Closed-Loop Decoder Adaptation Algorithms for Kalman Filters in Brain-Machine Interface Systems

Siddharth Dangi

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2011-139
http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-139.html

December 16, 2011

Acknowledgement

The derivation of the main algorithms presented in this report -- the Adaptive Kalman Filter and SmoothBatch and their variants -- was largely my own contribution. However, it would not have been possible without important and fruitful interactions with fellow graduate students, namely Amy Orsborn and Suraj Gowda.

The testing of the algorithms was joint work led by Amy. Amy conducted the analysis of the experimental data and created the figures used in this report. I am grateful to Amy and Helene Moorman for their tireless e orts in training the non-human primates, running the day-to-day BMI experiments, and allowing the inclusion of the resulting experimental data in this report.

Finally, I would like to thank my advisor, Jose Carmena, for his sound advice and helpful encouragement.

Closed-Loop Decoder Adaptation Algorithms for Kalman Filters in Brain-Machine Interface Systems

Siddharth Dangi

16 December 2011

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Jose Carmena, Research Advisor

Date

Professor Michel Maharbiz, Second Reader

Date

Author Contribution and Acknowledgements

The conception and derivation of the main algorithms presented in this report – the Adaptive Kalman Filter and SmoothBatch and their variants, including the alternate update rules and normalized step-sizes, was largely my own contribution. However, it would not have been possible without important and fruitful interactions with fellow graduate students in the Brain-Machine Interface Systems Lab, namely Amy Orsborn and Suraj Gowda.

The testing of the algorithms was joint work led by Amy Orsborn. Amy Orsborn conducted the analysis of the experimental data and created the figures used in this report. I am grateful to Amy Orsborn and Helene Moorman for their tireless efforts in training the non-human primates, running the day-to-day BMI experiments, and allowing the inclusion of the resulting experimental data in this report. Finally, I would like to thank my advisor, Jose Carmena, for his sound advice, helpful encouragement, and enormous passion for the BMI field, which is highly contagious and serves as a constant reminder of why BMI research is so important.

Part I

INTRODUCTION

1 Brain-Machine Interfaces

Brain-Machine Interfaces (BMIs) aim to help severely disabled patients suffering from neurological injuries and diseases by decoding neural activity into control signals for assistive devices. Early work has exhibited a compelling proof-of-concept for BMIs, with several groups showing demonstrations of rodents [1], non-human primates [2–12], and humans [13, 14] controlling artificial actuators using neural activity. Significant improvements in reliability (lifetime usability of the interface) and performance (achieving control and dexterity comparable to natural movements) are still needed to achieve clinical viability for humans [15, 16]. The potential benefits, however, are enormous. For instance, amputees could be outfitted with prosthetic arms – directly controlled by neural signals – enabling them to effortlessly perform everyday reaching and grasping movements that would otherwise be impossible.

At the heart of any Brain-Machine Interface (BMI) is a decoding algorithm, or "decoder", whose purpose is to translate recorded neural activity into control signals for a computer cursor, prosthetic device, or other external actuator. BMI decoders are often created offline by first recording neural activity while a subject performs movements [2–4, 8, 9], or imagines moving [7, 12–14], and then training a decoder to predict these movements from the neural activity. BMIs, however, are inherently closed-loop systems, since the BMI user typically receives performance feedback by visually observing the actuator's movements. Therefore, it is not surprising that open-loop decoder prediction power does not directly correlate with closed-loop performance [17, 18]. These results suggest that improvements in BMI performance cannot be achieved solely by finding an optimal open-loop decoding algorithm, thus highlighting the importance of fundamentally treating BMIs as closed-loop systems.

2 Closed-Loop Decoder Adaptation

Recent work in improving BMI performance has focused on the pivotal roles of both brain and machine (i.e., decoder) adaptation during closed-loop operation. For instance, Ganguly and Carmena [10] demonstrated that when subjects practiced BMI control with a fixed decoder, they learned a stable neural representation of the decoder over time, and the development of this representation paralleled performance improvements. In other words, the brain could adapt itself in order to improve performance. Other researchers have taken the opposite approach, by adapting the decoder during closed-loop operation to improve performance [3, 19–23]. Appropriately enough, we refer to this process as closed-loop decoder-adaptation.

Closed-Loop Decoder Adaptation (CLDA) is an emerging paradigm for achieving rapid performance improvements in online Brain-Machine Interface (BMI) operation. CLDA refers to the process of adapting or updating the decoder's parameters during closed-loop operation – i.e., while the subject is using the BMI. The main purpose of CLDA is to "improve" the decoder – to make it more accurately represent the true underlying mapping between the user's neural activity and their intended movements.

The design of a CLDA algorithm involves multiple crucial decisions, including the choices of:

- which decoder parameters to update (and which not to update),
- how to update them (the actual parameter update formulas),

- how *often* to update them (the "time-scale of adaptation"),
- what data to update them with (the "teacher signal").

In this report, we derive CLDA algorithms for a Kalman filter decoder that were designed as improvements upon the standard Batch method [24], and present experimental results using these algorithms in non-human primate experiments involving a center-out cursor control task.

Part II

METHODS

3 Electrophysiology

Two microwire arrays of 128 teflon-coated tungsten electrodes (35 μm diameter, 500 μm wire spacing, 8x16 array configuration; Innovative Neurophysiology, Durham, NC) were implanted (one in each brain hemisphere) in an adult male rhesus macaque (macaca mulatta). Both 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). Localization was performed using stereotactic coordinates from rhesus brain anatomy [25].

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. Unit activity was recorded using a combination of two 128-channel MAP and OmniPlex recording systems (Plexon Inc, Dallas, TX). Single and multi-unit activity was sorted using an online sorting application (Plexon, Inc, Dallas, TX), and only neural activity with well-identified waveforms were used for BMI control.

4 Behavioral Task

The subject sat in a primate chair with its head restrained, and observed the task display via a computer monitor projecting to a semi-transparent mirror parallel to the floor. Figure 1 shows an illustration of the task set-up and trial timeline. The subject was trained to perform a 2D center-out reaching task to 8 targets (1.7cm radius) uniformly spaced around a 14cm diameter circle. Trials were initiated by moving the cursor to the center target and holding for 400ms. (The task was self-paced – the subject had an no time limit to initiate a trial.) Upon entering the center, the reach target appeared. After the center-hold period ended, the subject was cued via target flash, and given a 3s time limit to move the cursor to the reach target. In order to receive a liquid reward, the subject was required to hold the cursor within the target boundary for 400ms. If the subject failed to hold at the center or target, or failed to reach the target within the time limit, the trial was classified as an error, and was aborted. Reach targets were presented in a block structure of 8 targets at a time, with pseudo-randomized order within each block.

The initial task-training was conducted with the subject making arm movements. The subject's arm was placed in a 2D KINARM exoskeleton (BKIN Technologies, Ontario, Canada), which constrained movements to the horizontal plane parallel to and just under the reach-target display. A cursor co-localized with the center of the subject's hand was displayed on the screen to provide visual feedback of hand location. During BMI operation, the subject performed the center-out task by moving the cursor strictly under neural control, and the subject's arm was removed from the KINARM and confined within the primate chair. The subject was proficient (over-trained) in the center-out task with arm movements before BMI experiments commenced.

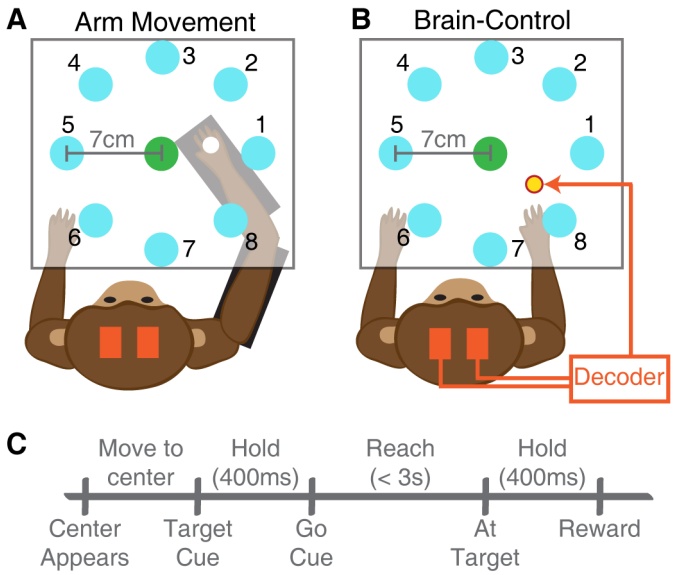

Figure 1: Center-out task schematic (A) and structure (B) [26].

Part III

ALGORITHMS AND EXPERIMENTAL RESULTS

5 Kalman Filter

5.1 Model and equations

The Kalman filter is a linear, recursive algorithm for producing estimates of an unknown state over time, and is a popular decoding algorithm choice for BMI applications [27]. The Kalman filter's estimates are based on both a prior distribution on how the state evolves, and periodic noisy observations of the state. In the BMI context, let x_t and y_t be vectors representing the kinematic state of the BMI actuator and the neural firing rates, respectively, at time t. The Kalman filter assumes the following models for how the state x_t evolves over time and how the observation y_t relates to the state:

$$\begin{aligned} x_{t+1} &= Ax_t + w_t \\ y_t &= Cx_t + q_t \end{aligned} \tag{1}$$

where $w_t \sim \mathcal{N}(0, W)$ and $q_t \sim \mathcal{N}(0, Q)$. The actual state estimation that is performed at each time step t is conventionally thought of as having two stages (note that the filter also tracks the covariance matrix P of the estimates over time):

1. **Predict stage**: the previous state estimate $\hat{x}_{t-1}$ is used to generate the *prior* estimate $\hat{x}_{t|t-1}$ (an estimate *before* observing y_t) of x_t:

$$\begin{aligned} \hat{x}_{t|t-1} &= A\hat{x}_{t-1} \\ P_{t|t-1} &= AP_{t-1}A^T \end{aligned}$$

2. **Update stage**: the *prior* estimate is "updated" using the observation y_t to form the *posterior* estimate $\hat{x}_t$ (an estimate *after* observing y_t) of x_t:

$$\begin{aligned} K_t &= P_{t|t-1}C^T\left(CP_{t|t-1}C^T + Q\right)^{-1} \\ \hat{x}_t &= \hat{x}_{t|t-1} + K_t\left(y_t - C\hat{x}_{t|t-1}\right) \\ P_t &= \left(I - K_tC\right)P_{t|t-1} \end{aligned}$$

The *posterior* estimate $\hat{x}_t$ is the output of the Kalman filter at time t.

We used a position-velocity Kalman filter, as in [22, 23] (operating in end-point coordinates):

$$x_t = \begin{bmatrix} p_{x,t} & p_{y,t} & v_{x,t} & v_{y,t} & 1 \end{bmatrix}^T$$

where $p_{x,t}$, $p_{y,t}$, $v_{x,t}$, $v_{y,t}$, represent the x and y coordinates of cursor position and velocity at time t. The constant 1 term accounts for non-zero mean observations y_t (e.g., the neurons' baseline firing-rates). Online BMI control was implemented using PlexNet (Plexon Inc, Dallas TX) to stream neural data on a local intranet from the Plexon recording system to a dedicated computer running Matlab (The Mathworks, Natick, MA). Neural firing rates were estimated with a 100ms bin width. Neural ensembles of 16-36 neurons were used. Units were selected only based on waveform quality.

5.2 Filter parameters and CLDA

The Kalman filter model is parametrized by four matrices: A, W, C, and Q. Note that since A and W describe the state evolution model, they effectively define a prior distribution on the time-series of states. Since these matrices represent our prior knowledge of how the state evolves, we initially set them using data collected from a manual control session, and then kept them fixed (i.e., they were not updated by our CLDA algorithms). For this reason, from this point onwards our discussion of CLDA algorithms for Kalman filter decoders will assume that A and W are fixed, and we will focus on update rules for only C and Q.

One important aspect of CLDA algorithms that was previously mentioned is the data, or "teacher signal", used to adapt the decoder. For instance, in a center-out task using a computer cursor BMI, consider the user's neural activity and the corresponding sequence of cursor movements produced by the decoder. If the decoder is not optimal (as is often the case), this sequence of movements may not correspond to the user's intended cursor movements, and so a CLDA algorithm might require an estimate of these intended cursor movements. Paired with the aforementioned neural activity, this (estimated) sequence of intended movements acts as a "teacher signal" that can be used to improve the decoder. As an example, CURSORGOAL is one method for estimating the user's intended cursor movements – it assumes that the user always intends to move the cursor in a straight line towards the current target [22, 23]. All of our experiments used CURSORGOAL as the teacher signal for our CLDA algorithms. In the derivations below, we use x_t to refer to the estimate of intended movements (which could be chosen to be the CURSORGOAL estimate, or any other estimate).

6 Batch

6.1 Method [22, 23]

Gilja et al.'s Batch approach of Kalman filter parameter estimation entails collecting data and processing the entire batch at once to update the decoder's parameters [24]. One standard batch estimate of the KF parameters C and Q is the maximum likelihood estimate:

$$
\begin{aligned}
C &= YX^T \left(XX^T\right)^{-1} \\
Q &= \frac{1}{N} \left(Y - CX\right) \left(Y - CX\right)^T
\end{aligned}
$$

where the Y and X matrices are formed by tiling recorded neural activity and intended kinematics, respectively.

The Batch CLDA algorithm typically involves having the subject use a decoder to perform closed-loop control, while cursor kinematcs and neural data are collected. Once enough data is collected (6-10min), the corresponding sequence of intended kinematics is estimated, and new decoder parameters are set using these estimated intended kinematics and the observed neural activity.

6.2 Performance [26]

The Batch method exhibited fast overall improvement (Figure 2), and stable task performance (Figure 2A) and reach kinematics (Figure 2B). Approximating a linear slope of improvement (using start and end performance rates calculated with a 100-trial moving average), performance improved at a rate of 1.34 ± 0.39 %/minute ($n = 5$). However, especially near the beginning of the session, the Batch method requires the subject to persist with a poorly performing decoder during the entire batch session (6-10 minutes), which can significantly reduce the subject's level of task engagement.

6.3 Discussion

In previous work [22,23] using the Batch CLDA algorithm, when initial closed-loop performance was already at moderate levels, only one Batch parameter update was required to significantly improve performance. However, our results indicate that, when starting with a low level of performance, multiple batch updates are required. One possible explanation for this phenomenon is that poor initial decoder performance can reduce subject engagement in the BMI task. Furthermore, in response to initial low performance, it is plausible that subjects may alter their control strategies more frequently, effectively adding undesired variance to the observed data and making it harder to accurately update the decoder's parameters.

In sum, the Batch CLDA algorithm achieves fast overall improvement and yields stable performance after adaptation is stopped, but its relatively low frequency of parameter updates can reduce the subject's level of engagement when starting with a low level of performance. The Adaptive Kalman Filter, which updates decoder parameters much more frequently, is designed to overcome this issue.

7 Adaptive Kalman Filter

7.1 Method [28]

The Adaptive Kalman Filter is a CLDA algorithm designed to update the decoder's parameters much more often than the Batch method. Specifically, the Adaptive KF updates the decoder's parameters at each Kalman filter iteration (i.e., every 100ms). The Adaptive KF's update rules are based on gradient descent, and we derive them below.

7.1.1 State Observation matrix C

To arrive at an update equation for C, we first write C as the solution to an optimization problem:

$$
C = \arg\min_C \underbrace{\mathbb{E}\left[\left\|y_t - Cx_t\right\|^2\right]}_{g(C)}
$$

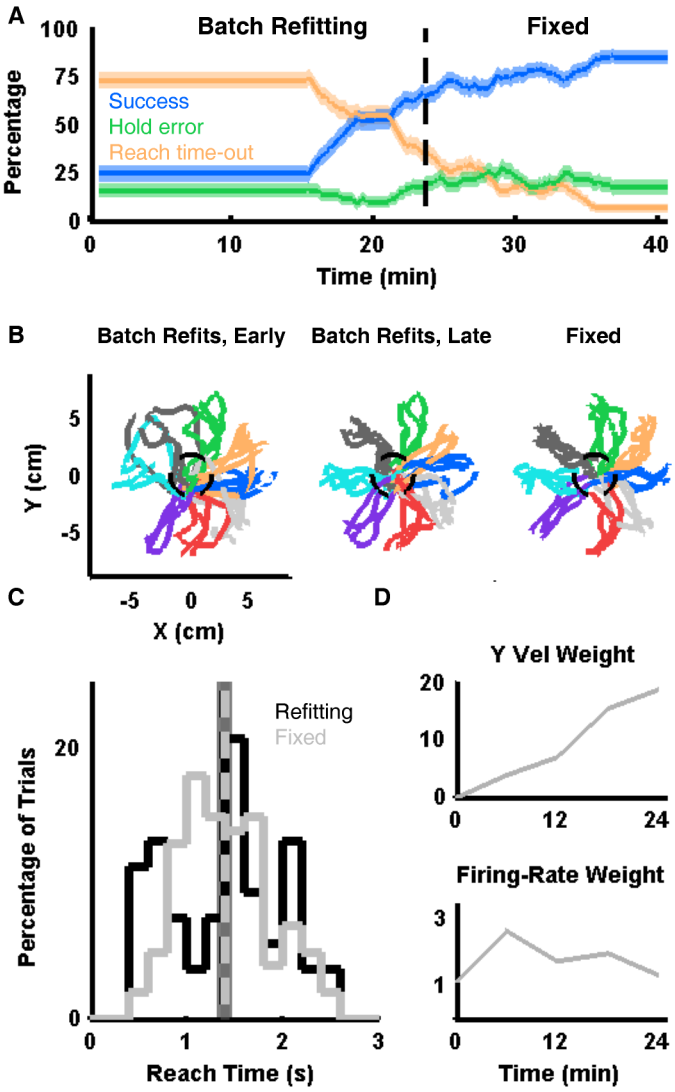

Figure 2: Performance of the Batch CLDA algorithm for one representative session in which 4 parameter updates were performed. (A) Task performance rates, quantified using a 100-trial moving average. Only successfully initiated trials were considered for analysis. (B) Successful reach trajectories (4 per target direction) at the beginning and end of decoder adaptation (left and center), and immediately after adaptation was stopped and decoder's parameters were held fixed (right). (C) Distribution of reach times for the first and last 100 trials during decoder adaptation, and the first 100 trials after fixing the decoder's parameters. (D) Progression of decoder weights (C matrix entries) for one representative neuron, for y-velocity (top) and baseline firing-rate states (bottom) [26].

where we have defined $g(C) \triangleq \mathbb{E}\left[\|y_t - Cx_t\|^2\right]$. Having done this, a natural way to update C is using stochastic gradient descent. We can calculate the true gradient of $g(C)$, and then obtain an approximation at each time step by removing the expectation operators:

$$\begin{aligned}\nabla g(C) &= 2\left(C\mathbb{E}\left[x_t x_t^T\right] - \mathbb{E}\left[y_t x_t^T\right]\right) \\ &\approx 2\left(Cx_t x_t^T - y_t x_t^T\right)\end{aligned}$$

Standard gradient descent would involve updates to C of the form

$$C^{(i+1)} = C^{(i)} - \mu \nabla g(C^{(i)})$$

but using our stochastic gradient method, we update C as

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

One drawback of this update equation is that it is sensitive to the scaling of the terms x_t and y_t, which makes it difficult to choose an appropriate step-size. One may recall that a similar issue arises when implementing a Least Mean Squares (LMS) filter. To motivate an analogous solution for our problem, let us recall the solution for the LMS problem.

The standard LMS filter assumes a linear relationship between the input x_n and the output y_n:

$$y_n = \theta^T x_n + \epsilon_n$$

where ϵ_n is an error term that is not endowed with a probability distribution. The basic LMS update equation for θ is given by

$$\theta^{(t+1)} = \theta^{(t)} + \mu(y_n - \theta^{(t)T}x_n)x_n$$

In a specific implementation known as "Normalized LMS" (NLMS), the step-size is so as to normalize by the powero fthe input:

$$\mu = \frac{1}{\|x_n\|^2}$$

One can verify that, in addition to possessing some favorable convergence properties, this choice of the step-size updates θ such that if on the next iteration there is a "repeated presentation" $(x_{n+1}, y_{n+1}) = (x_n, y_n)$, then no update will occur because the filter model will be exact.

Motivated by the NLMS algorithm's choice of step-size, we can find analogous an step-size for the update equation for C:

$$\mu = \frac{1}{\|x_t\|^2}$$

Indeed, one can verify that with this choice of step-size, if (x_{t+1}, x_t, y_{t+1}) is a repeated presentation of (x_t, x_{t-1}, y_t), then C will not be updated because that part of the Kalman filter model will be exact.

For our application, it is not necessary to update C to become exact for the data points at every iteration. Rather, we simply want to take a small step to "exactness", and thus we instead choose a step-size

$$\mu_C = \frac{\rho}{\|x_t\|^2 + \epsilon}$$

where $\rho \in [0, 1]$ is an adjustable parameter that we typically choose to be closer to 0, and the ϵ term is added to avoid divide-by-zero errors in implementation.

7.1.2 Noise covariance matrix Q

The noise covariance matrix Q cannot be similarly expressed as the solution of an optimization over an expectation. However, in order to arrive at an update equation for Q in the same fashion, we need to express Q as the solution of *some* optimization problem. One way to do this is to take the following statistical approach of parameter estimation.

Assume that we have i.i.d. samples $q_1, ..., q_N$ drawn from a distribution $\mathcal{N}(0, Q)$. The log probability of the observed data $\mathcal{D}$ given the parameter Q (also known as the log likelihood) can be written as

$$\begin{aligned} l(Q;\mathcal{D}) &= \log p(\mathcal{D}|Q) \\ &= \log \prod_{t=1}^{N} \frac{1}{(2\pi)^{m/2}\sqrt{\det Q}} \exp\left\{-\frac{1}{2} q_t^T Q^{-1} q_t\right\} \\ &= -\frac{mN}{2}\log 2\pi - \frac{N}{2}\log\det Q - \frac{1}{2}\sum_{t=1}^{N} q_t^T Q^{-1} q_t \end{aligned}$$

One way to estimate Q is to find the Maximum Likelihood (ML) estimate by solving the optimization problem

$$Q = \arg\max_{Q} l(Q;\mathcal{D})$$

To find an update equation for Q based on gradient ascent, we calculate the derivative of $l(Q;\mathcal{D})$ with respect to Q. Using knowledge of matrix derivatives, we find that

$$\frac{d}{dQ} l(Q;\mathcal{D}) = -\frac{N}{2}Q^{-1} + \frac{1}{2}Q^{-1}\left(\sum_{t=1}^{N} q_t q_t^T\right)Q^{-1} \tag{2}$$

Thus, one update equation for Q is (using only one data point w_t):

$$Q^{(i+1)} = Q^{(i)} + \mu\left(-\left(Q^{(i)}\right)^{-1} + \left(Q^{(i)}\right)^{-1} q_t q_t^T \left(Q^{(i)}\right)^{-1}\right) \tag{3}$$

We can also calculate the derivative of $l(Q;\mathcal{D})$ with respect to Q^{-1}, which turns out to be simpler:

$$\frac{d}{dQ^{-1}} l(Q;\mathcal{D}) = \frac{N}{2}Q - \frac{1}{2}\sum_{t=1}^{N} q_t q_t^T \tag{4}$$

This leads to another alternate update equation, in this case for Q^{-1} instead of Q (again, using only one data point q_t):

$$\left(Q^{-1}\right)^{(i+1)} = \left(Q^{-1}\right)^{(i)} + \mu\left(Q^{(i)} - q_t q_t^T\right)$$

Writing this more directly as an update for Q, we have

$$Q^{(i+1)} = \left(\left(Q^{(i)}\right)^{-1} + \mu\left(Q^{(i)} - q_t q_t^T\right)\right)^{-1} \tag{5}$$

Unfortunately, neither of the update equations ((3) or (5)) lend themselves to normalized step-sizes. Furthermore, they can both lead to numerical instability because both equations require matrix inversions. Therefore, in place of these equations, the Adaptive Kalman Filter uses a simpler method of updating Q:

$$Q^{(i+1)} = \alpha Q^{(i)} + (1-\alpha) q_t q_t^T$$

where $\alpha \in [0, 1]$ and is typically chosen to be closer to 1. This equation effectively implements an exponentially-weighted moving average, and as a result, it is not necessary to store q_{t-1},q_{t-2}, etc. as it would be for a standard moving average. While this update equation is "suboptimal" – in the sense that it does not represent gradient ascent on the log-likelihood – it is much simpler to implement. Note that in all the update equations above, q_t is simply shorthand for $y_t - C^{(i+1)}x_t$.

7.2 Performance [26]

The Adaptive KF's performance is shown in Figure 2. Performance reached a maximum level comparable to that of the Batch method, but unlike the Batch method, it showed non-trivial decline after adaption was stopped and the decoder was fixed. While the performance rates show a slight drop, movement quality is significantly reduced in the form of more variable trajectories and slower reaches (Figure 3B,C). The evolution of the decoder's parameters over time (Figure 3D) give insight into one possible reason for this drop in performance. Entries of the Kalman filiter's C matrix corresponding to the kinematic parts of the state (e.g. position and velocity) for neurons show noisy but clear trends over time. However, entries corresponding to the baseline firing rates show very high frequency oscillations. Since these baseline rates would be expected to remain relatively similar across a session, this suggests that the Adaptive KF may be overfitting to the data on short temporal time-scales, thus reducing the decoder's performance once fixed.

However, because the Adaptive KF updates decoder parameters much more frequently than the Batch method, the subject experienced improvements in performance more immediately, allowing for increased task engagement and subject motivation. In the first 10 minutes, the subject attempted to initiate almost twice as many trials in the Adaptive KF sessions than in the Batch sessions (Adaptive KF: 96.5 ± 4.9, $n = 2$; Batch: 56 ± 21, $n = 5$).

7.3 Discussion

The Adaptive KF improves performance (albeit more slowly than the Batch method), and its high frequency of decoder updates helps keep the subject continually engaged. However, it overfits decoder parameters on short temporal scales, leading to a decline of performance after CLDA is stopped. Estimating decoder parameters with batches of data avoids temporal overfitting and can reduce "noise" in decoder updates, but it is also desirable to update the decoder at a high frequency. A hybrid algorithm, SmoothBatch, combines these observed benefits of the Batch and Adaptive KF methods, yielding a more optimal approach to rapidly and reliably boost decoder performance.

8 SmoothBatch

8.1 Method [29]

While the Batch method executes parameter updates every 6-10min and the Adaptive KF executes its updates every KF iteration, the SmoothBatch CLDA algorithm updates the decoder on an intermediate (1-2 minute) time scale. The observed neural activity and intended kinematics are collected for a short (1-2 minutes) interval. Each batch of data is used to construct a new estimate of the C and Q matrices, $\hat{C}$ and $\hat{Q}$:

$$\begin{aligned} \hat{C} &= YX^T \left(XX^T\right)^{-1} \\ \hat{Q} &= \frac{1}{N}\left(Y - \hat{C}X\right)\left(Y - \hat{C}X\right)^T \end{aligned}$$

The BMI decoder is then updated using a weighted average between this new estimate and the previous parameter setting:

$$\begin{aligned} C^{(i)} &= \alpha C^{(i-1)} + (1-\alpha)\,\hat{C}^{(i-1)} \\ Q^{(i)} &= \beta Q^{(i-1)} + (1-\beta)\,\hat{Q}^{(i-1)} \end{aligned}$$

where i indexes discrete decoder iterations and $\alpha, \beta \in [0, 1]$ determines the speed of adaptation. Rather than set α and β directly, we re-parametrize them and instead set the half-life of the update

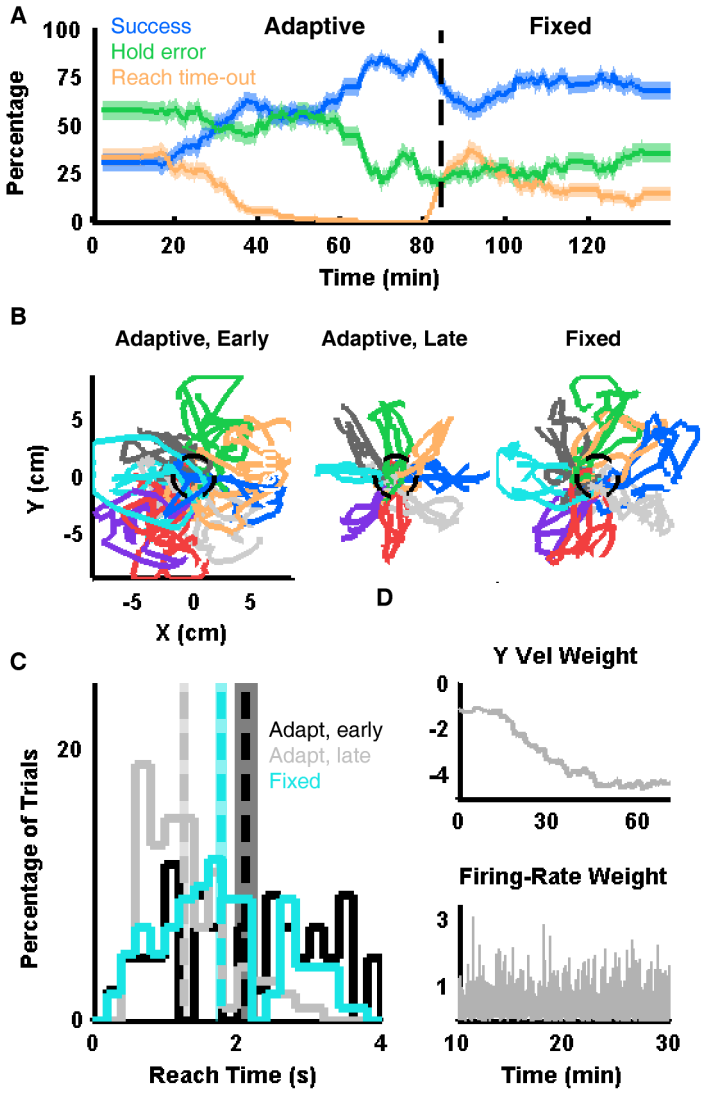

Figure 3: Performance of the Adaptive KF CLDA algorithm during one representative session. Format as in Figure 2 [26].

process – the time it takes for the presence of a $\hat{C}$ estimate in the current parameter settings to be reduced by a factor of one-half. The half-lives for C and Q – denoted by h_c and h_q – are related to α and β as follows:

$$\begin{aligned} \alpha^{h_c/b} &= \frac{1}{2} \\ \beta^{h_q/b} &= \frac{1}{2} \end{aligned}$$

where b is the length of each batch of data in time (referred to as the "batch size"). To avoid conducting a vast parameter search of both the batch size and half-life experimentally, SmoothBatch was first implemented in a BMI simulator that utilizes an empirically verified model of the user's learning process during closed-loop BMI operation [30]. Preliminary results from the simulator were used to narrow the search space, which was then explored experimentally. Rough optimization showed that batch sizes of 60 - 100s and half-lives of 90 - 210s produced the most rapid performance improvements. All presented data use parameters within this range, and the majority ($n = 46$) use an 80s batch size and 120s half-life. The C and Q half-lives were set to be equal, so that these matrices were always updated simultaneously.

8.2 Performance [29]

SmoothBatch CLDA rapidly improved closed-loop BMI performance regardless of the method in which the decoder's parameters were seeded. Figure 4 shows the evolution of performance for a representative session. As seen in Figure 4A, the subject was not readily able to perform the task with the initial decoder seeding (very few trials were initiated). After a few minutes (representing 1-2 decoder updates), the subject was able to initiate trials, but still performed the task with limited control, as evidenced by both a high rate of reach time-out events (Figure 4A) and irregular reach trajectories (Figure 4C). However, performance improved gradually, and typically after about 10 minutes of SmoothBatch parameter adaptation, both success percentage and success rate showed significant improvement.

8.3 Discussion and Future Work

By operating on an intermediate time-scale (1-2 min), SmoothBatch was able to combine the advantages of the Batch and Adaptive KF algorithms. SmoothBatch improves upon the Adaptive KF because avoids overfitting decoder parameters on short temporal scales, which allows performance to remain constant after adaption is stopped. However, it also improves upon the Batch method because it updates decoder parameters more frequently, which helps keep the subject motivated to perform the task.

Future work on closed-loop decoder adaptation will involve performing more detailed analysis of the various CLDA algorithms presented in this report, in order to inform experimental decisions. Specifically, future work on SmoothBatch and the other algorithms presented in this report will investigate whether, under certain model assumptions, any guarantees can be made of convergence of decoder parameters to their optimal values. Furthermore, it will be important and practical to determine whether and how this convergence depends on the algorithms' parameter settings of step-sizes, half-lives, and batch sizes.

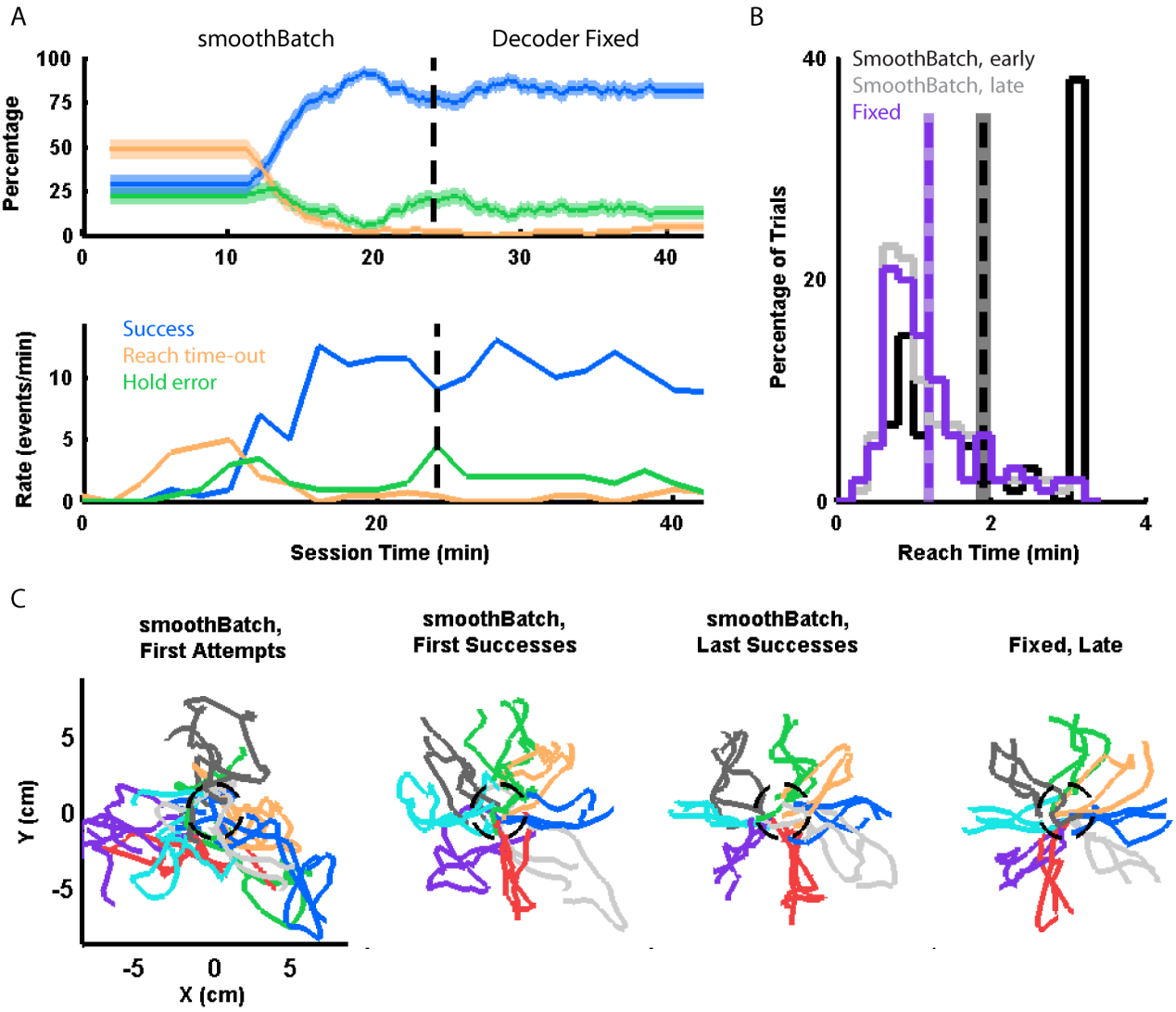

Figure 4: Performance of the SmoothBatch CLDA algorithm [29].

References

[1] John K. Chapin, Karen A. Moxon, Ronald S. Markowitz, and Miguel A. L. Nicolelis. Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex. *Nat Neurosci*, 2(7):664–670, July 1999.

[2] Mijail D Serruya, Nicholas G Hatsopoulos, Liam Paninski, Matthew R Fellows, and John P Donoghue. Instant neural control of a movement signal. *Nature*, 416(6877):141–142, March 2002. PMID: 11894084.

[3] Dawn M. Taylor, Stephen I. Helms Tillery, and Andrew B. Schwartz. Direct cortical control of 3D neuroprosthetic devices. *Science*, 296(5574):1829 –1832, June 2002.

[4] Jose M Carmena, Mikhail A Lebedev, Roy E Crist, Joseph E O'Doherty, David M Santucci, Dragan F Dimitrov, Parag G Patil, Craig S Henriquez, and Miguel A. L Nicolelis. Learning to control a Brain–Machine interface for reaching and grasping by primates. *PLoS Biol*, 1(2):e42, October 2003.

[5] Beata Jarosiewicz, Steven M Chase, George W Fraser, Meel Velliste, Robert E Kass, and Andrew B Schwartz. 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, December 2008. PMID: 19047633.

[6] Chet T. Moritz, Steve I. Perlmutter, and Eberhard E. Fetz. Direct control of paralysed muscles by cortical neurons. *Nature*, 456(7222):639–642, December 2008.

[7] Meel Velliste, Sagi Perel, M. Chance Spalding, Andrew S. Whitford, and Andrew B. Schwartz. Cortical control of a prosthetic arm for self-feeding. *Nature*, 453(7198):1098–1101, June 2008.

[8] Gopal Santhanam, Stephen I. Ryu, Byron M. Yu, Afsheen Afshar, and Krishna V. Shenoy. A high-performance brain–computer interface. *Nature*, 442(7099):195–198, July 2006.

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

[10] Karunesh Ganguly and Jose M. Carmena. Emergence of a stable cortical map for neuroprosthetic control. *PLoS Biol*, 7(7):e1000153, July 2009.

[11] Joseph E. O'Doherty, Mikhail A. Lebedev, Timothy L. Hanson, Nathan A. Fitzsimmons, and Miguel A. L. Nicolelis. A Brain-Machine interface instructed by direct intracortical microstimulation. *Frontiers in Integrative Neuroscience*, 3, September 2009. PMID: 19750199 PMCID: 2741294.

[12] Aaron J. Suminski, Dennis C. Tkach, Andrew H. Fagg, and Nicholas G. Hatsopoulos. Incorporating feedback from multiple sensory modalities enhances Brain–Machine interface control. *The Journal of Neuroscience*, 30(50):16777 –16787, December 2010.

[13] Leigh R. Hochberg, Mijail D. Serruya, Gerhard M. Friehs, Jon A. Mukand, Maryam Saleh, Abraham H. Caplan, Almut Branner, David Chen, Richard D. Penn, and John P. Donoghue. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442(7099):164–171, July 2006.

[14] Sung-Phil Kim, John D Simeral, Leigh R Hochberg, John P Donoghue, and Michael J Black. Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia. *Journal of Neural Engineering*, 5(4):455–476, December 2008.

[15] Vikash Gilja, Cindy A Chestek, Ilka Diester, Jaimie M Henderson, Karl Deisseroth, and Krishna V Shenoy. Challenges and opportunities for next-generation intracortically based neural prostheses. *IEEE Transactions on Bio-Medical Engineering*, 58(7):1891–1899, July 2011. PMID: 21257365.

[16] Jose del R. Milan and Jose Carmena. Invasive or noninvasive: Understanding Brain-Machine interface technology [Conversations in BME. *IEEE Engineering in Medicine and Biology Magazine*, 29(1):16–22, January 2010.

[17] Karunesh Ganguly and Jose M Carmena. Neural correlates of skill acquisition with a cortical brain-machine interface. *Journal of Motor Behavior*, 42(6):355–360, November 2010. PMID: 21184353.

[18] Shinsuke Koyama, Steven M Chase, Andrew S Whitford, Meel Velliste, Andrew B Schwartz, and Robert E Kass. Comparison of brain-computer interface decoding algorithms in open-loop and closed-loop control. *Journal of Computational Neuroscience*, 29(1-2):73–87, August 2010. PMID: 19904595.

[19] Lavi Shpigelman, Hagai Lalazar, and Eilon Vaadia. Kernel-ARMA for hand tracking and Brain-Machine interfacing during 3D motor control.

[20] Gregory J Gage, Kip A Ludwig, Kevin J Otto, Edward L Ionides, and Daryl R Kipke. Naïve coadaptive cortical control. 2005.

[21] Babak Mahmoudi and Justin C. Sanchez. A symbiotic Brain-Machine interface through Value-Based decision making. *PLoS ONE*, 6(3):e14760, March 2011.

[22] V Gilja, P Nuyujukian, C.A. Chestek, J.P. Cunningham, B.M. Yu, S.I. Ryu, and K.V. Shenoy. High-performance continuous neural cursor control enabled by feedback control perspective. Computational and Systems Neuroscience (COSYNE 2010).

[23] V Gilja et al. A high-performance continuous cortically-controlled prosthesis enabled by feedback control design. *2010 Neuroscience Meeting Planner. San Diego, CA*, 2010.

[24] Vikash Gilja. *Towards Clinically Viable Neural Prosthetic Systems.* PhD thesis, Stanford University, 2010.

[25] George Paxinos, Xu-Feng Huang, and Arthur W. Toga. *The Rhesus Monkey Brain in Stereotaxic Coordinates.* Academic Press, 1st edition, November 1999.

[26] A. L Orsborn, S. Dangi, H. G. Moorman, and J.M. Carmena. Exploring time-scales of closed-loop decoder adaptation in brain-machine interfaces. In *IEEE EMBS Conference*, 2011.

[27] Wei Wu, Yun Gao, Elie Bienenstock, John P Donoghue, and Michael J Black. Bayesian population decoding of motor cortical activity using a kalman filter. *Neural Computation*, 18(1):80–118, January 2006. PMID: 16354382.

[28] S. Dangi, S. Gowda, R. Heliot, and J.M. Carmena. Adaptive kalman filtering for closed-loop brain-machine interface systems. In *5th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 609–612, May 2011.

[29] A. L Orsborn, S. Dangi, H. G. Moorman, and J.M. Carmena. Closed-loop decoder adaptation on intermediate time-scales facilitates rapid bmi performance improvements independent of decoder initialization conditions. *Submitted to Transactions on Neural Systems and Rehabilitation Engineering*, 2011.

[30] Rodolphe Héliot, Karunesh Ganguly, Jessica Jimenez, and Jose M Carmena. Learning in closed-loop brain-machine interfaces: modeling and experimental validation. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics: A Publication of the IEEE Systems, Man, and Cybernetics Society*, 40(5):1387–1397, October 2010. PMID: 20007050.