject's intended cursor kinematics were inferred from the observed kinematics using the method developed by Gilja et al. (innovation 1 of the ReFIT-KF algorithm), which assumes that the subject always intends to move in a straight line toward the target and rotates observed cursor velocities accordingly (Gilja et al., 2010, 2012). For reasons discussed in section 3, we typically update only the KF decoder's observation model parameters. As such, update rules for only the C and Q matrices are given below for each CLDA algorithm. (The Batch algorithm's update equations for A and W, when these parameters are also adapted, are equivalent to maximum-likelihood estimation of these parameters, as presented in Orsborn et al., 2012.) Each time new values for decoder parameters are calculated, they are immediately used in the decoder as part of subsequent BMI control.

2.3.1 Batch Maximum Likelihood Estimation. One paradigm for KF CLDA algorithms entails collecting data and processing the entire batch at once to update the decoder's parameters. In this approach, the updated C and Q matrices are set to their maximum likelihood estimates based on the batch of data. We refer to this method as the Batch algorithm. The Batch algorithm's update rules are

$$\hat{C} = YX^T(XX^T)^{-1}, \tag{2.3}$$

$$\hat{Q} = \frac{1}{N}(Y - \hat{C}X)(Y - \hat{C}X)^T, \tag{2.4}$$

where the Y and X matrices are formed by tiling N columns of recorded neural activity and intended cursor kinematics, respectively. The size of the data batch is parameterized by the batch period $T_b \triangleq N \cdot \mathrm{dt}$ (where dt is the time between KF decoder iterations).

2.3.2 Adaptive Kalman Filter. The adaptive Kalman filter (Adaptive KF or AKF; Dangi et al., 2011) is a CLDA algorithm designed to update the KF's observation model parameters at every decoder iteration, which corresponds to every 100 ms in our experiments. If we let i index discrete decoder iterations, the Adaptive KF's update rules are

$$C^{(i+1)} = C^{(i)} - \mu(C^{(i)}x_t - y_t)x_t^T,$$

$$Q^{(i+1)} = \alpha Q^{(i)} + (1-\alpha)(y_t - C^{(i+1)}x_t)(y_t - C^{(i+1)}x_t)^T,$$

where y_t and x_t represent neural firing rates and an estimate of the user's intended kinematics, respectively, at time t.[1] The update rule for C is derived

[1] Note that here, t actually equals i. However, we choose to maintain both as separate indices in order to be consistent with other algorithms such as SmoothBatch, where they are different because the decoder is not updated at every iteration.

by writing this matrix as the solution to an optimization problem and then using stochastic gradient descent with step-size μ to iteratively make small corrections to it at every decoder iteration. The adaptive KF's update rule for Q is of heuristic form and effectively represents a weighted average (with parameter $\alpha \in [0, 1]$) of the current value of Q with a single-iteration estimate of Q.

2.3.3 SmoothBatch. The SmoothBatch CLDA algorithm (Orsborn et al., 2012) periodically updates the KF decoder's observation model (C and Q matrices) by performing a weighted average of the current parameters with those estimated from a new batch of data using the Batch algorithm. The observed neural activity and intended cursor kinematics are collected over one batch period. This batch of data is then used to construct new Batch estimates, $\hat{C}$ and $\hat{Q}$, of the C and Q matrices using equations 2.3 and 2.4. Finally, the observation model parameters are updated using a weighted average:

$$C^{(i)} = (1 - \rho)\hat{C}^{(i)} + \rho C^{(i-1)}, \tag{2.5}$$

$$Q^{(i)} = (1 - \rho)\hat{Q}^{(i)} + \rho Q^{(i-1)}, \tag{2.6}$$

where i indexes discrete batch periods and the weighting parameter $\rho \in [0, 1]$ controls the influence of $\hat{C}$ and $\hat{Q}$ on the new parameter settings.

3 CLDA Design Principles

The development of a CLDA algorithm involves making multiple crucial design decisions such as:

- How often to apply update rules (the timescale of adaptation)
- Whether to update all decoder parameters
- How to update decoder parameters (the actual parameter update formulas or "update rules")
- How to set CLDA algorithmic parameters (e.g., step sizes or learning rates)

Ideally, these design choices should enable a CLDA algorithm to make the decoder's parameters converge rapidly to or maintain values that are optimal for high-performance BMI control. In the following sections, we explore these design choices by examining various aspects of the Batch, Adaptive KF, and SmoothBatch CLDA algorithms. We focus primarily on SmoothBatch, which has been demonstrated to improve BMI control accuracy even when initial performance is severely limited due to an unfavorable seeding (Orsborn et al., 2012). For instance, with patients such as paralyzed individuals or amputees who would be likely clinical targets of

BMI technology, seeding methods involving overt movements would not be feasible, and therefore other methods that typically result in a lower level of initial performance would need to be used. Despite our focus on SmoothBatch, however, we will see that many of its design properties are likely to be shared by other CLDA algorithms, even when their underlying purpose is characteristically different.

3.1 Timescale of Adaptation. A CLDA algorithm's timescale of adaptation refers to the frequency with which its update rules are applied to adapt the decoder's parameters. In a clinical setting with paralyzed patients, initial closed-loop BMI performance may be limited. As such, it is critical to develop CLDA algorithms capable of rapidly improving control accuracy regardless of initial performance. Given the subject-decoder interactions inherent in closed-loop BMI systems, the timescale of adaptation may be particularly important in these situations. Indeed, previous work comparing two CLDA algorithms (the Batch and AKF algorithms) operating on different timescales showed that the adaptation frequency can have a significant impact on CLDA performance (Orsborn et al., 2011). We recently showed that the SmoothBatch algorithm, when operating on an intermediate timescale (1–2 minutes), successfully and robustly improves performance independent of the initial decoder (Orsborn et al., 2012). Here, we extend these results by comparing experimental data from the Batch, AKF, and SmoothBatch algorithms to highlight the role of the adaptation timescale.

All three CLDA algorithms were experimentally tested with one nonhuman primate subject,[2] starting from limited task performance (Batch and AKF using *VFB* seeds, $n = 7$ and $n = 2$, respectively; SmoothBatch using *VFB*, *Ipsi*, *Baseline*, and *Shuffled* seeds, $n = 64$). For all algorithms, the state transition model (A and W) was held fixed. Sessions selected for comparison had comparable task settings and neural ensemble sizes. Figures 2A to 2C show examples of the Batch, AKF, and SmoothBatch algorithms' performance in representative sessions. The evolution of task performance is illustrated with both moving averages (75 trial window) of percentages of trial outcomes (i.e., success, reach error or hold error; upper panels) and rates of each task event (estimated via binning events in 120 s nonoverlapping bins; lower panels). Each CLDA algorithm was able to improve performance, as evidenced by increases in success percentages and rates and corresponding decreases in errors during adaptation. However, the algorithms differ significantly in how well the subject is able to maintain performance after adaptation ceases (i.e., when using a fixed decoder). For both the Batch

[2]The Batch and AKF algorithms were not tested in the second subject. Two SmoothBatch sessions in subject 1 did not have sufficient data with the fixed decoder to assess the post-CLDA maintenance of performance and thus were excluded from our analysis.

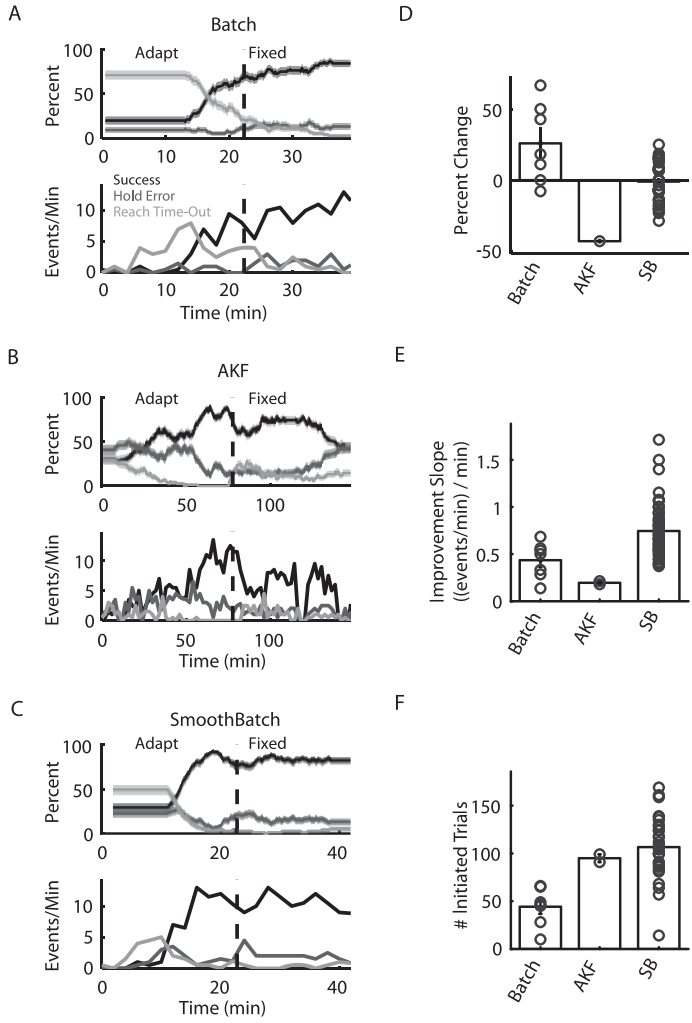

Figure 2: Comparison of the closed-loop BMI performance of three CLDA algorithms (Batch, Adaptive KF, and SmoothBatch) that operate on different timescales. (A–C) Experimental performance of these algorithms during representative BMI sessions. Performance was calculated using both moving averages (75 trial window) of percentages of trial outcomes (i.e., success, reach error, or hold error) and rates of each task event (estimated via binning events in 120 s nonoverlapping bins). (D) Percentage change in the success rate across all sessions, sorted by CLDA algorithm. (E) Improvement slope of the success rate for all sessions, sorted by CLDA algorithm. (F) Number of trials the subject attempted to initiate (via entering the center target) during the first 10 minutes of CLDA. Only sessions with *VFB* seeds were used to avoid potential differences due to the initial decoder.

and SmoothBatch algorithms, the subject readily maintained performance after the decoder was fixed. However, for the AKF, task performance (both success percentage and rate) dropped significantly after CLDA stopped.

Performance changes were assessed in two ways. First, we compared the maximum success rate during adaptation with the maximum rate during the first 15 minutes of using the fixed decoder. Second, we compared the success percentage during the last 50 trials during adaptation to that of the first 50 trials with the fixed decoder. Figure 2D depicts the percent change in the success rate for all sessions, sorted by CLDA algorithm. Batch ($n = 7$) and SmoothBatch ($n = 64$) showed no significant changes in performance (Wilcoxon paired test, $p > 0.05$). Note that a decoder update occurs between the adapt and fixed portion for the Batch algorithm, which may explain the slight (though not statistically significant) increase in performance observed. Only one AKF session had sufficient data with a fixed decoder to allow comparison, but this session shows a very large drop in performance. Success percentages showed identical trends (Batch and SmoothBatch had no significant change; Wilcoxon paired test, $p > 0.05$; not shown).

The adaptation timescale also has a significant influence on the rate of performance improvements. Improvement slopes were quantified by computing the change in success rate (or percentage) from the start of adaptation to the maximum achieved during adaptation, and dividing by the time required to achieve that maximum performance. Figure 2E shows the success rate improvement slope for all sessions sorted by CLDA algorithm. SmoothBatch shows significantly faster improvements than both Batch and AKF algorithms (Kruskal-Wallis analysis of variance, $p < 0.05$). The AKF improves the slowest, though not significantly slower than the Batch algorithm. Identical trends were found when comparing percentage-based slope estimates.

One key aspect of the adaptation timescale is the frequency with which the subject sees improvements in performance. This may be particularly important when starting from limited performance when the subject has little to no ability to control the cursor. For instance, the Batch algorithm requires the subject to persist in trying to use a poorly performing decoder for long periods of time, which may cause frustration, lack of motivation, or highly variable user strategies. By contrast, the AKF and SmoothBatch give the subject feedback more rapidly, keeping the subject more engaged in the task and potentially avoiding these user-related complications that could hinder the algorithm's progress. Subject engagement was assessed by calculating the number of trials the subject attempted to initiate (via entering the center target) during the first 10 minutes of CLDA. Here, only sessions with *VFB* seeds were used to avoid potential differences due to the initial decoder (Orsborn et al., 2012). Figure 1F shows that the subjects initiated significantly more trials in SmoothBatch sessions than Batch sessions (Kruskal-Wallis analysis of variance, $p < 0.05$). AKF sessions similarly show more trial initiations, though not significantly different from Batch or

SmoothBatch. This suggests that increasing the frequency of user feedback can indeed increase user engagement. Increased engagement likely contributes to the CLDA algorithm's ability to improve performance. Together these analyses show the importance of the CLDA algorithm timescale when starting from limited initial performance.

3.2 Selective Parameter Adaptation. BMI decoding algorithms are often characterized by several different parameters. For instance, a KF decoder is parameterized by four matrices, $\{A, W, C, Q\}$, each with different physical interpretations within a cursor control BMI system. A and W characterize the state transition model and represent cursor kinematics, while C and Q form the observation model that represents the mapping between neural activity and cursor movement. The KF combines these two models in a recursive algorithm to predict cursor movement, and its confidence in the models (related to the covariance matrices W and Q) ultimately determines their relative contributions to the final state estimate via a Kalman gain matrix, K.

A priori, a KF CLDA algorithm should adapt all four matrices since they all contribute to the final observed cursor movement. Indeed, previous work with Batch-based KF CLDA updated all KF parameters (Gilja et al., 2010, 2012). However, experimental evidence reveals that repeated adaptation (as is often needed when starting from limited initial performance) of the full KF model can have negative consequences. In particular, decoded cursor speed shows marked decreases with each successive update of KF parameters. Analysis of the matrices across consecutive CLDA updates reveals a clear explanation for this phenomenon. Figure 3A shows the changes (relative to their initial values) in the cursor speed and norms of the Kalman gain (top) and norms of the KF matrices (state transition model, middle; observation model, bottom) across Batch CLDA sessions where all KF parameters are adapted. Thick lines show the mean across all sessions ($n = 9$; 4 *VFB* seeds, 5 *Contra* seeds), and shading shows standard errors. (Note that not all sessions contained the same number of Batch CLDA updates.) The mean predicted cursor speed decreased significantly with each CLDA update. Interestingly, the Kalman gain shows a parallel decrease in amplitude (estimated via the matrix norm). Inspecting the KF matrices reveals large and rapid decreases in W (by an order of magnitude over the course of CLDA) and significant increases in C. A and Q matrices, by contrast, show no clear trends.

As noted above, the covariance matrix W plays a crucial role in determining the relative contributions of the state and observation models. Smaller W matrices indicate that the KF is more confident in its linear state transition model, and thus will give less weight to the neural data when forming a prediction (reflected by a smaller Kalman gain). We see that the A matrix of the state transition model does not change significantly, but rather that the confidence in this model increases with each CLDA update. State transition models in BMIs (and models of natural arm movements) typically have

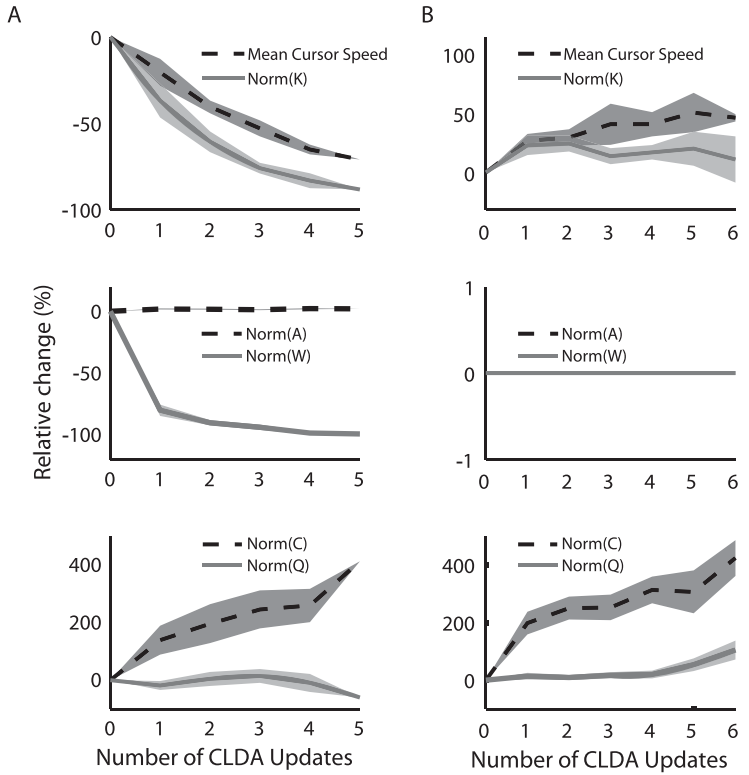

Figure 3: Advantages of CLDA with selective parameter adaptation for a Kalman filter decoder. Changes (relative to their initial values) in mean cursor speed and norm of the Kalman gain (top row), and KF model matrices (middle and bottom rows) across Batch CLDA sessions. (A) Adaptation of all KF parameters $\{A, W, C, Q\}$ leads to dramatic decreases in cursor speed. (B) Selective adaptation of only the KF state observation model parameters $\{C, Q\}$ avoids undesirable decreases in cursor speed.

damped velocity dynamics (Gilja et al., 2010, 2012). Thus, decreasing the weight on the system input—the neural data—inherently reduces cursor speed. As such, the observed decreases in W likely contribute significantly to the cursor slowing that occurs when all KF parameters are updated.

To test this hypothesis, consecutive Batch updates were performed while holding the state transition model (A and W) fixed. The speed, Kalman gain, and KF matrices for these sessions are shown in Figure 3B (formatting identical to Figure 3A; $n = 19$, all *VFB* seeds). Here, we see that fixing A and W eliminates cursor slowing. In fact, cursor speeds show a slight increase, accompanied by a small increase in the Kalman gain. Changes in the

observation model are similar to changes in sessions where all matrices were adapted. This selective adaptation improved performance more successfully than full adaptation because it did not inherently limit cursor speed.

While all KF matrices contribute to cursor movement, and thus are candidates for adaptation, it is important to consider their physical interpretations and the system as a whole before selecting what parameters to adapt. Given that the state transition model represents cursor kinematics, intuition suggests that the model should not change dramatically. In fact, the A matrices show little to no change, as expected. However, the cursor movements used for decoder parameter updates are themselves generated by a KF with kinematics specified by the decoder. Thus, the KF becomes increasingly confident in its model of cursor movement, driving decreases in W and corresponding decreases in cursor speed. If we instead interpret the KF as having a fixed model of cursor kinematics and adapt only the mapping between the subject's neural activity and cursor movement, we avoid this problem of recursively refitting a kinematics model on itself. For this reason, SmoothBatch modifies only the C and Q matrices of the KF. Similarly, other CLDA algorithms for the KF update only these parameters as well (Li et al., 2011).

BMI decoders typically contain multiple parameters, and therefore deciding which parameters to update is an essential design choice for CLDA algorithms. Our experiments have demonstrated that CLDA algorithms for KF decoders should not adapt the state transition model parameters, as this leads to reductions in cursor speed and other undesirable effects. More generally, these results may suggest that it is undesirable to update any decoder parameters that represent a prior model on movement. Instead, CLDA algorithms should primarily focus on adapting parameters that represent the mapping between the subject's neural activity and their intended movements.

3.3 Smooth Decoder Updates. CLDA algorithms typically collect and process real-time data, such as the user's neural activity and BMI cursor movements, to form parameter updates that will improve performance. However, occasionally an algorithm may encounter unreliable data that, if not properly addressed, can have a negative impact on performance. For instance, if the BMI user is distracted or not paying attention for a short time during data collection, the parameter updates formed from those data could be inaccurate. Such unreliable data can be especially concerning for certain types of CLDA algorithms, such as the Batch algorithm. In these types of algorithms, new parameter estimates completely overwrite and replace the decoder's current parameters at every CLDA update. Whenever the algorithm encounters unreliable data, current decoder parameters are replaced with new, inaccurate estimates, and the BMI user could struggle to adapt accordingly. One potential solution to this problem is to detect and

then reject unreliable data using either an automated algorithm or manual supervision. However, for CLDA algorithms that aim to improve control accuracy when starting from low performance, most initially collected data are at least to some degree unreliable. For example, a 2D BMI cursor's movements can often appear random and not goal directed when initial control is severely limited. In these situations, the subject could either be (unsuccessfully) attempting to control the cursor or simply distracted and disengaged from the task. Since it is difficult to distinguish between these cases and reject the unreliable data, a CLDA algorithm has no choice but to operate on the entire data set. In addition, since the cursor in this example may not explore the full work space, there may not be sufficient variability in the corresponding collected batch of data to form accurate parameter estimates that could enable cursor control in all directions.

For CLDA algorithms that aim to operate from limited initial performance, one proven solution to encountering unreliable data is to perform smooth decoder updates. For instance, the SmoothBatch algorithm (Orsborn et al., 2012) is designed to facilitate gradual updates of parameters rather than completely overwriting them. SmoothBatch performs a weighted average of current parameters with new parameter estimates, which helps prevent a single unreliable batch of data from causing drastic parameter changes that could impede improvements in performance. If fact, it can be shown that SmoothBatch's update rules can be formally interpreted as the solutions to the following optimization problems:

$$C^{(i)} = \arg\min_{C} \left[(1-\rho) \|C - \hat{C}^{(i)}\|_F^2 + \rho \|C - C^{(i-1)}\|_F^2 \right], \tag{3.1}$$

$$Q^{(i)} = \arg\min_{Q} \left[(1-\rho) \|Q - \hat{Q}^{(i)}\|_F^2 + \rho \|Q - Q^{(i-1)}\|_F^2 \right], \tag{3.2}$$

where $\|M\|_F$ denotes the Frobenius norm of a matrix M, which is defined as

$$\|M\|_F \triangleq \sqrt{\sum_i \sum_j |m_{ij}|^2} = \sqrt{\mathrm{Tr}(M^T M)}. \tag{3.3}$$

The cost functions in equations 3.1 and 3.2 are designed to meet two competing objectives: producing updated parameters that are near the batch maximum likelihood estimates yet still close to the previous parameter setting. SmoothBatch's update rules therefore achieve a balance of both objectives that can be adjusted as desired by setting the weighting parameter $\rho \in (0, 1)$. For instance, as $\rho \to 1$, the cost functions increasingly penalize deviations from the previous parameter setting, $C^{(i-1)}$ and $Q^{(i-1)}$, forcing SmoothBatch to make very smooth parameter updates. As $\rho \to 0$, the cost functions instead penalize deviations from the Batch parameter estimates,

$\hat{C}^{(i)}$ and $\hat{Q}^{(i)}$. In the special case of $\rho = 0$, SmoothBatch reduces to the Batch algorithm:

$$C^{(i)} = \hat{C}^{(i)} \quad \text{and} \quad Q^{(i)} = \hat{Q}^{(i)}.$$

Therefore, SmoothBatch is effectively a regularized version of the Batch algorithm that directly aims to achieve smooth parameter updates. When initial BMI performance is limited and unreliable data are difficult to avoid, CLDA algorithms like SmoothBatch that ensure smoothness in their parameter updates can avoid large, disruptive parameter changes. By mitigating the impact of unreliable batches of data on performance, CLDA algorithms with smooth parameter updates can help ensure that a high level of BMI performance will eventually be reached.

3.4 Intuitive CLDA Parameters. *CLDA parameters* refers to the step sizes, learning rates, and other algorithmic parameters of a particular CLDA algorithm. The ability of a CLDA algorithm to effectively facilitate BMI performance improvements will greatly depend on the settings of these CLDA parameters. Note that these are distinct from decoder parameters such as the KF model matrices $\{A, W, C, Q\}$. A common approach to identifying a favorable setting of algorithmic parameters is to conduct a rigorous parameter sweep. While this technique works well offline, it is often infeasible in an online experimental setting, especially if any CLDA parameters lack intuitive interpretations. If there is no reference for what a parameter's order of magnitude should be, the parameter test space increases significantly, making a vast parameter sweep time-consuming or even infeasible. Given the inherent time constraints involved when performing closed-loop BMI experiments, this approach quickly becomes impractical.

In contrast, designing CLDA parameters with intuitive interpretations can greatly simplify parameter selection. As an example, the SmoothBatch algorithm's weighting parameter ρ is designed to implement a clear and intuitive linear trade-off between achieving the maximum likelihood estimates of parameters and maintaining the current parameter setting. Since ρ is restricted to the range $[0, 1]$, the experimenter can perform a much more focused parameter sweep in order to identify an appropriate value. In addition, the clear interpretation of both boundary values helps provide an intuitive feel for the parameter. Moreover, another useful interpretation of ρ can be obtained by recursively expanding SmoothBatch's update rules (e.g., for C):

$$\begin{aligned} C^{(i)} &= (1-\rho)\hat{C}^{(i)} + \rho C^{(i-1)} \\ &= (1-\rho)\hat{C}^{(i)} + \rho\big((1-\rho)\hat{C}^{(i-1)} + \rho C^{(i-2)}\big) \\ &= \dots \end{aligned}$$

$$= (1-\rho)\left(\hat{C}^{(i)} + \rho\hat{C}^{(i-1)} + \cdots + \rho^i\hat{C}^{(1)}\right) + \rho^i C^{(0)}$$
$$= (1-\rho)\sum_{j=0}^{i}\rho^j\hat{C}^{(i-j)} + \rho^i C^{(0)}, \tag{3.4}$$

where $C^{(0)}$ represents the initial (seed) value of the matrix. From this expansion, we see that SmoothBatch's update rules effectively implement an exponentially weighted moving average of past maximum likelihood estimates. In other words, the weights attached to past maximum likelihood estimates experience exponential decay. Thus, in addition to being a weighting parameter, ρ also has an intuitive interpretation as the weighting factor of an exponentially weighted moving average.

Furthermore, this reformulation reveals another intuitive parameterization of the algorithm. Since the maximum likelihood estimates $\{\hat{C}^{(i)},\ i \geq 1\}$ occur T_b seconds apart, one can define a half-life h as

$$\rho^{h/T_b} = \frac{1}{2}. \tag{3.5}$$

The half-life h represents the time it takes for a previous maximum likelihood estimate's weight in the decoder to be reduced by a factor of two. Therefore, the SmoothBatch CLDA parameters $\{T_b, \rho\}$ can be conveniently reparameterized as $\{T_b, h\}$. Since both the batch period and the half-life have units of time, this reparameterization provides yet another intuitive representation of SmoothBatch's CLDA parameters.

Overall, the multiple interpretations of ρ and its ability to be reparameterized into the half-life demonstrate the intuitive nature of SmoothBatch's algorithmic parameters. When a CLDA algorithm is designed like SmoothBatch to have CLDA parameters with clear and intuitive interpretations, appropriate values of these parameters will be simpler to choose in an experimental setting, thereby maximizing the algorithm's ability to rapidly improve closed-loop performance.

4 Convergence Analysis

Ideally, CLDA algorithms should be designed to make the decoder's parameters converge rapidly to or maintain values that are optimal for high-performance BMI control. Presumably, for every decoding algorithm, there exist optimal settings of decoder parameters that allow the user to maximize BMI performance. Since decoder parameters are typically continuous valued, it is unrealistic to expect a CLDA algorithm to achieve an optimal parameter setting exactly. Instead, a CLDA algorithm should aim to make decoder parameters converge close to an optimal setting. Analyzing the design of a CLDA algorithm with respect to this objective raises

multiple important questions. How close to this optimal setting will decoder parameters converge? How fast does convergence occur? How do CLDA parameters affect convergence, and what trade-offs are inherent in the choices of these parameters? How does the decoder's seeding method affect convergence? While conducting closed-loop BMI experiments is ultimately the only definitive way to address these questions conclusively, such experiments are often lengthy and costly.

Developing a method to predict and analyze the convergence properties of CLDA algorithms, before testing them experimentally, would greatly facilitate and expedite the CLDA algorithm design process. While closed-loop simulation methods could potentially be used to study convergence, these tools have not yet been used for this purpose (Héliot, Ganguly, et al., 2010; Cunningham et al., 2011). In this section, we explore a novel paradigm: mathematical convergence analysis under reasonable model assumptions. Using mean-square errors and KL divergence to quantify a CLDA algorithm's convergence ability, we can analyze how an algorithm will perform under different settings of its CLDA parameters. To demonstrate the utility of our analysis, we apply our measures to the SmoothBatch algorithm to make predictions about and gain insight into its convergence properties. Experimental testing with two monkeys performing a BMI task across 72 sessions validates the utility of our convergence measures. These results demonstrate that mathematical convergence analysis is an effective analytical tool that can ultimately inform the design of CLDA algorithms.

4.1 Proposed Convergence Measures. Each time a CLDA algorithm's update rules are applied, some of the decoder's parameters are changed. If we consider the evolution of one of these parameters as forming a sequence, we can then investigate whether this sequence will converge near the corresponding optimal parameter value. Notationally, we use a superscript i to denote the ith term in the sequence. For example, $\theta^{(0)}$ denotes the initial (seed) value of a parameter θ, $\theta^{(1)}$ denotes its value after the first CLDA update, and so on.

Let θ^* denote the optimal value of some parameter θ. To quantify convergence, we seek some measure of the deviation of $\theta^{(i)}$ from θ^*. One candidate measure is the difference between the expected value of $\theta^{(i)}$ and the optimal value θ^*:

$$\mathbb{E}[\theta^{(i)}] - \theta^*.$$

Intuitively, one might expect that if a CLDA algorithm is effective at improving BMI performance, then for each decoder parameter θ, this difference should tend toward 0 as more parameter updates are performed:

$$\lim_{i \to \infty} \mathbb{E}[\theta^{(i)}] - \theta^* = 0.$$

Despite its simplicity, a critical shortcoming of this condition is that it is only a first-order convergence measure. In other words, it does not provide any measure of the "variance" of $\theta^{(i)}$ around θ^*, which could be very large even if the first-order condition holds. To measure this variance, we consider the following second-order measure, the (normalized) mean-square error (MSE) of a parameter θ at update iteration i:[3]

$$\mathrm{MSE}^{(i)}(\theta) \triangleq \frac{\mathbb{E}\big[\|\theta^{(i)} - \theta^*\|_F^2\big]}{\|\theta^*\|_F^2}. \tag{4.1}$$

Note that in the above definition, the added normalization ensures that $\mathrm{MSE}^{(i)}(\theta)$ is not affected by the general scaling of the parameter θ.

While a highly performing decoder will necessarily have low MSEs, all decoders with the same MSEs may not necessarily result in the same performance. Because of this problem, we propose using an additional, more probabilistic convergence measure, KL divergence (KLD), to support our analysis. Our intuition for using KL divergence is as follows. Let x_1^N and y_1^N be shorthand notation for intended kinematics $\{x_t\}_{t=1}^N$ and neural activity $\{y_t\}_{t=1}^N$, respectively. In addition, let $\theta^{(i)}$ and θ^* represent the set of decoder parameters at update iteration i and the optimal set of parameters, respectively (and let $p_{\theta^{(i)}}$ and p_{θ^*} be the distributions on $\{x_1^N, y_1^N\}$ induced by these parameter sets). Then the KL divergence between θ^* and $\theta^{(i)}$ is

$$\begin{aligned} KLD(\theta^{(i)}) &= D_{KL}(p_{\theta^*} \| p_{\theta^{(i)}}) \\ &= \mathbb{E}\left[\log \frac{p(x_1^N, y_1^N|\theta^*)}{p(x_1^N, y_1^N|\theta^{(i)})}\right]. \\ &= \mathbb{E}\big[\log p(x_1^N, y_1^N|\theta^*)\big] - \mathbb{E}\big[\log p(x_1^N, y_1^N|\theta^{(i)})\big], \end{aligned}$$

where we have defined the shorthand notation $KLD(\theta^{(i)})$. In other words, it is the difference between the expected log likelihood under the distributions induced by θ^* and $\theta^{(i)}$, respectively. Our intuition for using KL divergence as a convergence measure is as follows: if $KLD(\theta_1) = KLD(\theta_2)$ for two sets of parameters θ_1 and θ_2, then the averaged probability of observing $\{x_1^N, y_1^N\}$ is the same under both θ_1 and θ_2. In other words, the KL divergence has

[3]We have invoked the expectation operator $\mathbb{E}[\cdot]$, because $\theta^{(i)}$ is a random variable before conducting a closed-loop CLDA experiment. However, after an experiment has been conducted, MSE is no longer a random quantity, but rather a sequence (indexed by i) that can be calculated from the corresponding sequence $\{\theta^{(i)}, i \geq 0\}$ (see the appendix for proposed methods of choosing θ^*). Therefore, it is important to note that experimental traces of MSEs are actually just SEs (squared errors).

a probabilistic interpretation of equality that is not present with MSE, our first convergence measure.

4.2 Case Study: SmoothBatch Algorithm. The convergence measures that we have proposed are general enough to apply to a large class of decoders and CLDA algorithms. However, to concretely demonstrate the application of these convergence measures to a real CLDA algorithm, we choose the SmoothBatch algorithm as a specific example in the sections that follow. SmoothBatch updates the observation model parameters of a Kalman filter decoder by periodically performing a weighted average of current parameters with maximum likelihood estimates from a new batch of data (see section 2.3). The SmoothBatch algorithm has multiple CLDA parameters: the weighting parameter ρ, the batch period T_b, and the half-life h. By using SmoothBatch's parameter update rules and applying our convergence measures for $\theta \in \{C, Q\}$, we can predict the trade-offs inherent in the choices of these parameters and gain valuable insight into SmoothBatch's convergence properties.

In order to facilitate our mathematical convergence analysis, we must first make certain model assumptions. To make our calculations tractable, we will assume that the KF models hold true: that there exist underlying matrices $\{A, W, C, Q\}$ for which equations 3.1 and 3.2 hold true. Furthermore, we will assume that a KF decoder with its parameters set to these matrices would allow the user to maximize BMI performance. Accordingly, since these matrices then effectively represent the optimal parameter setting, we will denote them as $\{A^*, W^*, C^*, Q^*\}$. Finally, we will assume that these true, underlying matrices are unchanging over time.

With these model assumptions, we can apply our convergence measures to analyze the SmoothBatch algorithm. In the following sections, we focus our attention directly on the results of our calculations, and we refer readers to the appendix for the full derivation of these results. Using the Frobenius matrix norm and evaluating equation 4.1 for $\theta \in \{C, Q\}$ by substituting in SmoothBatch's update rules, we find that SmoothBatch's MSEs for C and Q are of the form

$$\mathrm{MSE}^{(i)}(C) = \rho^{2i} \cdot \frac{\|C^{(0)} - C^*\|_F^2}{\|C^*\|_F^2} + (1 - \rho^{2i}) \frac{1-\rho}{1+\rho} \cdot \frac{f_C(T_b)}{\|C^*\|_F^2}, \tag{4.2}$$

$$\mathrm{MSE}^{(i)}(Q) = \rho^{2i} \cdot \frac{\|Q^{(0)} - Q^*\|_F^2}{\|Q^*\|_F^2} + (1 - \rho^{2i}) \frac{1-\rho}{1+\rho} \cdot \frac{f_Q(T_b)}{\|Q^*\|_F^2}, \tag{4.3}$$

where $f_C(T_b)$ and $f_Q(T_b)$ denote monotonically decreasing functions of the batch period T_b.

We can use the same model assumptions to apply our second convergence measure, KL divergence. For our KF decoder model, we find the KL divergence between the probability distributions under $\theta^* = \{C, Q\}$ and

$\theta^{(i)} = \{C^{(i)}, Q^{(i)}\}$ to be

$$\begin{aligned} KLD(\theta^{(i)}) &= D_{KL}(p_{\theta^*} \| p_{\theta^{(i)}}) \\ &= \frac{1}{2}\log\left(\frac{\det Q^{(i)}}{\det Q^*}\right) + \frac{1}{2}\left(\mathrm{tr}\left[Q^{(i)^{-1}} Q^*\right] - m\right) \\ &\quad + \frac{1}{2}\mathrm{tr}\left[(C^* - C^{(i)})^T Q^{(i)^{-1}} (C^* - C^{(i)}) M_x\right]. \end{aligned} \tag{4.4}$$

Due to the matrix inverses in equation 4.4 and the particular form of SmoothBatch's update rules, the KL divergence turns out not to be an ideal measure for predicting SmoothBatch's convergence properties. As a result, we will instead use KL divergence as a secondary convergence measure to both analyze SmoothBatch's actual experimental performance and validate predictions derived from our primary measure, MSE. It is important to note, however, that for other CLDA algorithms with different parameter update rules, KL divergence may actually be a more tractable measure than MSE, and therefore more useful for predicting convergence properties.

In the following sections, we further interpret equations 4.2 and 4.3 to analyze SmoothBatch's convergence properties and their dependence on the batch period T_b, the weighting parameter ρ, the half-life h, and the decoder's initial seeding.

4.3 Time Evolution of SmoothBatch's MSE. Since a CLDA algorithm's goal is to make decoder parameters converge close to the optimal parameter setting, it is critical to determine how close these parameters can get. Can we expect an algorithm's MSEs to decrease over time to zero, or will they instead settle at nonzero values? If the latter holds true, then how do choices of the algorithm's CLDA parameters affect these values? Our convergence analysis allows us to answer these questions and predict how SmoothBatch's MSEs evolve over time. Rearranging terms in equation 4.2, we can express SmoothBatch's MSE for C as

$$\mathrm{MSE}^{(i)}(C) = \rho^{2i} \underbrace{\left(\frac{\|C^{(0)} - C^*\|_F^2}{\|C^*\|_F^2} - \frac{1-\rho}{1+\rho}\frac{f_C(T_b)}{\|C^*\|_F^2}\right)}_{\triangleq \mathrm{MSE}_{tr}(C)} + \underbrace{\frac{1-\rho}{1+\rho}\frac{f_C(T_b)}{\|C^*\|_F^2}}_{\triangleq \mathrm{MSE}_{ss}(C)}.$$

Our analysis predicts that SmoothBatch's MSE for C can be decomposed into the sum of a transient term, $\mathrm{MSE}_{tr}(C)$, which decays as ρ^{2i} and a steady-state value, $\mathrm{MSE}_{ss}(C)$, which remains unchanged over time (the same analysis applies to SmoothBatch's MSE for Q). If the SmoothBatch algorithm is applied long enough, the transient terms will decay significantly and the steady-state terms will dominate the MSEs. Therefore, our analysis predicts

that SmoothBatch's MSEs for C and Q will settle to nonzero steady-state values:

$$\mathrm{MSE}_{\mathrm{ss}}(C) = \frac{1-\rho}{1+\rho}\frac{f_C(T_\mathrm{b})}{\|C^*\|_F^2} \quad \text{and} \quad \mathrm{MSE}_{\mathrm{ss}}(Q) = \frac{1-\rho}{1+\rho}\frac{f_Q(T_\mathrm{b})}{\|Q^*\|_F^2}. \tag{4.5}$$

From equation 4.5, we can analyze how SmoothBatch's steady-state MSEs are affected by changes in its CLDA parameters. For instance, since $f_C(T_\mathrm{b})$ and $f_Q(T_\mathrm{b})$ are both monotonically decreasing functions of the batch period T_b, increasing this parameter (for a fixed ρ) results in a decrease of the steady-state MSEs. Similarly, increasing the weighting parameter ρ (for a fixed T_b) or the half-life h (for a fixed ρ or T_b) has the same effect. Using these relationships, we have a principled way to adjust SmoothBatch's CLDA parameters in order to achieve lower steady-state MSE values. However, as we will see, adjusting these same parameters also affects another quantity: the transient decay rate.

4.4 Rate of Convergence. Although it is desirable for a CLDA algorithm to have low steady-state MSE values, it is equally important for it to achieve fast rates of convergence. For instance, even if one could design a CLDA algorithm with zero steady-state MSEs, the algorithm would be useless if its convergence rate was too slow for a real experiment. Fortunately, the transient terms of SmoothBatch's MSEs decay as ρ^{2i}, predicting that these MSEs exhibit exponential convergence to their steady-state values. In order to make this exponential convergence clearer, we can write ρ^{2i} in terms of the exponential function, thereby directly casting the transient's decrease over time as an exponential decay:

$$\mathrm{MSE}^{(i)} = \exp\left(-2i\ln\frac{1}{\rho}\right)\cdot \mathrm{MSE}_{\mathrm{tr}} + \mathrm{MSE}_{\mathrm{ss}}.$$

This equation holds for both C and Q. Moreover, by introducing the batch period T_b into both the numerator and denominator and using the fact that the product $T_\mathrm{b}i$ corresponds to the time of the ith CLDA update, we can roughly express MSE directly as a function of the time t:

$$\mathrm{MSE}^{(t)} \approx \exp\left(-\frac{2\ln\frac{1}{\rho}}{T_\mathrm{b}}t\right)\cdot \mathrm{MSE}_{\mathrm{tr}} + \mathrm{MSE}_{\mathrm{ss}}.$$

From this expression, we observe that by decreasing either ρ or T_b (while keeping the other parameter fixed), one can increase the rate of convergence and therefore allow SmoothBatch to reach its steady-state MSEs faster. Finally, using the definition of the half-life h in equation 3.5, we find that we

Table 2: Summary of the Predicted Effects of Changes to SmoothBatch's CLDA Parameters on the Transient MSE Decay Rate and Steady-State MSEs.

		Has the Following Effect on the:	
Increasing This CLDA Parameter	While Holding This Fixed	Transient MSE Decay Rate	Steady-State MSEs
ρ	T_b	Decrease	Decrease
	h	No change	Indeterminate
T_b	ρ	Decrease	Decrease
	h	No change	Indeterminate
h	ρ or T_b	Decrease	Decrease

can also express the rate of convergence as

$$\mathrm{MSE}^{(t)} \approx \exp\left(-\frac{2\ln 2}{h}t\right) \cdot \mathrm{MSE}_{\mathrm{tr}} + \mathrm{MSE}_{\mathrm{ss}}.$$

This equation allows us to express the convergence rate solely in terms of h and predicts that another way to increase the convergence rate is to simply decrease the half-life. In addition, from the properties of exponential decay, this result further implies that the time it takes for the transient MSE to decrease to a certain fraction (e.g., 0.01) of its original value is also proportional to h. Thus, our analysis predicts that h is directly tied to the convergence rate of SmoothBatch's MSEs, thereby confirming the notion of the half-life as an intuitive CLDA parameter as stated in section 3.4.

4.5 CLDA Parameter Trade-Offs. The predicted effects of changes to SmoothBatch's CLDA parameters on both the MSE decay rate and steady-state MSEs are summarized in Table 2. Note that for certain CLDA parameter changes, the predicted effects are indeterminate or result in no change. Overall, our analysis predicts that the SmoothBatch algorithm exhibits a fundamental trade-off between achieving a faster rate of convergence and lower steady-state MSEs. In general, increases in SmoothBatch's CLDA parameters lower its steady-state MSEs, but they also lead to a slower convergence rate, thereby increasing the total adaptation time required to reach a steady state. The opposite effect occurs when SmoothBatch's CLDA parameters are decreased.

4.6 Effect of Decoder's Seeding. Another important consideration is how decoder initialization affects different aspects of convergence. Since $C^{(0)}$ and $Q^{(0)}$ represent the initial values of the KF observation model, the

decoder's seeding clearly determines the initial MSEs before any closed-loop decoder adaptation has been performed:

$$\mathrm{MSE}^{(0)}(C) = \frac{\|C^{(0)} - C^*\|_F^2}{\|C^*\|_F^2} \qquad \text{and} \qquad \mathrm{MSE}^{(0)}(Q) = \frac{\|Q^{(0)} - Q^*\|_F^2}{\|Q^*\|_F^2}.$$

However, $C^{(0)}$ and $Q^{(0)}$ do not appear in any of our expressions for either the rate of convergence or the steady-state MSEs. Therefore, our analysis predicts that the decoder's seeding method does not affect either of these two quantities.[4] Since the steady-state MSE is a measure of the deviation of parameters from the optimal parameter setting—which in turn relates directly to performance—our convergence analysis predicts that the decoder's initial seeding should not affect the level of performance achievable using the SmoothBatch algorithm.

4.7 Improving the SmoothBatch Algorithm. Our analysis of the SmoothBatch algorithm has predicted a key trade-off between the convergence rate and the steady-state MSEs. However, with our new understanding of how SmoothBatch's CLDA parameters affect its convergence properties, we may be able to reengineer the algorithm to overcome this seemingly fundamental limitation and further improve the algorithm's performance. Specifically, we would like to modify SmoothBatch to achieve lower steady-state MSEs without significantly sacrificing the convergence rate. We propose allowing CLDA parameters to vary in time. Since the seed decoder's parameters may initially be far from their optimal setting, one could perform aggressive adaptation early to make large parameter updates that bring the parameters into the right ballpark. However, as decoder parameters near their optimal values, more finely tuned adaptation could be used to avoid potentially disruptive large parameter updates.

We propose using a time-varying version of SmoothBatch's ρ parameter, where $\rho^{(i)} \to 1$ as i increases so that parameter updates become smoother as time progresses. Letting $\rho \triangleq \rho^{(1)}$ and choosing a decay factor $\alpha < 1$, our proposed method for increasing $\rho^{(i)}$ over time is to make $1 - \rho^{(i)}$ decay exponentially as i increases:

$$1 - \rho^{(i)} \triangleq \alpha^{i-1}(1 - \rho).$$

Rearranging terms, we therefore propose the following systematic method for increasing $\rho^{(i)}$ over time:

$$\rho^{(i)} \triangleq 1 - \alpha^{i-1}(1 - \rho), \qquad \alpha < 1. \tag{4.6}$$

[4]The seeding method can still affect the total time to achieve convergence—for example, the time required for the transient MSEs to decay below a certain level—but our analysis predicts that it will not affect the rate at which the transient MSEs decay.

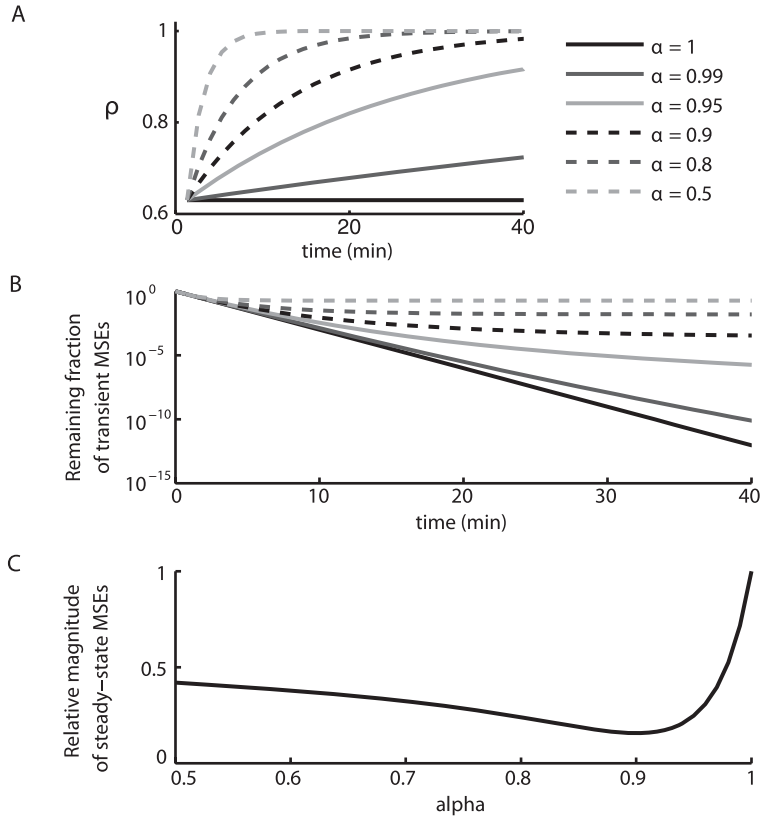

Figure 4: Improving the SmoothBatch CLDA algorithm by using a time-varying ρ parameter. (A) Trajectory of the weighting parameter ρ over time for different values of the decay factor α. (B) Predicted decay of the transient MSEs over time as closed-loop decoder adaptation is performed. (C) Predicted magnitude of the steady-state MSEs, relative to standard SmoothBatch, after 40 minutes of adaptation has been performed. For $\alpha \approx 0.9$, a significant reduction in the steady-state MSEs can be achieved with very little difference in how the transient MSEs decay. The batch period and the initial value of ρ were set to $T_b = 80$ s and $\rho^{(1)} = 0.63$ (corresponding to $h \approx 2$ minutes) for this example.

Figure 4 illustrates the predicted effects of implementing a time-varying weighting parameter for an example case where $T_b = 80$ s and $\rho^{(1)} = 0.63$. The decay factor α governs the rate at which the weighting parameter will increase over time (see Figure 4A). As closed-loop decoder adaptation is performed, the transient MSEs are predicted to decay differently depending on α (see Figure 4B, log scale). For standard SmoothBatch ($\alpha = 1$), the transient MSEs decay exponentially to zero, but when $\rho^{(i)}$ is time varying

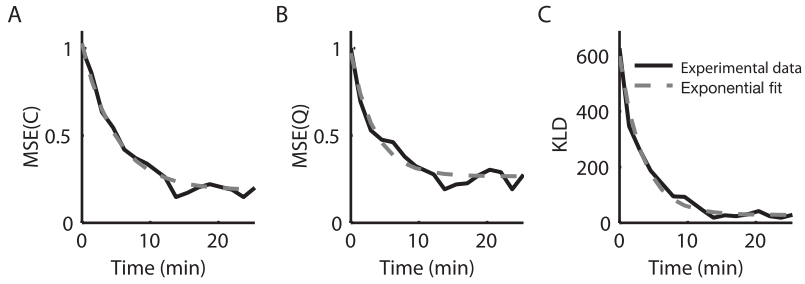

Figure 5: Exponential decay of transient MSEs. Experimental traces of (A, B) MSEs and (C) KLD, along with their exponential fits, for a session in which SmoothBatch was performed after VFB seeding.

($\alpha < 1$), they decay at slightly slower rates. If $\rho^{(i)}$ is increased too quickly (e.g., $\alpha \leq 0.8$), lower steady-state MSEs are achieved (see Figure 4C), but the transient MSEs no longer decay to zero (i.e., they are no longer "transient"). However, when the weighting parameter is increased at an appropriate rate (e.g., $\alpha \approx 0.9$), a significant, order-of-magnitude reduction in the steady-state MSEs can be achieved with very little difference in how the transient MSEs decay. In other words, our analysis predicts that with appropriate increases in the weighting parameter over time, we can overcome SmoothBatch's seemingly fundamental trade-off and achieve lower steady-state MSEs without significantly sacrificing the convergence rate.

4.8 Experimental Validation. To test our convergence predictions, we evaluated the performance of the SmoothBatch algorithm by conducting closed-loop experiments with two nonhuman primates performing a center-out task across 72 sessions. Figure 5 shows empirical traces of MSEs and the KLD, along with their exponential fits, for a sample session (see the appendix for details on the fitting procedure). As predicted by our convergence analysis, the MSEs indeed appear to decay exponentially and eventually settle to nonzero steady-state values. Across all sessions, exponential functions fit these traces more accurately than linear functions. The average squared error for exponential fits to traces of both MSEs and KLD was significantly smaller than for linear fits (Wilcoxon paired test, $p < 10^{-6}$).

Next, we analyzed the relationships between different settings of SmoothBatch's CLDA parameters and the corresponding empirically fit MSE decay rates and steady-state MSEs. Table 2 summarizes the predicted relationships for all of SmoothBatch's CLDA parameters. To simplify the results presented here, we focus only on the weighting parameter ρ. We restrict our analysis to sessions with more than one data point for a given ρ and with the same seeding condition (*VFB*), to isolate effects caused only

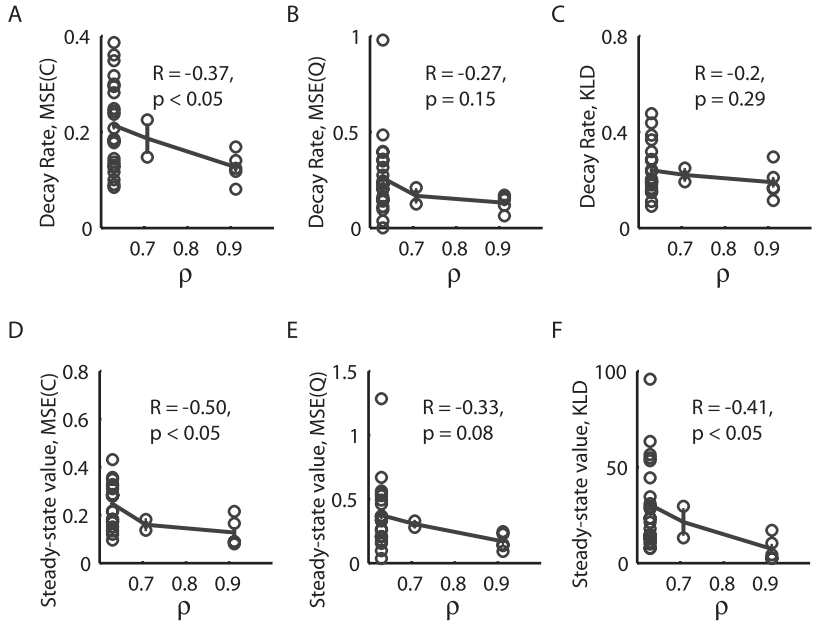

Figure 6: Empirically found relationships between SmoothBatch's ρ CLDA parameter and different aspects of convergence. (A–C) Experimental decay rates of (A, B) MSEs and (C) KLD, plotted versus the weighting parameter ρ across all sessions with VFB seeding. (D–F) Experimental steady-state values of (D, E) MSEs and (F) KLD, plotted versus the weighting parameter ρ across all sessions with VFB seeding.

by ρ ($n = 29$). As predicted by our convergence analysis, MSE decay rates are negatively correlated with ρ for both C ($r = -0.37$; $p < 0.05$) and Q ($r = -0.27$; $p = 0.15$) (see Figures 6A and 6B). In addition, steady-state MSE values are negatively correlated with ρ for both C ($r = -0.50$; $p < 0.05$) and Q ($r = -0.33$; $p = 0.08$), further confirming the predictions derived from our convergence analysis (see Figures 6D and 6E). Decay rates and steady-state values of KLDs showed similar correlations with ρ as MSEs (see Figures 6C and 6F).

Finally, we tested the prediction that the decoder's initial seeding method should not affect either the MSE decay rate or the steady-state MSEs. Figure 7 shows the decay rates (top row) and steady-state values (bottom row) for MSE(C) (left), MSE(Q) (middle), and KLD (right), separated by seeding condition. We restrict our analysis to sessions with the same value of ρ to isolate effects of initialization (60 sessions total: *VFB*, $n = 22$; *Baseline*, $n = 17$; *Shuffled*, $n = 11$; *Ipsi*, $n = 8$; *Contra*, $n = 2$). Bars indicate the mean, error bars show standard error of the mean, and points show

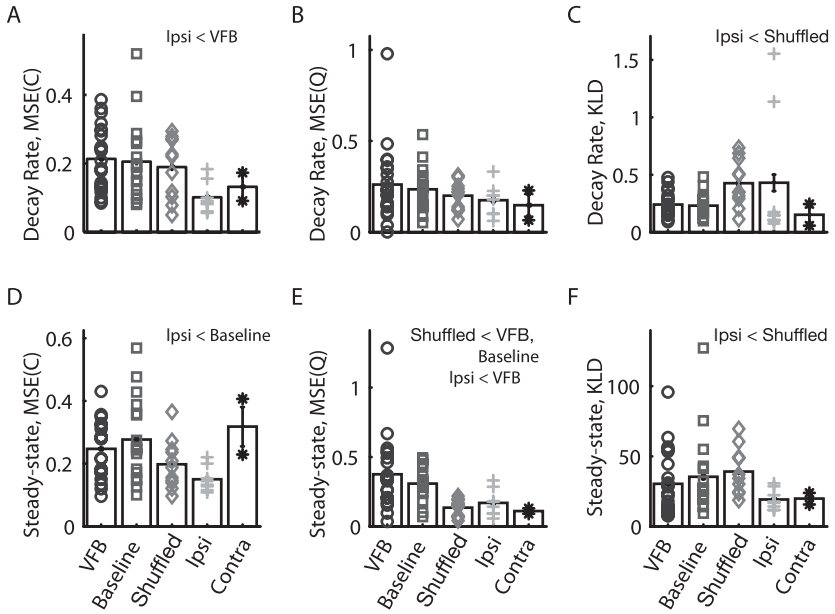

Figure 7: Effect of the decoder's seeding method on convergence. Experimentally observed (A–C) decay rates and (D–F) steady-state values for MSE(C), MSE(Q), and KLD, separated by seeding condition.

individual session data. A Kruskal-Wallis analysis of variance reveals some statistically significant differences (indicated on the figure) in decay rates and steady-state values. However, no seeding conditions show consistent differences across any of the KF parameters or convergence properties. Consistent with this result, previous work comparing behavioral improvements across seeding conditions revealed no significant differences across seeding conditions (Orsborn et al., 2012). Specifically, all sessions reached similar final performance, and the time required to achieve maximum performance did not depend on the decoder's initialization.

5 Discussion

Closed-loop decoder adaptation is a powerful paradigm for rapidly improving online BMI performance. In this article, we have established a general framework for the design and analysis of CLDA algorithms. We first identified a core set of design considerations that frequently arise when designing CLDA algorithms. We explored the consequences of choosing different timescales of adaptation, motivated the idea of selective parameter adaptation, illustrated the need for performing smooth parameter updates,

and stressed the importance of having intuitive CLDA parameters. We then introduced mathematical convergence analysis as an effective design tool for predicting the convergence properties of a prototype CLDA algorithm before conducting closed-loop experiments. Using the SmoothBatch algorithm (Orsborn et al., 2012) as our case study, we demonstrated the ability to predict how CLDA parameters affect different aspects of convergence and understand the resulting trade-offs that are inherent in the choices of these parameters. We also characterized the effects of adjusting SmoothBatch's CLDA parameters and found that the algorithm exhibits a seemingly fundamental trade-off between the rate of convergence and the steady-state MSEs. By allowing for a time-varying weighting parameter, we found that we can achieve significantly lower MSEs with very little sacrifice of the rate of convergence. Although we demonstrated the utility of our convergence analysis using SmoothBatch as a particular example, our methods can generally be applied to a large class of other decoders and CLDA algorithms.

CLDA algorithms can operate on a variety of different decoders and can have different underlying goals, which may influence important aspects of their design. In clinical situations where motor deficits prevent patients from enacting the types of natural movements often used to seed the decoder, other methods of decoder initialization must be used (e.g., visual feedback seedings; see section 2.2) that may result in low initial performance. The SmoothBatch algorithm has been demonstrated to achieve high-performance BMIs in these settings (Orsborn et al., 2012). CLDA algorithms that aim to achieve different goals may differ from SmoothBatch in important aspects of their design. For instance, while SmoothBatch has been demonstrated to rapidly improve performance when operating on an intermediate timescale of adaptation, a longer timescale may be more appropriate for other algorithms with a different CLDA focus. As an example, Gilja et al. (2010, 2012) used batch maximum likelihood estimates of parameters to adapt a KF decoder. Using decoders seeded from contralateral arm movements and BMI performed during overt arm movements, subjects attained greater than 90% performance with the seed decoder in a center-out task. As a result, a single CLDA update formed from a 10 to 15 min batch of collected data was sufficient to achieve a significant improvement in reach kinematics. However, when attempting to improve performance from unfavorable seedings, such a large batch period would be counterproductive; it would force the BMI user to persist for a long time with what might be a poorly performing decoder, and thus would likely reduce subject engagement in the task. Therefore, a single update of decoder parameters would likely not achieve significant performance improvements. Instead, high performance in this case could be achieved only by performing multiple CLDA updates (Orsborn et al., 2012) or holding the decoder fixed long enough to facilitate the emergence of a stable motor memory (Ganguly & Carmena, 2009).

Different types of CLDA training signals may also be more appropriate for other CLDA algorithms. Li et al. (2011) recently developed a CLDA algorithm for a KF decoder that aims to maintain long-term BMI control accuracy using an adaptive method of self-training updates. Unlike SmoothBatch, Li et al.'s algorithm used overt arm movements to seed the decoder and was designed to sustain the accuracy of already high-performing decoders by adapting to neuronal plasticity and instability in neural recordings. As a result, the algorithm was able to directly use a Kalman smoothed version of decoder outputs as its training signal (it does not need to estimate the user's "intended kinematics"). In contrast, when SmoothBatch is used to improve performance from poor decoder seedings, the initial decoded cursor outputs are naturally a poor reflection of the user's intended kinematics, and therefore Kalman smoothing would not be an effective training signal.[5]

Despite differences in the operation and goals of CLDA algorithms, some design elements are still likely to be shared in common by many CLDA algorithms. For example, despite its vastly different purpose, Li et al.'s algorithm maintains the property of selective decoder adaptation like SmoothBatch and does not adapt the KF's transition model parameters. Many algorithms are also designed in one way or another to have smooth decoder updates. Taylor et al. (2002) developed a CLDA algorithm for a PVA decoder in which the next set of decoder weights was determined from a corrected version of the current weights along with past weights that resulted in good performance. Gage et al. (2005) estimated new KF parameters using a trial-by-trial sliding block of data, thus naturally allowing smoothness across successive estimates. Li et al.'s algorithm also achieved smooth updates, albeit in a characteristically different way, by endowing decoder parameters with probability distributions and updating them using Bayesian regression updates.

Closed-loop decoder adaptation should synergize well with previous results that have demonstrated the importance of neural plasticity in the BMI learning process (Taylor et al., 2002; Carmena et al., 2003; Jarosiewicz et al., 2008; Ganguly & Carmena, 2009; Ganguly, Dimitrov, Wallis, & Carmena, 2011; Koralek et al., 2012). With respect to the learning and retention of neuroprosthetic skill, a study by Ganguly and Carmena (2009) demonstrated the importance of a stable neural map. Some studies may have misinterpreted these results. For instance, contrary to Li et al.'s interpretation, Ganguly and Carmena did not assert that a fixed decoder may be sufficient for long-term control accuracy (Li et al., 2011). Rather, they showed that a "stable circuit," consisting of a fixed decoder and stable neurons, can facilitate the development of a stable neural map of the decoder that enables performance improvements, can be readily recalled across days,

[5] In experiments using SmoothBatch, we instead used Gilja et al.'s (2010, 2012) method for estimating intended cursor kinematics.

and is robust to interference from a second learned map. However, even if CLDA is used to achieve initial rapid gains in performance, it is likely that subsequent fixing of decoder parameters could still allow the formation of such a stable map.

While our convergence predictions are consistent with our experimental results, we still find some variability in our results that is not accounted for by our analysis. For instance, calculating convergence measures from experimental data requires explicit assumptions that could introduce variability (see the appendix). As an example, the true underlying mapping between the observed neural firing rates y_t and the kinematic state x_t of the cursor is unlikely to be exactly linear with additive gaussian noise. Existing work does suggest that incorporating nonlinearities into the KF could be beneficial (Li et al., 2009) and that CLDA can be successfully used with this decoding framework (Li et al., 2011). Exploring system nonlinearities may then be a fruitful avenue for future research. However, we feel that the design principles highlighted in this study of linear decoders will likely translate to nonlinear applications. Indeed, our findings regarding selective adaptation of KF matrices are consistent with the approach of Li et al. (2011) using a nonlinear KF.

Despite the variability, all of our data show clear trends consistent with our convergence analysis predictions (e.g., the signs of all calculated correlations are consistent with our predictions). Indeed, the goal of our convergence analysis is not to predict the exact value of the MSEs of different decoder parameters. Instead, what we are more interested in is finding the general form of the MSEs in order to understand both how they evolve over time and how they are affected by CLDA parameters (like SmoothBatch's ρ, T_b, and h). Furthermore, our methods should not only be able to evaluate a wide range of existing CLDA algorithms but also provide insights into ways of improving these algorithms. Our convergence analysis techniques accomplish these goals, thus demonstrating that they can be an effective analytical tool for evaluating and informing CLDA algorithm design.

In future work, we will continue to analyze the CLDA algorithm design process and develop new convergence analysis techniques. We will also continue using time-varying CLDA parameters to achieve more favorable convergence properties. Overall, our CLDA design principles and convergence analysis techniques will help us design the next generation of CLDA algorithms for BMI systems. Combined with neural plasticity, closed-loop decoder adaptation will serve as a powerful paradigm for rapidly achieving reliable and proficient neuroprosthetic control.

Appendix: Kalman Filter CLDA Convergence Measures

A.1 Calculating SmoothBatch's MSEs. Using the Frobenius matrix norm as the choice of norm in equation 4.1, SmoothBatch's MSE for C is

defined as

$$\mathrm{MSE}^{(i)}(C) \triangleq \frac{\mathbb{E}[\|C^{(i)} - C^*\|_F^2]}{\|C^*\|_F^2}.$$

To keep our calculations general, let us for now allow SmoothBatch's weighting parameter to be time varying (as proposed in section 4.6) and therefore denote it as $\rho^{(i)}$:

$$C^{(i)} = (1-\rho^{(i)})\hat{C}^{(i)} + \rho^{(i)} C^{(i-1)}, \qquad \rho^{(1)} \triangleq \rho.$$

Note that for the standard SmoothBatch algorithm, ρ is unchanging over time ($\rho^{(i)} = \rho$). Using the definition of the Frobenius norm and this time-varying update rule for C to expand the numerator term of the MSE, we have that

$$\begin{aligned}
\mathbb{E}[\|C^{(i)} - C^*\|_F^2] &= \mathbb{E}[\mathrm{tr}((C^{(i)} - C^*)^T (C^{(i)} - C^*))] \\
&= (\rho^{(i)})^2 \mathbb{E}[\|C^{(i-1)} - C^*\|_F^2] + (1-\rho^{(i)})^2 \mathbb{E}[\|\hat{C}^{(i)} - C^*\|_F^2] \\
&\quad + 2\rho^{(i)}(1-\rho^{(i)})\mathrm{tr}(\mathbb{E}[C^{(i-1)} - C^*]^T \mathbb{E}[\hat{C}^{(i)} - C^*]) \\
&= (\rho^{(i)})^2 \mathbb{E}[\|C^{(i-1)} - C^*\|_F^2] \\
&= +(1-\rho^{(i)})^2 \mathbb{E}[\|\hat{C} - C^*\|_F^2], \qquad \text{(A.1)}
\end{aligned}$$

where we switched the order of the trace and expectation as necessary (both are linear operators), made an independence assumption between $\hat{C}^{(i)}$ and $C^{(i-1)}$, and then used the fact that $\mathbb{E}[\hat{C}^{(i)} - C^*] = 0$ (i.e., the ML estimator for C is unbiased). Simulations show that under certain conditions, the second expectation term on the last line is a monotonically decreasing function of the batch period T_b, and therefore we will simply denote this term as $f_C(T_\mathrm{b})$:

$$f_C(T_\mathrm{b}) = \mathbb{E}[\|\hat{C} - C^*\|_F^2].$$

Applying equation A.1 recursively and then simplifying terms, we have that

$$\mathbb{E}[\|C^{(i)} - C^*\|_F^2] = A(i) \cdot \|C^{(0)} - C^*\|_\mathrm{F}^2 + B(i) \cdot f_C(T_\mathrm{b}),$$

where $A(i)$ and $B(i)$ are calculated to be

$$A(i) \triangleq \prod_{l=1}^{i} (\rho^{(l)})^2$$

$$B(i) \triangleq \sum_{l=1}^{i} \left((1-\rho^{(l)})^2 \prod_{k=l+1}^{i} (\rho^{(k)})^2 \right).$$

Since $\rho^{(i)} = \rho$ for the standard SmoothBatch algorithm, $A(i)$ and $B(i)$ simplify to

$$A(i) = \rho^{2i} \quad \text{and} \quad B(i) = (1 - \rho^{2i})\frac{1-\rho}{1+\rho}.$$

Therefore, SmoothBatch's MSE for C is

$$\text{MSE}^{(i)}(C) = \rho^{2i}\frac{\|C^{(0)} - C^*\|_F^2}{\|C^*\|_F^2} + (1 - \rho^{2i})\frac{1-\rho}{1+\rho}\frac{f_C(T_b)}{\|C^*\|_F^2}.$$

The approximate calculation of the MSE for Q follows a similar derivation. Although the ML estimator for Q is biased, we will assume that $\mathbb{E}[\hat{Q}^{(i)} - Q^*] = 0$, which holds approximately true when $N \gg k$ (where N is the number of data points used to form the ML estimate and k is the number of columns in C). As before, if we define a term $f_Q(T_b)$ as

$$f_Q(T_b) = \mathbb{E}[\|\hat{Q} - Q^*\|_F^2],$$

we find that SmoothBatch's MSE for Q is approximately of the form

$$\text{MSE}^{(i)}(Q) = \rho^{2i}\frac{\|Q^{(0)} - Q^*\|_F^2}{\|Q^*\|_F^2} + (1 - \rho^{2i})\frac{1-\rho}{1+\rho}\frac{f_Q(T_b)}{\|Q^*\|_F^2}.$$

A.2 KL Divergence Measure for KF Decoder. The KL divergence $D_{KL}(p_{\theta^*}\|p_{\theta^{(i)}})$ is a measure of the difference between the joint probability distributions induced by the parameter settings θ^* and $\theta^{(i)}$. We use the KL divergence as a convergence measure in the following way. Let $\{\bar{A}, \bar{W}\}$ represent the KF decoder's transition model parameters (set from manual control data in our case). Let $\theta^* = \{\bar{A}, \bar{W}, C^*, Q^*\}$ represent the union of $\{\bar{A}, \bar{W}\}$ with the optimal KF observation model parameters, and let $\theta^{(i)} = \{\bar{A}, \bar{W}, C^{(i)}, Q^{(i)}\}$ represent the union of $\{\bar{A}, \bar{W}\}$ with the KF decoder's observation model parameters after the ith set of CLDA updates. We then calculate that

$$KLD(\theta^{(i)}) = D_{KL}(p_{\theta^*}\|p_{\theta^{(i)}}) = \frac{1}{2}\log\left(\frac{\det Q^{(i)}}{\det Q^*}\right) + \frac{1}{2}(\text{tr}[Q^{(i)^{-1}}Q^*] - m) + \frac{1}{2}\text{tr}[(C^* - C^{(i)})^T Q^{(i)^{-1}}(C^* - C^{(i)})M_x],$$

where m is the number of rows or columns in Q. M_x is defined as $M_x \triangleq \Sigma_x + \mu_x \mu_x^T$, where μ_x is the mean of x_t and Σ_x solves the following equation:

$$\Sigma_x = \bar{A}^T \Sigma_x \bar{A} + \bar{W}.$$

A.3 Setting C^* and Q^*. The MSE and KLD measures assume the existence of optimal matrices C^* and Q^*. Since these matrices are unknown, evaluating our convergence measures for experimental data requires estimating them. Given that the optimal matrices are defined to be those that facilitate maximal performance, we approximated C^* and Q^* based on the subject's behavioral performance. Specifically, C^* and Q^* were set as the average of all C and Q matrices where the subject's performance exceeded a certain threshold, defined as 85% of the maximum performance achieved during SmoothBatch adaptation. Choosing C^* and Q^* in this way helps mitigate the effects of behavioral variability, such as those caused by changes in subject motivation. Performance was estimated using the success rate, which was calculated using 60 second nonoverlapping bins. Another candidate method for estimating the optimal matrices consisted of simply choosing the C and Q matrices that resulted in maximum performance during the session. The calculated empirical MSEs and KLDs and the resulting analytical conclusions were very similar with both methods.

A.4 Exponential Fitting. Experimental traces of MSEs were fit with exponential functions to estimate their decay rate. Because steady-state MSEs must be positive, they were estimated from the data as the mean of MSEs for the last 20% of the timeseries. The data were fit to the following function,

$$\mathrm{MSE}_{\mathrm{tr}}(t) - \mathrm{MSE}_{\mathrm{ss}} = a_1 e^{-a_2 t},$$

using an unconstrained nonlinear minimization function (Matlab's fminsearch) to minimize the squared error. Other methods of exponential fitting, including explicitly constraining $\mathrm{MSE}_{\mathrm{ss}} \geq 0$, produced similar results. Experimental data occasionally showed variability during early SmoothBatch adaptation. Fits were calculated using data from the maximum squared errors (observed within the first half of adaptation) to the end of adaptation. Typically this procedure removed a very limited number of data points (mean: 1.15, standard deviation: 1.57, median: 0). Exponential fitting of KLD traces was performed in the same way, and similar approaches were used to fit linear functions (with offset and slope parameters) to the data for comparison (see section 4.8).

Acknowledgments

This work was supported by the National Science Foundation Graduate Research Fellowship (A.L.O.), the American Heart Association predoctoral fellowship (A.L.O.), the Defense Advanced Research Projects Agency contract N66001-10-C-2008 (J.M.C.), and the National Science Foundation grant CBET-0954243 (J.M.C.).

References

Carmena, J. M., Lebedev, M. A., Crist, R. E., O'Doherty, J. E., Santucci, D. M., Dimitrov, D. F., et al. (2003). Learning to control a brain–machine interface for reaching and grasping by primates. *PLoS Biology*, *1*(2), e42.

Chapin, J. K., Moxon, K. A., Markowitz, R. S., & Nicolelis, M.A.L. (1999). Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex. *Nature Neuroscience*, *2*(7), 664–670.

Cheng, S., & Sabes, P. N. (2006). Modeling sensorimotor learning with linear dynamical systems. *Neural Computation*, *18*(4), 760–793.

Cunningham, J. P., Nuyujukian, P., Gilja, V., Chestek, C. A., Ryu, S. I., & Shenoy, K. V. (2011). A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces. *Journal of Neurophysiology*, *105*(4), 1932–1949.

Dangi, S., Gowda, S., Héliot, R., & Carmena, J. M. (2011). Adaptive Kalman filtering for closed-loop brain-machine interface systems. In *Proceedings of the IEEE Engineering in Medicine and Biology Society International Conference on Neural Engineering* (pp. 609–612). Piscataway, NJ: IEEE.

Ethier, C., Oby, E. R., Bauman, M. J., & Miller, L. E. (2012). Restoration of grasp following paralysis through brain-controlled stimulation of muscles. *Nature*, *485*(7398), 368–371.

Gage, G. J., Ludwig, K. A., Otto, K. J., Ionides, E. L., & Kipke, D. R. (2005). Naive coadaptive cortical control. *Journal of Neural Engineering*, *2*(2), 52–63.

Ganguly, K., & Carmena, J. M. (2009). Emergence of a stable cortical map for neuroprosthetic control. *PLoS Biology*, *7*(7), e1000153.

Ganguly, K., & Carmena, J. M. (2010). Neural correlates of skill acquisition with a cortical brain-machine interface. *Journal of Motor Behavior*, *42*(6), 355–360.

Ganguly, K., Dimitrov, D. F., Wallis, J. D., & Carmena, J. M. (2011). Reversible large-scale modification of cortical networks during neuroprosthetic control. *Nature Neuroscience*, *14*(5), 662–667.

Gilja, V., Chestek, C. A., Diester, I., Henderson, J. M., Deisseroth, K., & Shenoy, K. V. (2011). Challenges and opportunities for next-generation intracortically based neural prostheses. *IEEE Transactions on Bio-Medical Engineering*, *58*(7), 1891–1899.

Gilja, V., Nuyujukian, P., Chestek, C. A., Cunningham, J. P., Yu, B. M., Fan, J. M., et al. (2012). A high-performance neural prosthesis enabled by control algorithm design. *Nature Neuroscience*, *15*(12), 1752–1757.

Gilja, V., Nuyujukian, P., Chestek, C., Cunningham, J., Yu, B., Ryu, S., et al. (2010). High-performance continuous neural cursor control enabled by a feedback control perspective. In *Proceedings of the 2010 Computational and Systems Neuroscience Conference*. Norwood, NJ: Ablex.

Haykin, S. (2001). *Adaptive filter theory* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

Héliot, R., Ganguly, K., Jimenez, J., & Carmena, J. M. (2010). Learning in closed-loop brain-machine interfaces: modeling and experimental validation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B, Cybernetics*, *40*, 1387–1397.

Héliot, R., Venkatraman, S., & Carmena, J. M. (2010). Decoder remapping to counteract neuron loss in brain-machine interfaces. In *Proceedings of the Engineering in Medicine and Biology Society Annual International Conference of the IEEE* (pp. 1670–1673). Piscataway, NJ: IEEE.

Hochberg, L. R., Bacher, D., Jarosiewicz, B., Masse, N. Y., Simeral, J. D., Vogel, J., et al. (2012). Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature*, *485*(7398), 372–375.

Hochberg, L. R., Serruya, M. D., Friehs, G. M., Mukand, J. A., Saleh, M., Caplan, A. H., et al. (2006). Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, *442*(7099), 164–171.

Jarosiewicz, B., Chase, S. M., Fraser, G. W., Velliste, M., Kass, R. E., & Schwartz, A. B. (2008). Functional network reorganization during learning in a brain-computer interface paradigm. *Proceedings of the National Academy of Sciences of the United States of America*, *105*(49), 19486–19491.

Kim, S.-P., Simeral, J. D., Hochberg, L. R., Donoghue, J. P., & Black, M. J. (2008). Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia. *Journal of Neural Engineering*, *5*(4), 455–476.

Koralek, A. C., Jin, X., Ii, J. D. L., Costa, R. M., & Carmena, J. M. (2012). Corticostriatal plasticity is necessary for learning intentional neuroprosthetic skills. *Nature*, *483*(7389), 331–335.

Koyama, S., Chase, S. M., Whitford, A. S., Velliste, M., Schwartz, A. B., & Kass, R. E. (2010). Comparison of brain-computer interface decoding algorithms in open-loop and closed-loop control. *Journal of Computational Neuroscience*, *29*(1–2), 73–87.

Li, Z., O'Doherty, J. E., Hanson, T. L., Lebedev, M. A., Henriquez, C. S., & Nicolelis, M. A. L. (2009). Unscented-Kalman filter for brain-machine interfaces. *PLoS ONE*, *4*(7), e6243.

Li, Z., O'Doherty, J. E., Lebedev, M. A., & Nicolelis, M. A. L. (2011). Adaptive decoding for brain-machine interfaces through Bayesian parameter updates. *Neural Computation*, *23*(12), 3162–3204.

Mahmoudi, B., & Sanchez, J. C. (2011). A symbiotic brain-machine interface through value-based decision making. *PLoS ONE*, *6*(3), e14760.

Millán, J. del R., & Carmena, J. M. (2010). Invasive or noninvasive: Understanding brain-machine interface technology. *IEEE Engineering in Medicine and Biology Magazine*, *29*(1), 16–22.

Moritz, C. T., Perlmutter, S. I., & Fetz, E. E. (2008). Direct control of paralysed muscles by cortical neurons. *Nature*, *456*(7222), 639–642.

Musallam, S., Corneil, B. D., Greger, B., Scherberger, H., & Andersen, R. A. (2004). Cognitive control signals for neural prosthetics. *Science*, *305*(5681), 258–262.

O'Doherty, J. E., Lebedev, M. A., Hanson, T. L., Fitzsimmons, N. A., & Nicolelis, M. A. L. (2009). A brain-machine interface instructed by direct intracortical microstimulation. *Frontiers in Integrative Neuroscience*, *3*(20).

Orsborn, A. L., Dangi, S., Moorman, H. G., & Carmena, J. M. (2011). Exploring timescales of closed-loop decoder adaptation in brain-machine interfaces. In *Proceedings of the Engineering in Medicine and Biology Society Annual International Conference of the IEEE* (pp. 5436–5439). Piscataway, NJ: IEEE.

Orsborn, A., Dangi, S., Moorman, H., & Carmena, J. (2012). Closed-loop decoder adaptation on intermediate time-scales facilitates rapid BMI performance improvements independent of decoder initialization conditions. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *20*(4), 468–477.

Santhanam, G., Ryu, S. I., Yu, B. M., Afshar, A., & Shenoy, K. V. (2006). A high-performance brain-computer interface. *Nature*, *442*(7099), 195–198.

Serruya, M. D., Hatsopoulos, N. G., Paninski, L., Fellows, M. R., & Donoghue, J. P. (2002). Instant neural control of a movement signal. *Nature*, *416*(6877), 141–142.

Shpigelman, L., Lalazar, H., & Vaadia, E. (2008). Kernel-ARMA for hand tracking and brain-machine interfacing during 3D motor control. In D. Köller, D. Schuurmans, Y. Bengio, & L. Bottou (Eds.), *Advances in neural information processing systems*, *21*. Cambridge, MA: MIT Press.

Suminski, A. J., Tkach, D. C., Fagg, A. H., & Hatsopoulos, N. G. (2010). Incorporating feedback from multiple sensory modalities enhances brain–machine interface control. *Journal of Neuroscience*, *30*(50), 16777–16787.

Taylor, D. M., Tillery, S.I.H., & Schwartz, A. B. (2002). Direct cortical control of 3D neuroprosthetic devices. *Science*, *296*(5574), 1829–1832.

Velliste, M., Perel, S., Spalding, M. C., Whitford, A. S., & Schwartz, A. B. (2008). Cortical control of a prosthetic arm for self-feeding. *Nature*, *453*(7198), 1098–1101.

Wahnoun, R., He, J., & Helms Tillery, S. I. (2006). Selection and parameterization of cortical neurons for neuroprosthetic control. *Journal of Neural Engineering*, *3*(2), 162–171.

Wu, W., Black, M. J., Gao, Y., Bienenstock, E., Serruya, M., Shaikhouni, A., et al. (2003). Neural decoding of cursor motion using a Kalman filter. In S. Becker, S. Thrün, & K. Obermayer (Eds.), *Advances in neural information processing systems*, *15*. Cambridge, MA: MIT Press.

Received October 13, 2012; accepted January 10, 2013.