Continuous Closed-Loop Decoder Adaptation with a Recursive Maximum Likelihood Algorithm Allows for Rapid Performance Acquisition in Brain-Machine Interfaces

Siddharth Dangi
siddharthdangi@berkeley.edu

Suraj Gowda
surajgowda@berkeley.edu
Department of Electrical Engineering and Computer Sciences, 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.

Amy L. Orsborn
amy.orsborn@gmail.com
UC–Berkeley–UCSF Joint Graduate Group in Bioengineering, University of California, Berkeley, Berkeley, CA 94720, U.S.A.

Kelvin So
kelv.so@gmail.com

Maryam Shanechi
shanechi@berkeley.edu
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA 94720, U.S.A.

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

Closed-loop decoder adaptation (CLDA) is an emerging paradigm for both improving and maintaining online performance in brain-machine interfaces (BMIs). The time required for initial decoder training and any subsequent decoder recalibrations could be potentially reduced by performing continuous adaptation, in which decoder parameters are updated at every time step during these procedures, rather than waiting to update the decoder at periodic intervals in a more batch-based process. Here, we
present recursive maximum likelihood (RML), a CLDA algorithm that performs continuous adaptation of a Kalman filter decoder’s parameters. We demonstrate that RML possesses a variety of useful properties and practical algorithmic advantages. First, we show how RML leverages the accuracy of updates based on a batch of data while still adapting parameters on every time step. Second, we illustrate how the RML algorithm is parameterized by a single, intuitive half-life parameter that can be used to adjust the rate of adaptation in real time. Third, we show how even when the number of neural features is very large, RML’s memory-efficient recursive update rules can be reformulated to be computationally fast so that continuous adaptation is still feasible. To test the algorithm in closed-loop experiments, we trained three macaque monkeys to perform a center-out reaching task by using either spiking activity or local field potentials to control a 2D computer cursor. RML achieved higher levels of performance more rapidly in comparison to a previous CLDA algorithm that adapts parameters on a more intermediate timescale. Overall, our results indicate that RML is an effective CLDA algorithm for achieving rapid performance acquisition using continuous adaptation.

1 Introduction

Brain-machine interfaces (BMIs) aim to assist patients with neurological injuries and disease by enabling them to use their own neural activity to control external devices such as computer cursors or robotic arms, or even drive movements of their own body via muscle stimulation. Various forms of BMI control have been demonstrated in rodents (Chapin, Moxon, Markowitz, & Nicolelis, 1999; Gage, Ludwig, Otto, Ionides, & Kipke, 2005; Koralek et al., 2012), nonhuman primates (Serruya, Hatsopoulos, Paninski, Fellows, & Doro, 2002; Taylor, Tillery, & Schwartz, 2002; Carmena et al., 2003; Musallam, Corneil, Greger, Scherberger, & Andersen, 2004; Santhanam, Ryu, Yu, Afshar, & Shenoy, 2006; Velliste, Perel, Spalding, Whitford, & Schwartz, 2008; Moritz, Perlmutter, & Fetz, 2008; Jarosiewicz et al., 2008; Ganguly & Carmena, 2009; Suminski, Tkach, Fagg, & Hatsopoulos, 2010; Ethier, Oby, Bauman, & Miller, 2012), and humans (Hochberg et al., 2006; Kim et al., 2008; Hochberg et al., 2012; Collinger et al., 2013). Yet achieving neuroprostheses that are clinically viable still requires major advances in both reliability and performance (Millan & Carmena, 2010; Gilja et al., 2011).

At the heart of any BMI is a decoding algorithm (a decoder) that maps features of neural activity into output control signals for a prosthetic device. Decoders are typically seeded using neural activity collected while a subject performs a task with his or her natural arm (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 (Wahnoun, He, & Helms
Continuous Closed-Loop Decoder Adaptation

Tillery, 2006; Hochberg et al., 2006; Kim, Simeral, Hochberg, Donoghue, & Black, 2008; Velliste et al., 2008; Suminski et al., 2010). After the decoder has been initialized, two different mechanisms have been leveraged to facilitate performance improvements. The first, neural adaptation or neural plasticity, relies on the ability of neurons to adapt their receptive fields and tuning properties in order to perform actions required to achieve a desired goal (Fetz, 1969; Moritz et al., 2008; Ganguly et al., 2009; Koralek et al., 2012). The second, closed-loop decoder adaptation (CLDA), aims to update decoder parameters during closed-loop BMI operation in order to make the decoder’s output more accurately reflect the user’s intended BMI movements. A variety of CLDA algorithms have been developed that operate on different types of BMI decoders, including the population vector algorithm (Taylor et al., 2002), the Wiener filter (Heliot, Venkatraman, & Carmena, 2010), and the Kalman filter (Gage et al., 2005; Li, O’Doherty, Lebedev, & Nicolelis, 2011; Gilja et al., 2012; Orsborn, Dangi, Moorman, & Carmena, 2012).

A CLDA algorithm that enables rapid performance acquisition could present a variety of advantages in certain situations. When initial BMI performance with a seed decoder is poor, subjects may lose motivation and become disengaged from the task if performance improves too slowly. However, by using a CLDA algorithm during an initial calibration or training session that provides them with improved BMI performance more quickly, subjects would be more likely to remain engaged. After initial decoder training, such an algorithm could also prove useful in helping to maintain a high level of performance. For example, in a scenario where the user participates in a periodic (e.g., daily) decoder adjustment procedure to account for phenomena such as channel loss or electrode array shifts that disrupt BMI operation, such an algorithm could shorten the required recalibration time and allow performance to recover more rapidly.

One potential way to achieve rapid acquisition or recovery of performance is to design a CLDA algorithm that adapts decoder parameters on a fast timescale by performing continuous adaptation—in other words, updating parameters at every decoder time step during initial decoder training or subsequent recalibrations. Here, we introduce a recursive maximum likelihood (RML) algorithm that formulates the adaptation of Kalman filter (KF) parameters in terms of a weighted maximum likelihood estimation problem, which naturally allows more recently observed data to be more influential than past data in decoder updates. We demonstrate that RML possesses a variety of useful properties and practical algorithmic advantages. First, we show how, unlike some continuously adaptive methods such as stochastic gradient descent that can produce noisy individual updates, RML leverages the accuracy of updates based on a batch of data while still adapting parameters on every time step. Second, we illustrate how the algorithm’s rate of adaptation is parameterized by an easily interpretable half-life parameter that, unlike typical batch-based algorithms, can
be adjusted in real time. Third, we show how even if the number of neural features is very large, RML’s memory-efficient recursive update rules can be reformulated to avoid costly matrix inversions so that continuous adaptation remains feasible. Finally, although RML is designed for continuous adaptation, we illustrate how RML’s update rules are naturally modified to perform batch updates within the same algorithmic framework. We then test the RML algorithm in closed-loop experiments with three macaque monkeys trained to perform a 2D center-out cursor control task using either spiking activity or local field potentials. In comparison to SmoothBatch (Orsborn et al., 2012), a previous CLDA algorithm that adapts parameters on a more intermediate timescale, we show that RML achieves higher levels of performance following a short period of adaptation, with an average across subjects of 16% lower movement error, 18% lower movement variability, 17% lower reach times, and 13% lower normalized path length. Overall, our results demonstrate that RML is an effective CLDA algorithm for achieving rapid performance acquisition using continuous adaptation.

2 Neurophysiological Methods

2.1 Electrophysiology. Spiking activity (monkeys C and J) and local field potentials (monkey S) were used for BMI control in this study. All three monkey subjects were adult male rhesus macaques that were chronically implanted with bilateral microwire arrays of 128 Teflon-coated tungsten electrodes (35 μm diameter, 500 μm wire spacing, 8×16 array configuration; Innovative Neurophysiology, Durham, NC), targeting the arm areas of primary motor cortex (M1) and dorsal premotor cortex (PMd). Monkey C was also implanted bilaterally with 64-channel arrays (similar to those previously mentioned) in ventral premotor cortex (PMv), although units from these arrays were not used for BMI control. All procedures were conducted in compliance with the NIH Guide for Care and Use of Laboratory Animals and were approved by the University of California–Berkeley Institutional Animal Care and Use Committee.

Neural activity was recorded using an OmniPlex system (Plexon Inc., Dallas, TX) for monkey C and a MAP system (Plexon Inc.) for monkeys J and S. For monkey C, spiking activity was sorted using the PlexControl online sorting application (Plexon, Inc.), and only neural activity with well-identified waveforms (high signal-to-noise ratio and low variability in waveform shape) were used for BMI control. For monkey J, channel-level activity (Chestek et al., 2011) was defined by setting thresholds for each channel at 5.5 standard deviations from the mean while the subject sat quietly for 2 minutes at the start of each session. The SortClient sorting application (Plexon, Inc.) was then used to define unit templates, which typically captured all threshold crossings, in order to reject nonneuronal artifacts during BMI sessions. For monkey S, local field potential (LFP) signals were sampled at 1 kHz with channels referenced to ground, and signal
Continuous Closed-Loop Decoder Adaptation

2.2 Behavioral Task. The monkeys were head-restrained in a primate chair and had their arms confined within the chair while performing a self-paced 2D center-out task. Monkeys were previously trained to perform this task using their right arm. Figure 1 depicts the task structure and trial time line. Trials were initiated by moving the cursor under neural control to the center target and holding for 300 ms, after which the monkeys had to move the cursor to one of eight peripheral targets and hold for 300 ms to receive a liquid reward (target radii = 1.2 cm). If the monkeys entered a peripheral target but left before completing the required hold duration, a target hold error was assessed and no reward was given. The distance from the center to a peripheral target was 10 cm for monkey C and 6.5 cm for monkeys J and S. After every trial, the monkeys were required to move the cursor back to the center target to initiate the next trial. If the monkeys failed to hold or reach the target within 10 s, the trial was restarted without reward.

2.3 Feature Extraction. Neural data were streamed to a dedicated computer running Matlab (Mathworks, Natick, MA) or Python to implement feature extraction and closed-loop BMI control. For monkeys C and J, spike
counts binned at 100 ms were passed into the decoder as neural features for BMI control. For monkey, S, the LFP spectral power in consecutive 10 Hz bands from 0 to 80 Hz was estimated for each channel (in a 20-channel subset) using the multitaper method. Spectral estimation was performed every 100 ms using a sliding window containing the most recent 200 ms of raw LFP activity. The log spectral power estimates, across multiple frequency bands and LFP channels, were then used as neural features for BMI control. (See So, Dangi, Orsborn, Gastpar, & Carmena, 2014, for more details on LFP-based BMI methodology.)

2.4 Performance Evaluation. In addition to measuring the percentage of correct trials, we used the following metrics to assess the quality of successful reach trajectories:

1. Movement error (ME; cm): the average deviation of movement perpendicular to the reach direction; measures the straightness of the reach trajectory
2. Movement variability (MV; cm): the standard deviation of movement errors perpendicular to the movement direction; measures the consistency of the reach trajectory
3. Reach time (RT; secs): the time between the go cue and entering the peripheral target; measures the speed of the reach movement
4. Normalized path length (NPL; arbitrary units): the distance traveled between leaving the center and entering the target, divided by the straight-line distance; measures the “extra” distance traversed relative to a straight-line trajectory

3 Computational Methods

3.1 Decoder Model. We used a Kalman filter (KF) decoding algorithm (“decoder”) to implement closed-loop BMI control (Wu, Gao, Bienenstock, Donoghue, & Black, 2006). In this article, we focus on BMI cursor control, although our methods could apply more generally to other types of BMI systems. Let \( x_t \) and \( y_t \) be vectors representing the kinematic state of a computer cursor and observed neural features, respectively, at time \( t \). The KF state vector \( x_t \) is defined to include the position \( (p) \) and velocity \( (v) \) of the cursor at time \( t \), along both the horizontal and vertical directions of the screen:

\[
x_t = \begin{bmatrix} p_{\text{horizontal},t} & p_{\text{vertical},t} & v_{\text{horizontal},t} & v_{\text{vertical},t} & 1 \end{bmatrix}^T.
\]

In our experiments, \( y_t \) was composed of either spike counts binned at 100 ms (monkeys C and J) or log spectral power estimates in consecutive 10 Hz bands from 0 to 80 Hz (monkey S). The KF assumes the following models for how the cursor state \( x_t \) evolves over time and how the observed neural
Continuous Closed-Loop Decoder Adaptation

features \( y_t \) relate to the state:

\[
\begin{align*}
    x_t &= A x_{t-1} + w_t, \quad w_t \sim N(0, W), \\
    y_t &= C x_t + q_t, \quad q_t \sim N(0, Q),
\end{align*}
\]

where \( w_t \) and \( q_t \) are additive gaussian noise terms. The last column of \( C \) (corresponding to the constant 1 in \( x_t \)) accounts for nonzero means of neural observations, such as baseline firing rates.

In closed-loop 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 history of observed neural features \( \{y_t, y_{t-1}, y_{t-2}, \ldots\} \). The KF is a linear algorithm that performs this estimation recursively at each time step \( t \). In other words, it estimates \( x_t \) using only \( y_t \) and the previous estimate \( \hat{x}_{t-1} \). The actual KF equations that perform the state estimation at each time step can be found in Wu et al. (2006).

The KF model is parameterized by the matrices \( \{A, W, C, Q\} \). The transition model parameters \( A \) and \( W \) were trained using a data set of arm movements collected while the subject performed the center-out task in manual control, and these parameter estimates were used across all sessions. We constrained the structures of \( A \) and \( W \) so that these state transition model parameters modeled position as the integral of velocity plus noise (Gilja et al., 2012). In contrast, the observation model parameters \( C \) and \( Q \) were initialized (seeded) differently during each session using one of the following two methods:

1. Visual feedback (VFB): \( C \) and \( Q \) were trained using neural activity recorded as the subject passively viewed a cursor moving through the center-out task.
2. Shuffled: A VFB decoder was first trained, and the rows of \( C \) and rows and columns of \( Q \) were then randomly shuffled in order to randomize the assignment of decoder weights to neural features.

When decoder adaptation was performed during the initial training phase of a session, only the \( C \) and \( Q \) parameters of the KF were updated by the RML algorithm, since these parameters model the mapping between intended cursor movements and observed neural activity. Since \( A \) and \( W \) model cursor dynamics, which were not expected to vary from session to session, these parameters were held fixed during all sessions (Dangi, Orsborn, Moorman, & Carmena, 2013b).

In conjunction with CLDA, we simultaneously used an assistive control paradigm (Wang et al., 2013) during the initial decoder training only to temporarily assist the cursor toward the target. In this phase, the cursor trajectory was determined by

\[
\overrightarrow{v_{\text{cursor}}} = \alpha \cdot \overrightarrow{v_{\text{assist}}} + (1 - \alpha) \cdot \overrightarrow{v_{\text{user}}},
\]
where $\overrightarrow{v_{\text{user}}}$ is the decoded output from the Kalman filter, $\overrightarrow{v_{\text{assist}}}$ is a vector that points directly toward the current target, and $\overrightarrow{v_{\text{cursor}}}$ is a weighted average of the two that determines the final cursor output shown to the subject. The magnitude of $\overrightarrow{v_{\text{assist}}}$ was set to correspond to a speed of 2 cm per second if the cursor was not currently inside a target, and 0 cm per second otherwise. The weighted average was set by the assist level $\alpha \in [0, 1]$. The starting assist level (typically 0.6) was linearly decreased concurrently with CLDA, reaching 0 at the same time that CLDA was ceased. All performance analyses were conducted on data with no assist and with the cursor under full volitional control by the subjects.

### 3.2 Estimating Intended Movements

In the context of BMI cursor control, a Kalman filter decoder outputs a sequence $\{\hat{x}_t\}$ of decoded cursor kinematics over time. If the decoder’s parameters are not yet optimized or the user has not learned to use the decoder for accurate cursor control, the user will likely make movement errors when attempting to control the cursor. In order to perform closed-loop decoder adaptation during a calibration phase, one could estimate the user’s intended cursor kinematics over time, which we will denote as $\{\hat{x}_t\}$. Multiple methods for estimating intended kinematics have been presented in the literature (Shpigelman, Lalazar, & Vaadia, 2008; Gilja et al., 2012; Li et al., 2011; Shanechi & Carmena, 2013). In this present study, we chose to use Gilja et al.’s method (innovation 1 of the ReFIT-KF algorithm), which assumes that the user always intends to move the cursor directly toward the next target (Gilja et al., 2012). Given estimates $\{\hat{x}_t\}$ of the user’s intended cursor kinematics and simultaneously recorded neural features $\{y_t\}$, a CLDA algorithm such as RML can then update the KF decoder’s parameters in order to make future decoded outputs more accurately reflect the user’s underlying intention.

### 3.3 Prior CLDA Algorithm: SmoothBatch

The SmoothBatch CLDA algorithm has been previously demonstrated to improve BMI performance independent of the decoder’s initialization method (Orsborn et al., 2012) and has been used in both spike-based (Orsborn et al., 2012) and LFP-based (So et al., 2014) BMI experiments. To update the KF observation model parameters $C$ and $Q$, SmoothBatch first collects a batch of neural activity and cursor kinematics as the subject operates the BMI cursor in closed-loop control. Then the algorithm estimates the user’s intended cursor kinematics using, for example, innovation 1 of the ReFIT-KF algorithm (Gilja et al., 2012). Next, SmoothBatch constructs batch maximum likelihood estimates, $\hat{C}$ and $\hat{Q}$, of the $C$ and $Q$ matrices:

\[
\hat{C} = Y\hat{X}^T(\hat{X}\hat{X}^T)^{-1},
\]

\[
\hat{Q} = \frac{1}{N}(Y - \hat{C}\hat{X})(Y - \hat{C}\hat{X})^T,
\]
Continuous Closed-Loop Decoder Adaptation

where the $\tilde{X}$ and $Y$ matrices are formed by tiling $N$ columns of intended kinematics and recorded neural activity, respectively. Finally, the algorithm updates the observation model parameters using a weighted average:

$$C_{\text{new}} = (1 - \rho) \hat{C} + \rho C,$$

$$Q_{\text{new}} = (1 - \rho) \hat{Q} + \rho Q,$$

where $\rho \in [0, 1]$ controls the influence of $\hat{C}$ and $\hat{Q}$ on the new parameter settings. Once parameters have been updated, a new data batch starts being collected and the next iteration of the algorithm begins. Note that SmoothBatch does not update the KF transition model parameters $A$ and $W$. The parameter $\rho$ can be reparameterized in terms of a half-life $h$ (i.e., the time it takes for a previous maximum likelihood estimate’s weight in the decoder to be reduced by a factor of 2):

$$\rho^\frac{1}{h} = \frac{1}{2},$$

where $dt$ is the time between decoder time steps (typically 100 ms).

### 3.4 New CLDA Algorithm: Recursive Maximum Likelihood

Here, we introduce the recursive maximum likelihood (RML) algorithm for adapting the parameters of a KF decoder during closed-loop control. Let us assume that we have a method for generating estimates of the user’s intended cursor kinematics at each time step. Let $\tilde{x}_{1:n}$ and $y_{1:n}$ represent the collection of these intended kinematics and observed neural activity, respectively, up to time step $n$:

$$\tilde{x}_{1:n} \triangleq \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n\},$$

$$y_{1:n} \triangleq \{y_1, y_2, \ldots, y_n\}.$$

Let $\theta^{(n)} = \{C^{(n)}, Q^{(n)}\}$ represent the current KF observation model parameters that are used as part of the BMI decoder on time step $n$. In order to fully specify the RML algorithm, we must specify update rules that dictate how to produce $\theta^{(n+1)}$. To derive these update rules, let us consider the log-likelihood function $l(\theta^{(n+1)}; y_{1:n}, \tilde{x}_{1:n})$, which is defined as the log probability of observing the neural activity $y_{1:n}$ if these data originated from a model parameterized by $\theta^{(n+1)}$ (and $\tilde{x}_{1:n}$ was the corresponding set of intended cursor kinematics):

$$l(\theta^{(n+1)}; y_{1:n}, \tilde{x}_{1:n}) = \log p(y_{1:n}|\theta^{(n+1)}, \tilde{x}_{1:n})$$

$$= \sum_{t=1}^{n} \log p(y_t|\theta^{(n+1)}, \tilde{x}_t).$$
Here, one option (known as maximum likelihood estimation) would be to update the KF parameters by setting \( \theta^{(n+1)} \) to be the parameters that maximize the log likelihood:

\[
\theta^{(n+1)} = \arg \max_\theta \sum_{t=1}^{n} \log p(y_t|\theta, \tilde{x}_t).
\]

In this approach, all data points from \( t = 1 \) to \( n \) contribute equally to the objective. However, since more recently observed data are often more relevant for accurate parameter estimation than past data, it is often more desirable to assign greater weight or importance to more recent data points. By doing so, one can naturally allow more recently observed data to be more influential in decoder updates. Therefore, let us instead consider a weighted log-likelihood objective function \( l_w(\cdot) \) that contains a monotonically increasing weighting function \( \beta(t) \):

\[
l_w(\theta^{(n+1)}, y_1:n, \tilde{x}_1:n) = \sum_{t=1}^{n} \beta(t) \log p(y_t|\theta^{(n+1)}, \tilde{x}_t).
\]

For the RML algorithm, we choose an exponentially decaying weighting function:

\[
\beta(t) = \lambda^{n-t}, \quad (3.7)
\]

where \( \lambda \in (0, 1) \). RML’s update rules for \( C \) and \( Q \) are then derived by solving the following weighted maximum likelihood problem:

\[
\theta^{(n+1)} = \arg \max_\theta \sum_{t=1}^{n} \lambda^{n-t} \log p(y_t|\theta, \tilde{x}_t). \quad (3.8)
\]

Since \( y_t|\theta, \tilde{x}_t \sim \mathcal{N}(C\tilde{x}_t, Q) \), we have that

\[
\log p(y_t|\theta, \tilde{x}_t) = \text{const.} - \frac{1}{2} \log |Q| - \frac{1}{2} \left( y_t - C\tilde{x}_t \right)^T Q^{-1} \left( y_t - C\tilde{x}_t \right).
\]

We can then calculate the derivatives of the weighted log-likelihood objective in equation 3.8 with respect to \( C \) and \( Q \):

\[
\frac{\partial}{\partial C} \sum_{t=1}^{n} \lambda^{n-t} \log p(y_t|\theta, \tilde{x}_t) = Q^{-1} \sum_{t=1}^{n} \lambda^{n-t} (y_t - C\tilde{x}_t) \tilde{x}_t^T,
\]

\[
\frac{\partial}{\partial Q} \sum_{t=1}^{n} \lambda^{n-t} \log p(y_t|\theta, \tilde{x}_t) = -\sum_{t=1}^{n} \lambda^{n-t} (y_t - C\tilde{x}_t) (y_t - C\tilde{x}_t)^T.
\]
Continuous Closed-Loop Decoder Adaptation

\[
\frac{\partial}{\partial Q} \sum_{t=1}^{n} \lambda^{n-t} \log p(y_t|\theta, \tilde{x}_t) \\
= -\frac{1}{2} \left( \sum_{t=1}^{n} \lambda^{n-t} \right) Q^{-1} + \frac{1}{2} Q^{-1} \left( \sum_{t=1}^{n} \lambda^{n-t} (y_t - C\tilde{x}_t)(y_t - C\tilde{x}_t)^T \right) Q^{-1}.
\]

Setting these derivatives equal to 0 and solving and using the fact that
\[
\sum_{t=1}^{n} \lambda^{n-t} = \frac{1 - \lambda^n}{1 - \lambda},
\]
we arrive at the following basic rules for updating \(C\) and \(Q\) after the \(n\)th time step of the Kalman filter:

\[
C^{(n+1)} = \left( \sum_{t=1}^{n} \lambda^{n-t} y_t\tilde{x}_t^T \right) \left( \sum_{t=1}^{n} \lambda^{n-t} \tilde{x}_t\tilde{x}_t^T \right)^{-1}, \tag{3.9}
\]
\[
Q^{(n+1)} = \frac{1 - \lambda}{1 - \lambda^n} \sum_{t=1}^{n} \lambda^{n-t} (y_t - C^{(n+1)}\tilde{x}_t)(y_t - C^{(n+1)}\tilde{x}_t)^T. \tag{3.10}
\]

3.5 Recursive Update Rules on Sufficient Statistics. One of the main advantages of choosing the exponentially decaying weighting function \(\beta(t)\) in equation 3.7 is that the resulting update rules (see equations 3.9 and 3.10) can be re-written in a simpler, recursive form so that they do not require storing histories \(\tilde{x}_1^n\) and \(y_1^n\). Let us define new parameters \(R, S, T,\) and EBS (effective batch size) as

\[
R^{(n+1)} \triangleq \sum_{t=1}^{n} \lambda^{n-t} \tilde{x}_t\tilde{x}_t^T, \\
S^{(n+1)} \triangleq \sum_{t=1}^{n} \lambda^{n-t} y_t\tilde{x}_t^T, \\
T^{(n+1)} \triangleq \sum_{t=1}^{n} \lambda^{n-t} y_t y_t^T, \\
\text{EBS}^{(n+1)} \triangleq \sum_{t=1}^{n} \lambda^{n-t}.
\]

Then, by substituting these definitions into equations 3.9 and 3.10, it is straightforward to show that we can express \(C^{(n+1)}\) and \(Q^{(n+1)}\) exclusively...
in terms of these new parameters as

\[ C^{(n+1)} = S^{(n+1)} \left( R^{(n+1)} \right)^{-1}, \]  

3.11

\[ Q^{(n+1)} = \frac{1}{EBS^{(n+1)}} \left( T^{(n+1)} - S^{(n+1)} R^{(n+1)^{-1}} S^{(n+1)^T} \right). \]  

3.12

From these equations, we see that once we have determined the updated values \( R^{(n+1)}, S^{(n+1)}, T^{(n+1)}, \) and \( EBS^{(n+1)} \), the data \( \tilde{x}_{1:n} \) and \( y_{1:n} \) are no longer needed to determine \( C^{(n+1)} \) and \( Q^{(n+1)} \). Therefore, an intuitive interpretation of \( R, S, T, \) and \( EBS \) is that they are sufficient statistics for the weighted maximum likelihood estimation of \( C \) and \( Q \).

The advantage of introducing these sufficient statistics—rather than adapting \( C \) and \( Q \) directly as in equations 3.9 and 3.10—is that they are easily updated at every time step in a recursive fashion. For example, for \( S^{(n+1)} \) we have

\[ S^{(n+1)} = \sum_{t=1}^{n} \lambda^{n-t} y_{t} \tilde{x}_{t}^T \]

\[ = \sum_{t=1}^{n-1} \lambda^{n-t} y_{t} \tilde{x}_{t}^T + \lambda^{n-n} y_{n} \tilde{x}_{n}^T \]

\[ = \lambda \left( \sum_{t=1}^{n-1} \lambda^{n-t-1} y_{t} \tilde{x}_{t}^T \right) + \lambda^{n-n} y_{n} \tilde{x}_{n}^T \]

\[ = \lambda S^{(n)} + \lambda y_{n} \tilde{x}_{n}^T, \]

and likewise for the others. Therefore, in order to determine the updated values \( C^{(n+1)} \) and \( Q^{(n+1)} \), we first simply update \( R^{(n)}, S^{(n)}, T^{(n)}, \) and \( EBS^{(n)} \) recursively as follows:

\[ R^{(n+1)} = \lambda R^{(n)} + \tilde{x}_{n} \tilde{x}_{n}^T, \]  

3.13

\[ S^{(n+1)} = \lambda S^{(n)} + y_{n} y_{n}^T, \]  

3.14

\[ T^{(n+1)} = \lambda T^{(n)} + y_{n} y_{n}^T, \]  

3.15

\[ EBS^{(n+1)} = \lambda EBS^{(n)} + 1, \]  

3.16

and then update \( C \) and \( Q \) as in equations 3.11 and 3.12. This parameter update procedure is depicted in Figure 2.

In some continuously adaptive methods such as stochastic gradient descent, parameters are adjusted at each time step based on a single data point.
Figure 2: RML algorithm block diagram. Block diagram illustrating how the RML algorithm is used to adapt the parameters of a Kalman filter decoder. \( z^{-1} \) blocks represent delay elements. Unlike other CLDA algorithms for KF decoders, RML first updates a set of sufficient statistics \( R, S, T, \) and EBS before updating the actual KF \( C \) and \( Q \) matrices directly. Importantly, however, such methods do not necessarily guarantee the accuracy of individual updates, but rather rely on the successive combination of multiple updates to produce accurate adjustments to parameters. While RML also performs continuous adaptation and uses a single data point at each time step in the update equations 3.13 to 3.16, those equations are used to update the RML sufficient statistics. Indeed, since the updated sufficient statistics are then subsequently used to update the decoder, each overall decoder update still represents the solution to a weighted batch maximum likelihood estimation problem. In this way, RML is able to leverage the accuracy of updates based on a batch of data while still adapting parameters on every time step.

3.6 Half-Life Reparameterization. Another advantage of the recursive form of RML’s update rules for the sufficient statistics is that \( \lambda \) can be reparameterized more conveniently in terms of an intuitive half-life parameter \( h \). For instance, starting from equation 3.14 and using repeated substitution,
one can show that for any number of time steps \( k \geq 0 \), we have that

\[
S^{(n+k)} = \sum_{t=0}^{k-1} \lambda^{-1-t} y_{n+t} x_{n+t}^T + \lambda^k S^{(n)}.
\]

In other words, we see that influence of the value \( S^{(n)} \) in the future version of the parameter \( S^{(n+k)} \) decreases exponentially in \( k \). By calculating when the factor \( \lambda^k \) is equal to one-half, we can define a half-life \( h \) as

\[
\lambda^h = \frac{1}{2},
\]

where \( dt \) is the time period (e.g., 100 ms) between successive Kalman filter decoder time steps. One of the advantages of a continuously adaptive CLDA algorithm like RML is the ability to instantaneously adjust adaptation rates, such as the half-life \( h \). In our closed-loop experiments testing the RML algorithm (see section 4), we start with a relatively low half-life of 30 s, which we continuously increase over the course of CLDA to a final value of 300 s. In this way, we begin CLDA by making large initial parameter updates that bring the parameters into the right ballpark, while smoothly transitioning toward more conservative, finely-tuned adaptation as decoder parameters begin to converge. While using a time-varying half-life for SmoothBatch has also been proposed in previous work (Orsborn et al., 2012; Dangi et al., 2013b), it has not yet been tested in closed-loop spike-based experiments. Therefore, in our BMI experiments, we compare RML against both the standard form of SmoothBatch with a nonchanging half-life (as originally tested in Orsborn et al., 2012) and SmoothBatch with a time-varying (albeit not continuously) half-life.

### 3.7 Avoiding Costly Matrix Inversions

The actual Kalman filter equations that perform state estimation require the computation of a Kalman gain matrix on every time step of filter. For instance, on time step \( n+1 \), the Kalman gain is computed as

\[
K_{n+1} = P_{n+1|n} C_{n+1} (C_{n+1} P_{n+1|n} C_{n+1}^T + Q_{n+1})^{-1},
\]

where \( P_{n+1|n} \) is the a priori estimation error covariance matrix (Wu et al., 2006). If \( C \) and \( Q \) are being adapted by a CLDA algorithm that does not update parameters on every time step, then \( P_{n+1|n} \) converges quickly to a constant matrix \( P \) after each update and repeated computation of the inverse in equation 3.18 can be avoided until the next update, since the inverse does not change. However, for a CLDA algorithm that performs continuous adaptation of \( C \) and \( Q \), this inverse is constantly changing and needs to be
calculated on every time step. When the number of neural features (which corresponds to the number of rows or columns of the matrix that needs to be inverted) is large, the resulting matrix inversion calculation may be too computationally expensive to be completed within a typical BMI decoder loop time (e.g., 50–100 ms). For example, in an LFP-based BMI system where 20 features are being extracted on each of 50 LFP channels, a large $1000 \times 1000$ matrix would need to be inverted.

To significantly reduce the computation involved in the Kalman gain calculation and make a continuously adaptive CLDA algorithm like RML computationally feasible when the number of neural features is large, we can apply a formula known as the Woodbury matrix identity (Press, Teukolsky, Vetterling, & Flannery, 1992). By applying this identity to the matrix inversion in the Kalman gain, we can compute this inverse in an alternate way that requires inverting a matrix of the same size as $P_{n+1|n} + C(n+1)^T Q(n+1)^{-1} C(n+1)$, which is typically very small (only $5 \times 5$ in our position-velocity KF decoders):

$$K_{n+1} = P_{n+1|n} C(n+1)^T Q(n+1)^{-1} \times \left[ I - C(n+1) \right] \cdot \left( P_{n+1|n} + C(n+1)^T Q(n+1)^{-1} C(n+1) \right)^{-1} C(n+1)^T Q(n+1)^{-1} \right]^{-1}. \quad (3.19)$$

Nonetheless, this alternate formula still requires the computation of $Q^{-1}$ on every time step, which would mean that a continuous adaptation approach would still be infeasible if $Q$ is a large matrix. However, two further applications of the Woodbury matrix identity can make RML a viable CLDA algorithm. First, by taking inverses on both sides of RML’s update rule for $Q$ and then applying the Woodbury identity, we have that

$$Q^{n+1} = E^t(n+1) \cdot \left[ T^{n+1} - T^{n+1} S(n+1) \right].$$

$$= \left( S^{n+1} T^{n+1} - R^{n+1} \right)^{-1} S(n+1) T^{n+1} \quad (3.20)$$

Second, if we apply the same identity to the update rule for $T$, we have that

$$T^{n+1} = \left( \lambda T^{n} + y_n y_n^T \right)^{-1}$$

$$= \frac{T^{n+1} - T^{n} y_n y_n^T T^{n+1}}{\lambda \left( \lambda + y_n^T T^{n+1} y_n \right)}. \quad (3.21)$$
Therefore, rather than updating $Q$ at every time step and then having to calculate its inverse in order to run the KF equations, we can instead choose to always update $Q^{-1}$ directly using equation 3.20 and use this value in equation 3.19. To overcome the fact that the update rule for $Q^{-1}$ still requires computing $T^{-1}$ (where $T$ has the same dimensions as $Q$), we see that we can also simply update $T^{-1}$ directly (rather than $T$) at every time step using equation 3.21, which is computationally efficient due to the Woodbury identity. Therefore, the above alternate update rules for $Q^{-1}$ and $T^{-1}$ can be used to achieve more efficient decoder updates when computational resources are scarce.

### 3.8 Generalization to Batch-Based Parameter Updates.

Although so far we have introduced RML exclusively as a CLDA algorithm for continuous adaptation, RML’s update rules can be easily modified to perform batch-based updates within the same algorithmic framework. Indeed, if we modify the weighted maximum likelihood estimation problem in equation 3.8 so that consecutive groups of $N \geq 1$ data points are weighted equally, then it is straightforward to show that we arrive at the following analogous update rules for our sufficient statistics:

\[
R^{(n+N)} = \lambda R^{(n)} + \sum_{t=0}^{N-1} \tilde{x}_{n+t} \tilde{x}_{n+t}^T, \tag{3.22}
\]

\[
S^{(n+N)} = \lambda S^{(n)} + \sum_{t=0}^{N-1} y_{n+t} y_{n+t}^T, \tag{3.23}
\]

\[
T^{(n+N)} = \lambda T^{(n)} + \sum_{t=0}^{N-1} y_{n+t} y_{n+t}^T, \tag{3.24}
\]

\[
EBS^{(n+N)} = \lambda EBS^{(n)} + N. \tag{3.25}
\]

As before, $C$ and $Q$ are still updated according to equations 3.11 and 3.12 whenever these sufficient statistics are updated. Intuitively, when $N = 1$, these update rules are equivalent to those presented before in equations 3.13–3.16. When $N > 1$, the $\lambda$ weighting factor is reparameterized into a half-life $h$ according to the following modified equation:

\[
\lambda = \frac{1}{2^h}. \tag{3.26}
\]

### 4 Results

#### 4.1 RML Decoder Adaptation.

Across all three subjects, the RML CLDA algorithm rapidly and reliably leads to acquisition of high performance after CLDA ceased and decoder parameters were held fixed.
Figure 3: Closed-loop BMI performance with RML. Example application of the RML algorithm (representative session from monkey J), starting from a KF decoder (seeded using the shuffled method) with which the subject could not perform successful trials. The subject’s apparent high performance from the start is due to the assistive paradigm that was used concurrently with CLDA. Performance is plotted in terms of (A) percentage and (B) rate of events per minute. The dotted vertical line indicates the time at which assistance was turned off, CLDA ceased, and decoder parameters were held fixed. (C) The subject’s best 40 consecutive reach trajectories in the center-out task after RML adaptation was performed.

Figure 3 illustrates monkey J’s performance during a representative session where the decoder was seeded using the shuffled procedure (see section 3.1), resulting in an initial decoder with which the monkey could not perform successful trials. A short period (5 minutes) of RML adaptation was then performed. An assistive control paradigm was used concurrently with CLDA, which accounts for the subject’s apparent high performance...
Figure 4: Decoder evolution during RML adaptation. Tuning plots representing the evolution (dark-to-light shaded lines) of the decoder's model of preferred velocity vectors, as CLDA is performed, for a subset of the units being used for BMI control. This sequence of tuning evolution corresponds to the same BMI session plotted in Figure 3.

The evolution of decoder parameters during this example session is shown in Figure 4. Since our KF state vector models cursor velocity, the rows of the KF $C$ matrix can be used to determine the preferred velocity directions of the different neural features being used for decoding (binned spike counts for monkeys C and J, and LFP spectral power for monkey S). By plotting the preferred velocity vectors for each neural feature as modeled in $C$, we can visualize the direction tuning of different units in
Continuous Closed-Loop Decoder Adaptation

the decoder and observe how the decoder changes over time as CLDA is performed. Figure 4 shows the evolution of the preferred velocity directions of a subset of units. With the shuffled seed decoder (darkest-shaded line for each unit), the subject is unable to perform successful trials. As RML decoder adaptation is performed (dark-to-light transition), the C matrix is rapidly updated to more closely reflect the true velocity tuning. By the end of adaptation (lightest-shaded lines), the subject could use the final decoder to skillfully control the cursor in the center-out task even after CLDA and assist were ceased.

4.2 Performance Comparison to SmoothBatch. In order to measure the performance of the RML CLDA algorithm, we compared it against SmoothBatch, a previously developed algorithm that has been demonstrated to rapidly improve BMI performance independent of the decoder's initialization method (Orsborn et al., 2012). In each session, starting from a shuffled decoder seeding, a KF decoder was adapted for a short amount of time (5 minutes; “short adaptation”), once each using either the RML or SmoothBatch algorithm (using a random ordering). During CLDA, we simultaneously used an assistive control paradigm (see section 3.1) in which assist started at 60% and decreased linearly to 0% by the end of adaptation. Since RML is a continuously adaptive algorithm, the half-life can be changed at every time step of the KF decoder. Therefore, for RML adaptation, a time-varying half-life was used that started at 30 s and increased linearly to 300 s over the course of adaptation. For SmoothBatch, prior work that originally introduced the algorithm reported that batch sizes of 60 s to 100 s and half-lives of 90 s to 210 s produced the most rapid performance improvements (Orsborn et al., 2012). Therefore, for this work, we chose to use a 60 s batch size and a 120 s half-life. (In additional experiments, a time-varying half-life was also used with SmoothBatch; see below.) When decoder adaptation and assist ceased after short adaptation, subjects then used the fixed decoder to perform the 2D center-out task. Sessions in which the monkeys did not perform successful trials following decoder adaptation are reported below but were not included in statistical significance testing.

The shaded gray bars in Figure 5 illustrate the subjects’ performance with a fixed decoder after short adaptation with either RML or SmoothBatch (SB) CLDA, averaged over multiple sessions. The corresponding individual session data is plotted in Figure 6. For all subjects, performance after RML was better than after SmoothBatch with respect to movement error (ME), movement variability (MV), reach time (RT), normalized path length (NPL), and % correct. Monkey C achieved 17% lower ME, 19% lower MV, 24% lower RT, 16% lower NPL, and 7% higher percent correct after RML adaptation than after SB CLDA. Performance differences for monkeys J and S were similar (ME: 13% and 18%, MV: 15% and 21%, RT: 15% and 12%, NPL: 10% and 13%, and % correct: 12% and 10%, respectively, for J and S). Statistically,
Figure 5: Performance comparison between the RML and SmoothBatch CLDA algorithms (averaged data). For all three subjects, both RML and SmoothBatch (SB) adaptation were performed starting from identical decoder seedings, across multiple sessions. Shaded gray bars depict performance (averaged across sessions) with a fixed decoder after a short adaptation period (5 minutes), starting from a shuffled KF seeding. White bars depict performance (averaged across sessions) with a fixed decoder after a more extended adaptation period (10+ minutes), starting from a VFB seeding. Error bars denote standard error of the mean, and asterisks indicate statistical significance (* $p < 0.05$, ** $p < 0.01$). Performance was quantified by evaluating four different measures of cursor trajectory quality on successful trials and the percentage of successful trials. Smaller values represent better performance for all metrics except % correct.
Figure 6: Performance comparison between the RML and SmoothBatch CLDA algorithms (individual session data). Individual session data used to plot the averaged data in Figure 5. The circle markers connected by each line represent the performance after a short period (5 minutes) of RML and SmoothBatch CLDA, starting from a common shuffled decoder seeding. A circle marker for one CLDA algorithm with no connected line indicates that the monkey did not perform successful trials following CLDA with the other algorithm. Performance is quantified by evaluating four different measures of cursor trajectory quality on successful trials and the percentage of successful trials. Smaller values represent better performance for all metrics except % correct.
these performance differences were significant \((p < 0.05, \text{ one-sided, paired Wilcoxon signed-rank test})\) for all subjects across all four performance metrics. Furthermore, RML adaptation proved to be more robust than SmoothBatch, as all three subjects were able to successfully perform trials more often after RML adaptation versus after SB adaptation. Out of a total of \(n = 12\) sessions, monkey C was unable or unwilling to successfully acquire targets in only 1 session after RML, but this was the case in 6 of the sessions for SB. Monkey J \((n = 16)\) and monkey S \((n = 15)\) performed the task in all sessions after RML adaptation, but each was unwilling or unable to successfully acquire targets after SB adaptation in 4 of their respective sessions.

We also evaluated the subjects’ performance after a longer period of adaptation (10 or more minutes; “extended adaptation”) with both algorithms (see the white bars in Figure 5). For this condition, decoders were seeded using the VFB (visual feedback) method. With more extended decoder adaptation and a potentially more favorable decoder seeding, most differences in performance were insignificant (only the normalized path length for monkey C was significantly lower after RML than after SB, \(p < 0.05\)). However, performance after RML adaptation was still better than after SB adaptation in most cases, except for the reach times for monkey C and movement variability for monkey S. Moreover, all three monkeys performed successful trials after RML adaptation in all sessions \((n = 7, 11, \text{ and } 9 \text{ for monkeys C, J, and S, respectively})\). However, both monkeys S and J were either unable or unwilling to perform trials in one of their respective sessions after SB adaptation.

To control for the possibility that the difference in half-life settings between both algorithms might be responsible for the observed performance gap after a short amount of adaptation, we conducted an additional set of experiments in which a time-varying half-life was also used for SmoothBatch adaptation. Although the half-life could not be increased instantaneously (at every KF time step) as with RML due to the batch-based nature of SmoothBatch, the half-life schedule for SmoothBatch was set such that the first decoder update occurred with a half-life of 30 s and the last decoder update occurred with a half-life of 300 s, to match the RML condition as closely as possible. Figure 7 illustrates the subjects’ average performance with a fixed decoder after short adaptation with either RML or SB, both with a time-varying half-life. For monkeys C and S, performance after RML was still significantly better \((p < 0.05, \text{ one-sided, paired Wilcoxon signed-rank test})\) than after SmoothBatch across four out of five performance metrics (only the % correct for monkey C and reach times for monkey S after RML were not significantly better than after SB). For monkey J, reach times and normalized path length were lower after RML than after SB (the difference in reach times was significant: \(p < 0.05\)), and movement error, normalized path length, and % correct were not significantly different \((p > 0.05)\). Overall, RML adaptation still proved to be more robust than SmoothBatch.
Figure 7: Control comparison between the RML and SmoothBatch CLDA algorithms (averaged data). Performance (averaged across sessions) with a fixed decoder after a short adaptation period (5 minutes), starting from a shuffled KF seeding. In contrast to the data plotted in Figure 5, where a time-varying half-life was used for RML only, a time-varying half-life was used here for both RML and SmoothBatch. Error bars denote standard error of the mean, and asterisks indicate statistical significance (*p < 0.05, **p < 0.01). Performance was quantified by evaluating four different measures of cursor trajectory quality on successful trials and the percentage of successful trials. Smaller values represent better performance for all metrics except % correct.
Notably, while all three subjects always performed successful trials after RML adaptation in every session, they did not perform successful trials after SB adaptation in some sessions (they were either unable or unwilling to do so). In particular, this occurred in 3 out of $n = 7$ sessions for monkey C, 1 out of $n = 8$ sessions for monkey J, and 3 out of $n = 12$ sessions for monkey S.

5 Discussion

By developing a continuously adaptive CLDA algorithm that updates parameters on every decoder time step during initial decoder training and any subsequent recalibrations, we have demonstrated the ability to achieve fast acquisition of BMI performance when starting from potentially adverse seeding conditions. Since we used an assistive control paradigm concurrently with CLDA, the subject appeared to have high performance from the start, making it difficult to measure how performance improved during the course of decoder adaptation. Therefore, we chose to measure performance by stopping CLDA (and assist) after a fixed amount of time and evaluating the subjects’ cursor control with a fixed decoder. When comparing our RML algorithm to a previously developed CLDA algorithm (SmoothBatch) that is not continuously adaptive, we found that three macaque monkey subjects using spiking activity or local field potentials achieved higher levels of performance after a relatively short of period of RML adaptation compared to SmoothBatch adaptation. When more extended adaptation was performed, both algorithms reached comparable levels of performance. One of the advantages of a continuously adaptive CLDA algorithm is the ability to instantaneously adjust adaptation rates (such as the half-life $h$), which we have leveraged in our experiments with RML. By starting with a relatively low half-life (30 s) and gradually increasing this value over the course of CLDA, we aim to make large initial parameter updates that bring the parameters into the right ballpark while smoothly transitioning toward more conservative, finely tuned adaptation as decoder parameters begin to converge. However, our control experiments testing SmoothBatch with an increasing half-life showed that a time-varying half-life by itself does not account for the performance gap after RML versus SmoothBatch CLDA, suggesting that continuous adaptation itself appears to be important for rapid acquisition of performance.

Previous work in brain-machine interfaces and related fields has investigated comparisons between batch-based and continuously adaptive algorithms for improving performance (Kim et al., 2006; Danziger, Fishbach, & Mussa-Ivaldi, 2009; Orsborn, Dangi, Moorman, & Carmena, 2011). In brain-machine interfaces, continuously adaptive algorithms have been developed and tested in simulation for different decoder types and neural signal sources (Wolpaw & McFarland, 2004; Eden, Frank, Barbieri, Solo, & Brown, 2004; Kim et al., 2006; Wang & Principe, 2008; Dangi, Gowda, Heliot,
Continuous Closed-Loop Decoder Adaptation

& Carmena, 2011; Dangi, So, Orsborn, Gastpar, & Carmena, 2013a), but not all algorithms fully translated to experimental settings when compared to batch-based methods. For example, in Orsborn et al. (2011), closed-loop experiments demonstrated that CLDA with a continuously adaptive method led to performance improvements, but those improvements occurred relatively slowly and were not fully maintained upon fixing the decoder. In the realm of human-machine interfaces, Danziger et al. (2009) conducted experiments in which high-dimensional signals from a wearable glove were transformed to control the joint angles of a simulated two-link robot arm. Two learning algorithms—Moore-Penrose (MP) pseudoinverse (batch) and LMS gradient descent (continuously adaptive)—were applied to adapt the glove-to-robot transformation based on errors measured in past performance. Although the LMS group outperformed a control group while the MP group did not, the LMS subjects failed to achieve better generalization than the control subjects. Generalization was also found to be a problem in Orsborn et al. (2011), where closer inspection of the adapted parameter trajectories revealed that with a continuously adaptive stochastic gradient method, some parameters did not appear to converge, perhaps due to the method’s tendency to overfit on short timescales. Indeed, in algorithms based on stochastic gradient methods, each individual adjustment to decoder parameters is based on only a single data point and therefore not necessarily guaranteed to be accurate. In contrast, the SmoothBatch algorithm (Orsborn et al., 2012) avoids this problem by making updates based on small (1–2 minutes) batches of data. However, it too is partially limited by the fact that its data batches cannot be too small, or else the corresponding batch parameter estimates used as part of the weighted averages in its update rules will become inaccurate. On the other hand, if the data batches are made to be larger, then the batch parameter estimates will likely become more accurate, but then the user must potentially wait for a long time for the next decoder update to occur. Since the RML algorithm updates its sufficient statistics at each time step before updating the KF parameters themselves, each RML update is effectively still based on a (weighted) batch of data. As a result, RML is able to perform continuous adaptation while still ensuring that each decoder update results in a potentially more accurate adjustment of parameters than would be achieved if each update was based on only a single data point. Importantly, the RML algorithm is computationally fast and therefore feasible as a continuously adaptive CLDA algorithm for a wide variety of neural signal sources, including spikes, LFP, electrocorticography (ECoG), and electroencephalography (EEG). By first updating a set of sufficient statistics that are then used to adapt the actual decoder parameters, the RML algorithm is expressed in terms of recursive update rules that are computationally simple and memory efficient. Moreover, when the number of neural features being passed into the decoder is large, RML’s update rules can be reformulated using the matrix inversion lemma to avoid costly matrix inversions.
in the KF equations that would otherwise make continuous adaptation infeasible.

Overall, our results with RML indicate that continuous adaptation is indeed a feasible and successful CLDA paradigm for rapid performance acquisition in BMIs. In our experiments, we used a Kalman filter decoder operating with 100 ms between time steps. However, alternative decoding architectures that operate on faster timescales have been proposed, such as the point-process filter (PPF) (Eden et al., 2004). Since the PPF models the occurrence of individual spikes, a PPF decoder would likely need to operate with less than 5 ms between time steps. While we have taken an important step toward continuous adaptation by using RML to update every 100 ms, it should be explored whether continuous adaptation on an even faster timescale within a PPF framework (Shanechi & Carmena, 2013; Shanechi, Orsborn, Gowda, & Carmena, 2013) could enable more rapid acquisition of performance. Furthermore, for these types of continuously adaptive algorithms, it is important to determine whether CLDA will synergize well with previous results that have demonstrated the importance of neural plasticity for BMI learning (Taylor et al., 2002; Carmena et al., 2003; Jarosiewicz et al., 2008; Ganguly & Carmena, 2009; Ganguly, Dimitrov, Wallis, & Carmena, 2011; Koralek et al., 2012). For instance, Ganguly and Carmena (2009) demonstrated that by fixing the parameters of the BMI decoder and keeping neurons stable, a neural map of the decoder forms that is stable across time can be readily recalled, and is robust to interference from a second learned map. Therefore, even if continuously adaptive CLDA algorithms are used to rapidly improve initial performance, subsequent practice with a fixed decoder could potentially still lead to the formation of a stable map in order to achieve long-term retention of neuroprosthetic skill.

Acknowledgments

This work was supported by the National Science Foundation Graduate Research Fellowship (H.G.M.), on 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

Continuous Closed-Loop Decoder Adaptation


**Reference List**


Continuous Closed-Loop Decoder Adaptation


Received January 14, 2014; accepted April 15, 2014.