UNIVERSITY OF CALIFORNIA

Los Angeles

Simulation of brain-machine interfaces

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical and Computer Engineering

by

Ken-Fu Liang

2022

ABSTRACT OF THE DISSERTATION

Simulation of brain-machine interfaces

by

Ken-Fu Liang
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2022
Professor Jonathan C. Kao, Chair

Intracortical brain-computer interfaces (BCIs) are expensive and time-consuming to design because accurate evaluation traditionally requires real-time experiments. In a BCI system, a user interacts with an imperfect decoder and continuously changes motor commands in response to unexpected decoded movements. Decoder optimization using previously collected "offline" data will therefore not capture this closed-loop response. However, this "closed-loop" nature of BCI leads to emergent interactions between the user and decoder that are challenging to model. The gold standard for BCI evaluation is therefore real-time experiments, which significantly limits the speed and community of BCI research. My dissertation is focused on creating a pure software BCI simulator which is accurate and versatile in evaluating and characterizing performance of never-before-tested decoders. In our approach, there are three components: the decoder, neural activity, and user control policy. Our design approach uses (1) published BCI experiments as ground truth decoder comparisons, collected with empirical neural recordings and a user-in-the-loop, (2) physical BCI emulator experiments allows us to optimize the neural encoder in isolation to reproduce published ground truth decoder performance, and finally (3) software BCI simulator experiments to implement approximate control policies for each decoder, replacing the user-in-the-loop. We demonstrate that our simulator is accurate and versatile, reproducing the published results of three distinct types of BCI decoders: (1) a state-of-the-art linear decoder (FIT-KF), (2) a "two-stage" BCI decoder requiring closed-loop decoder adaptation (ReFIT-KF), and (3) a nonlinear recurrent neural network decoder (FORCE). We anticipate this simulator will help democratize and significantly accelerate BCI research.

This dissertation presents steps toward a pure software BCI simulator. In the following, we will introduce BCI experiments, neural encoder, BCI emulator, and BCI simulator. Much of the work presented in these chapters has been published in peer reviewed journals, conferences, or in the process of peer review. We briefly present the main content of each chapter and point the reader to the relevant publications below:

1. Chapter 1 gives an overview of the field of brain-computer interface, including the importance of closed-loop experiments. We will also introduce prior BCI emulators, simulators, and our approach.

2. Chapter 2 introduces BCI monkey experiments, including experiment setup, tasks, and BCI decoders, which we aim to reproduce. We also shows the important metrics in BCI experiments which we used to validate our BCI emulator and simulator.

3. Chapter 3 presents various neural encoders which were trained to reproduce neural activity in single neuron PSTHs, and population neuron dynamics and rotations. We will also introduce important data preprocessing and regularization method to improve generalization of neural encoder.

4. Chapter 4 resents BCI emulator that first accurately predicts the detailed performance of a variety of decoders, including those that require multiple stages of training, as well as linear and nonlinear decoders.

5. Chapter 5 presents BCI simulator which reproduced decoder performance in a pure

software. We will introduce a new constrained RL algorithm and compare with conventional LQR and naive RL algorithm in controlling BCI systems.

The dissertation concludes with Chapter 6, providing a holistic view of this work. Together, we anticipate this simulator will help democratize and significantly accelerate BCI research.

The dissertation of Ken-Fu Liang is approved.

University of California, Los Angeles

2022

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank Professor Jonathan C. Kao. Kao has been guiding me to explore brain-computer interfaces. Kao's support, kindness, and friendship over the last five years have been invaluable to me especially passing through pandemic. I would like to thank my colleagues and friends in the Neural Computation and Engineering Lab who have not only helped me learn neuroscience and BCIs, but have made my PhD life in LA enjoyable. I've also had very helpful courses including but not limited from Professors Suhas Diggavi, Tao Gao, Wentai Liu, Ankur Mehta, and Paulo Tabuada. I'd also like to thank my thesis committee: Professors Tao Gao, Wentai Liu, and Dejan Markovic. I'd like to thank friends in the area who have supported me during my PhD. Finally, I'd like to thank my family: my parents, godparents, as well as my bothers and sisters-in-law. I am so grateful for your prayers, love and support.

VITA

2012-2013	M.S. , National Taiwan University (NTU)
2008-2012	B.S. , National Chiao Tung University (NCTU)

PUBLICATIONS

K. F. Liang, J. C. Kao, "A closed-loop emulator that accurately predicts brain-machine interface decoder performance", COSYNE, 2022

S. Olsen, J. Zhang, **K. F. Liang**, M. Lam, U. Riaz, J. C. Kao, "An artificial intelligence that increases simulated brain–computer interface performance", IEEE Journal of Neural Eng., 2021

K. F. Liang, J. C. Kao, "Deep Learning Neural Encoder for Motor Cortex", IEEE Trans. Bio-Med. Eng., 2019

CHAPTER 1

Introduction

The goal of brain-computer interfaces is to restore lost motor function to people with motor neurological disease and injury, including paralysis and amyotrophic lateral sclerosis (ALS). This is achieved by recording neural signals from the brain and then decoding these neural signals into control signals that guide a prosthetic or communication device, such as a robotic arm or computer cursor. In this fashion, the neural prosthesis restores motor function by bypassing the area of injury or disease, creating a new link for the brain to communicate with the world.

Brain-computer interface requires animal or human closed-loop experiments to develop and optimize decoders, which significantly limits the community and speed of BCI research. To overcome this limitation, one approach is to synthesize neural activity to more rapidly perform experiments that incorporate the closed-loop nature of BMI systems. The prior study by Cunningham et al. [2] developed a BCI "emulator" where a human subject's hand movements generate synthetic neural activity to control a BCI decoder, removing invasive neurosurgery but still including true control policy from testing subjects. An important component in a BCI emulator is the neural encoder, which maps motor commands (recorded hand kinematics) to synthetic neural activity. However, the neural encoder, Poisson process velocity tuning (PPVT) model, used by the prior study fails to capture the rich heterogeneity and dynamics of motor cortical activity [3]. Another approach, as oppose to synthesize neural activity then decode, Willett et al. [1] designed a simulator which approximated BCI decoded error that enabled hyperparameter tuning of a VKF. This simulator, however, does not have a neural data model and requires closed-loop experiments to model a decoder's dynamics.

Our goal is to have versatile software simulator without the need of any animal experiment, we therefore follow Cunningham's approach and further improve neural encoder with a nonlinear neural network. We also replace user-in-the-loop in the emulator with AI agents to achieve a software BCI simulator. Throughout this dissertation, we will investigate BCI data from monkey experiments and present various neural encoders. Also, we will demonstrate that, by modeling the kinematic-to-neural relationship as closely as possible, we were able to accurately recapitulate detailed timing and kinematic performance of representative neural decoders observed in monkey experiments.

1.1 BCI overview

BCI systems are comprised of three major components: (1) neural activity, typically recorded from motor cortical regions of the brain; (2) a decoder, which translates the neural recordings into control signals; and (3) a prosthesis, such as a computer cursor on a screen or a robotic arm, controlled by the decoder. Intracortical neural signals are measured from electrodes that reside in the outer few millimeters of motor cortical regions of the brain. These electrodes record action potentials, also known as "spikes," are the fundamental currency of information in the brain. Intracortical brain-computer interfaces (BCIs) were initially demonstrated in the late 1990 and early 2000s [4, 5, 6], entering pilot clinical trials in 2004 [7].

The performance and robustness of BCI systems depend greatly on the decoder algorithm (or 'decoder'), which converts neural activity from motor cortex into the kinematics of a prosthetic device. The past two decades have seen significant advances to increase BCI performance, incorporating ideas from control feedback [8, 9, 10], dynamical systems [11, 12], closed-loop decoder and neural adaptation [13, 14, 15], and deep learning [16, 17].

1.2 BCI decoder online evaluation

The decoder algorithm, which translates recorded neural population activity into prosthesis kinematics, is essential for high-performance BCI systems. Historically, decoder design has been inspired by neuroscientific views of motor cortex as well as by linear estimation, statistical inference, and neural network theory. BCI decoder algorithms are trained in a supervised fashion with simultaneous observations of real arm or prosthesis kinematics [18] and neural population activity. For example, a subject with motor neurological disease or injury may be asked to imagine mimicking the movements of an automated computer cursor while neural activity is recorded. A regression could then be performed to learn a mapping from the subject's recorded neural population activity to the kinematics of the automated computer cursor. Then, during real-time BCI control, also called "online" or "closed-loop" control, the computer cursor would be causally controlled by the decoder, which uses the subject's real-time neural population activity to predict the prosthesis kinematics.

Closed-loop BCIs pose an additional challenge that other applications in information systems engineering do not routinely face. Consider training a supervised algorithm that infers a variable x from an observed variable y. To do so, one must learn a mapping $f(\cdot)$ so that $\hat{x} = f(y)$, where $\hat{x}$ is the estimate of x. A common approach is to learn f from observations of (x, y) ("training data") such that a desired error metric, $(x, \hat{x})$, is minimized when evaluated on data not in the training set ("testing" or "cross-validation" data). This process is called *offline evaluation*, which producing insight into what algorithms are most promising to evaluate in available closed-loop experiments. However, in BCI settings, this approach can lead to suboptimal decoders [2, 19, 20]. One reason for this is because the subject controlling the BCI system continuously observes the movements of the prosthesis and can make online corrections to compensate for inaccurately decoded kinematics. Thus, it is typically the case that BCIs running in closed-loop operate on data distributions that differ substantially from the data distributions of the training set [21]. As a result of this,

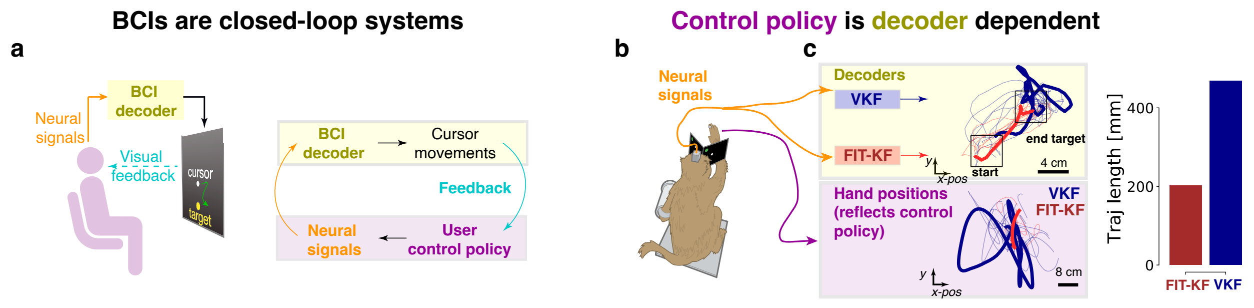
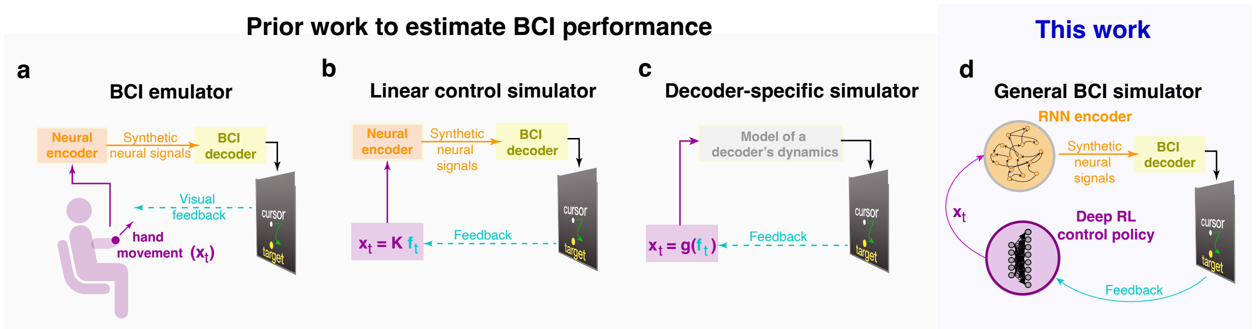


Figure 1.1: BCIs are closed-loop systems and control policy is decoder dependent. **(a)** BCI decoders are imperfect, so decoded cursor movements will not fully match the intent of the user. In response, the user will generate updated motor commands and neural signals. **(b)** BCI performance relies on a decoder dependent control policy. Hand and cursor positions were recorded while the monkey neurally controlled a VKF and FIT-KF. **(c)** In the yellow box, we plot the decoded output of the VKF (blue) and FIT-KF (red) from intracortical BCI experiments where the monkey sought to acquire a target at 45° (example trial bolded). In the purple box, we plot the monkey's recorded hand positions, which reflect his motor commands to control the decoder. Hand trajectories when controlling the VKF range all over the workspace and have longer trajectories with erratic movements compared to FIT-KF.

it is difficult to evaluate the performance of a putative decoder without running closed-loop experiments. This poses a challenge for decoder design, and we will address this with our BCI simulator.

1.3 Towards pure software BCI simulator

Although we have observed significant advances in decoder performance, BCIs remain in pilot clinical trials today, reflecting the relatively slow pace of BCI research where new innovations typically require months to years of experimental validation. Moreover, due to the unique considerations in BCI design, intracortical BCI research today can only be carried out by a handful of labs, further limiting the speed and community of BCI research. A key reason BCI development takes months to years is that BCIs are fundamentally "closed-loop" systems (Figure 1.1a). The performance of a BCI relies on the user's "control policy" used to interact with an imperfect BCI decoder. The user must constantly changes his or her



Purely software:	✗	✓	✓	✓
Accurate control policy:	✓	✗	✓	✓
Accurate neural signal:	✗	✗	No neural model	✓
Generalization:	✓	✓	✗	✓

Figure 1.2: Prior works without neurosurgery to estimate intracortical BCI performance and our new framework. **(a)** The BCI emulator circumvents invasive neurosurgery by generating synthetic neural signals, although it still requires human experiments to implement an accurate control policy. Though it can in theory generalize to new decoders, it does not because of its poor neural encoder. **(b)** The linear control simulator replaces the human with a linear control policy. This makes the simulator entirely software, but the control policy is not accurate. (c) The simulator by Willett et al. [1] enables hyperparameter optimization of a single decoder by modeling its dynamics (gray box), but cannot generalize to new decoders. **(d)** Our goal is the general solution, a purely software simulator that accurately predicts the performance of any type of decoder. **(a-d)** Green checkmarks indicate yes, red checkmarks indicate no, and yellow checkmarks indicate that while the answer is theoretically yes, in practice it is more nuanced and depends on a particular model.

motor commands in response to feedback of the decoded output. This constant updating results in emergent interactions and unique control policies for each decoder, meaning decoder performance depends on these interactions. We emphasize that these control policies are also *decoder dependent*. For example, Figure 1.1b illustrates BCI experiments where monkeys were allowed to move their hands to control a velocity Kalman Filter (VKF) [22] and a Feedback Intention Trained Kalman Filter (FIT-KF) [9]. The monkey's hand kinematics are desired because they reflect the motor commands used to control the decoder. The hand kinematics are markedly different for controlling the VKF versus the FIT-KF, with VKF requiring longer hand trajectory and erratic movements compared to FIT-KF (Figure 1.1c, FIT: 202 mm, VKF: 468 mm, $p < 10^{-7}$, Wilcoxon rank-sum test). Correctly predicting BCI performance therefore requires modeling how the user will uniquely interact with a particular decoder in closed-loop experiments. Consistent with this, several studies document that analysis carried out in "offline" simulations that do not incorporate closed-loop feedback can be discordant with closed-loop experiments [2, 19, 20].

To accelerate BCI research, prior studies have attempted to emulate or simulate BCI systems. Cunningham et al. [2] developed a BCI "emulator" where a human subject's hand movements generate synthetic neural activity to control a BCI decoder, removing invasive neurosurgery but still requiring experiments with a human-in-the-loop (Figure 1.2a). We use the term "emulator" to highlight that this system requires a physical experiment with hardware that mimics the BCI decoding system and a human to provide a control policy. While this approach is useful, and we later use it to validate a synthetic neural activity model (neural encoder), its use of human experiments significantly limits its community use and speed. To remove the human-in-the-loop and avoid experiments, Lagang and Srinivasan [23] used the linear quadratic regulator from control theory (Figure 1.2b) to approximate the human's BCI control policy. Other studies have also used linear policies from control theory in BCI design [24, 25, 26, 27, 28]. However, as shown by Willett and colleagues [29, 1], and by further experiments in this study, linear control policies are a poor approximation of user control policy and result in incorrect conclusions. Finally, Willett et al. [1] designed a simulator that enabled hyperparameter tuning of a VKF (Figure 1.2c). This simulator, however, does not have a neural data model and requires closed-loop experiments to model a decoder's dynamics. This simulator can therefore only optimize decoders already tested in closed-loop intracortical experiments and does not generalize to new, never-before-tested, decoders. It is also limited to optimize decoders of a linear form.

In contrast to prior work, our BCI simulator aims to faithfully model all aspects of BCI control to accurately estimate the performance of any BCI decoder entirely in software. This requires accurately modeling the unique user-decoder interactions without human or monkey experiments. It also requires generating synthetic neural activity that provides sufficient information for downstream neural decoders (Figure 1.2d). Our simulator solves the general

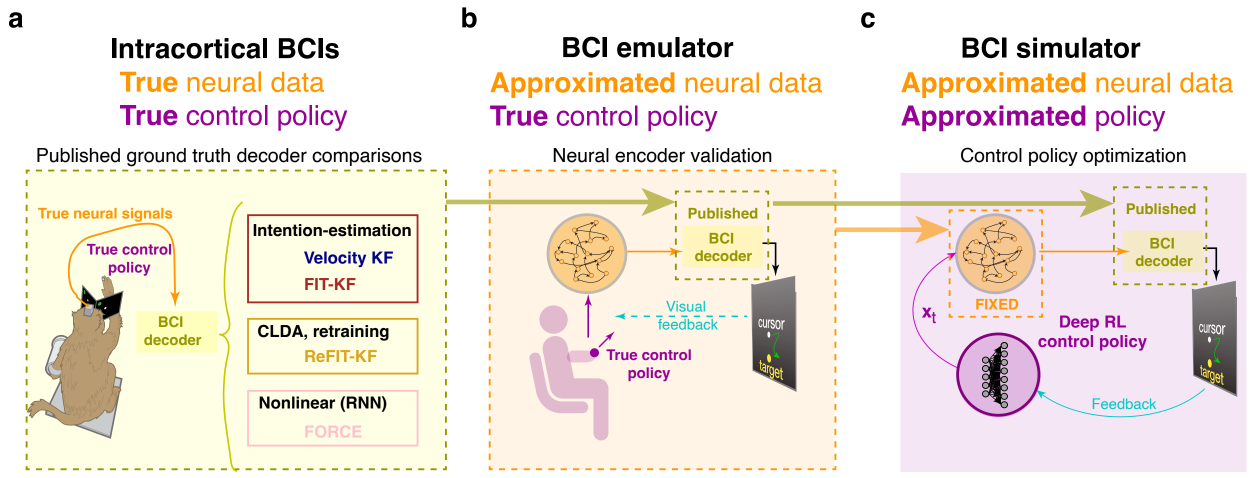


Figure 1.3: BCI simulator design approach. Our approach makes use of **(a)** published intracortical BCI literature, **(b)** BCI emulation, and **(c)** deep RL to design and validate the components of the BCI simulator. **(a)** Published BCI experiments, using true neural data and control policy, provide three different ground truth decoder comparisons to validate the BCI emulator and simulator. **(b)** The BCI emulator uses a human to implement a true control policy, enabling us to optimize the neural encoder to reproduce published BCI experiments. **(c)** We fix the neural encoder from the BCI emulator and then optimize the deep RL control policy in the simulator.

problem: it accurately evaluates the performance of representative decoder algorithms across diverse innovations and distinct study settings, entirely in software. Our idea fundamentally involves training an artificial intelligence (AI) agent that learns to control new decoders through deep reinforcement learning (RL).

There are three components in a BCI: the decoder, neural activity, and user control policy. A general BCI simulator must accurately approximate the neural activity and control policy to correctly predict decoder performance. Our design approach uses (1) published BCI experiments, with *true* (not approximated) neural activity and *true* user control policy, (2) physical BCI emulator experiments, with *approximated* neural activity but *true* control policy, and finally (3) software BCI simulator experiments, with *approximated* neural activity and *approximated* control policy. Published BCI experiments provide ground truth decoder comparisons, collected with empirical neural recordings and a user-in-the-loop, that we use to validate our BCI emulator and simulator (Figure 1.3a). The BCI emulator, which approximates neural activity but incorporates a user-in-the-loop, allows us to optimize the neural encoder in isolation to reproduce published ground truth decoder comparisons (Figure 1.3b). Finally, we build the BCI simulator by fixing a neural encoder from the BCI emulator and training deep RL agents to implement approximate control policies for each decoder, replacing the user-in-the-loop (Figure 1.3c). Although building the simulator requires both intracortical BCI data and BCI emulator experiments for a one-time build of a neural encoder, the complete BCI simulator is implemented entirely in software and accessible to all.

1.3.1 Monkey BCI experiments as ground truth decoder comparisons

Monkey BCI experiments require true neural data and true control policy. We chose to replicate three distinct BCI studies that improved BCI decoders through different innovations. This was to demonstrate that our simulator, which did not incorporate any decoder-specific information, could make general and correct predictions of BCI decoder performance across diverse innovations and distinct study settings. We chose to replicate the BCI experiments that established the (1) FIT-KF [9], (2) ReFIT-KF [8], and (3) FORCE [16] decoders. The FIT-KF is a linear state-of-the-art decoder whose high performance is due to the incorporation of control theory inspired dataset augmentation (also known as intention estimation) [9]. The ReFIT-KF, which also incorporates intention estimation, is a decoder that employs two stages of BCI training, a form of closed-loop decoder adaptation (CLDA) [30, 14]. The ReFIT-KF decoder first requires the user to control a position velocity Kalman filter (PVKF), followed by a retraining stage using the neural activity and cursor movements during PVKF control [8]. Finally, the FORCE decoder is a nonlinear decoder that uses a recurrent echostate network (ESN) and outperforms a linear velocity Kalman filter (VKF) [16]. Our goal in choosing these studies was to demonstrate the simulator made correct predictions under very different decoder innovations, namely: (1) state-of-the-art linear decoding with intention estimation, (2) closed-loop decoder adaptation and retraining, and (3) nonlinear decoding.

1.3.2 BCI emulator

BCI emulator uses approximated neural data and true control policy. We employ BCI emulator to validate the neural encoder, which maps motor commands (recorded hand kinematics) to synthetic neural activity. The BCI emulator employs closed-loop experiments (Figure 1.2b). A user moves his or her hand, which generates synthetic activity that is subsequently decoded. The decoded cursor movement is not perfect and user makes online corrections to compensate for inaccurately decoded kinematics. Because the user sees the decoded output and adjusts his or her motor commands to better control the decoder, the BCI emulator incorporates a true (not approximated) user control policy. This enables us to therefore evaluate a neural encoder in isolation.

1.3.3 BCI simulator

We next sought to achieve pure software BCI simulator with approximated neural data and approximated control policy. We replace the user-in-the-loop in BCI emulation with an AI agent which knows how to generate proper movements to interact with BCI system and acquire targets as humans did in emulator (Figure 1.2c). We fixed the neural encoder and focused on agent optimization. Because the BCI emulator reproduced published studies, we reasoned that human-like control policy would lead to a purely software BCI simulator that also reproduces published studies.

1.4 Conclusion

BCI community takes two decades to have significant advances in 2D plane decoding. New innovations typically require months to years of experimental validation. This dissertation will guide the steps toward a pure software BCI simulator and validate with prior studies. We emphasize that our work serves as a middle ground to alleviate the need of monkey experiments, and allow quickly evaluate and characterize neural decoders before shifting to monkey experiments or clinic trials. We believe our work will significantly reduce the barrier and accelerate BCI research.

CHAPTER 2

Intracortical brain-computer interfaces

Intracortical brain-computer interfaces (BCIs) were initially demonstrated in the late 1990 and early 2000s [4, 5, 6], entering pilot clinical trials in 2004 [7]. The past two decades have seen significant advances to increase BCI performance, incorporating ideas from statistical inference [31, 32, 33, 34], feedback control [8, 9, 10], dynamical systems [11, 12], closed-loop decoder and neural adaptation [13, 14, 15], and deep learning [16, 17, 35]. These months to years BCI experiments provide ground truth decoder comparisons, collected with empirical neural recordings and a user-in-the-loop, that we use to validate our BCI emulator and simulator. We chose to replicate three distinct BCI studies, FIT-KF [9], ReFIT-KF [8], and FORCE [16], that improved BCI decoders through different innovations. This was to demonstrate that our work, which did not incorporate any decoder-specific information, could make general and correct predictions of BCI decoder performance across diverse innovations and distinct study settings.

2.1 Summary

FIT-KF, ReFIT-KF, and FORCE decoders reported superior performance over the then state-of-the-art VKF decoder, due to superior fine control to successfully hold the cursor over the target and faster time to first touch the target. This also led to smoother kinematic trajectories, with improved path efficiency and less deviation from the straight-line path. In detail, FIT-KF and ReFIT-KF achieve similar trial times with ReFIT-KF having faster first-touch time than FIT-KF, but FIT-KF having better dial-in time than ReFIT-KF. FIT-

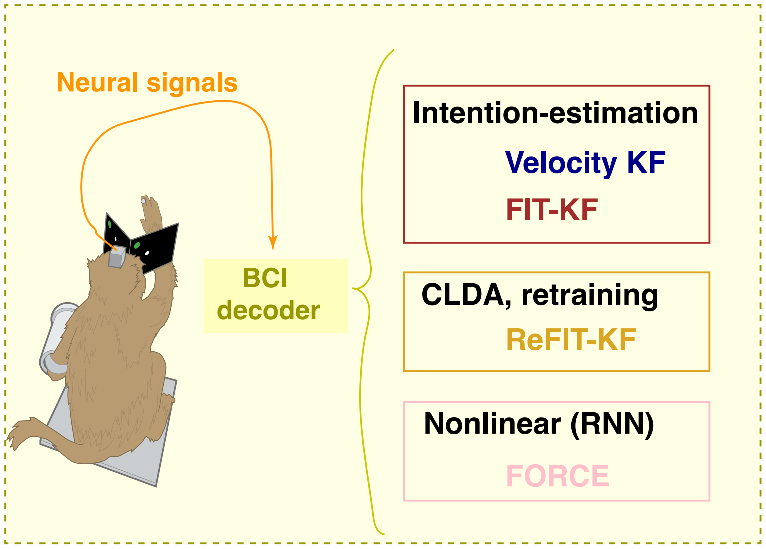


Figure 2.1: Intracortical monkey BCI experiment. Using true neural data and control policy to evaluate four distinct decoder algorithms. Neural signal was recorded from the motor cortical regions then decoded into cursor shown on the screen. Monkey adjusted his control policy due to imperfect decoded kinematics. Monkey's hand was free to move because they reflect the motor commands used to control the decoder.

KF had the shortest dial-in time, followed by ReFIT-KF, FORCE, and VKF. VKF had the longest first-touch time, ReFIT-KF had the shortest one, and FIT-KF and FORCE are in between. VKF achieved the widest trial-time distribution.

2.2 Introduction

BCIs aim to recover lost motor function and communication. The ultimate goal is to prove quality of life for people with paralysis. BCI systems are comprised of three major components as shown in Figure 2.1: (1) neural activity, typically recorded from motor cortical regions of the brain; (2) a decoder, which translates the neural recordings into control signals; and (3) a prosthesis, such as a computer cursor on a screen or a robotic arm, controlled by the decoder. To be clinically viable, BCIs must achieve high performance at a level justifying neurosurgery. The performance and robustness of BCI systems depend greatly on the decode algorithm (or “decoder”). Recent efforts have produced novel algorithms to increase BCI performance and robustness by incorporating ideas from statistical inference [31, 32, 33, 34], feedback control [8, 9, 10], dynamical systems [11, 12], closed-loop decoder and neural adaptation [13, 14, 15], and deep learning [16, 17, 35].

Validating our BCI emulator and simulator, we chose to replicate three distinct BCI studies that improved BCI decoders through different innovations. This was to demonstrate that our simulator, which did not incorporate any decoder-specific information, could make general and correct predictions of BCI decoder performance across diverse innovations and distinct study settings. We chose to replicate the BCI experiments that established the (1) FIT-KF [9], (2) ReFIT-KF [8], and (3) FORCE [16] decoders. The FIT-KF is a linear state-of-the-art decoder whose high performance is due to the incorporation of control theory inspired dataset augmentation (also known as intention estimation) [9]. The ReFIT-KF, which also incorporates intention estimation, is a decoder that employs two stages of BCI training, a form of closed-loop decoder adaptation (CLDA) [30, 14]. The ReFIT-KF decoder first requires the user to control a position velocity Kalman filter (PVKF), followed by a retraining stage using the neural activity and cursor movements during PVKF control [8]. Finally, the FORCE decoder is a nonlinear decoder that uses a recurrent echostate network (ESN) and outperforms a linear velocity Kalman filter (VKF) [16]. Our goal in choosing these studies was to demonstrate the simulator made correct predictions under very different decoder innovations, namely: (1) state-of-the-art linear decoding with intention estimation, (2) closed-loop decoder adaptation and retraining, and (3) nonlinear decoding.

2.3 Methods

2.3.1 Experiment setup

All surgical and animal care procedures were performed in accordance with National Institutes of Health guidelines and were approved by the Stanford University Institutional Animal Care and Use Committee. Experiments were conducted with two adult male rhesus macaques (Monkeys J and L). Monkey J (L) was implanted with two (one) 96 electrode Utah arrays (Blackrock Microsystems Inc., Salt Lake City, UT) using standard neurosurgical techniques. Monkey J's arrays were implanted in dorsal premotor cortex (PMd) and primary motor cortex (M1) as visually estimated from local anatomical landmarks, while Monkey L's array was implanted around the PMd/M1 border. Monkey J's (L's) arrays were implanted 75 (94) months prior to data collection for this work. The monkeys made point-to-point reaches in a 2D plane with a virtual cursor controlled by the contralateral arm or by a BMI. The experimental setup has been previously described (e.g., [36, 37]). The virtual cursor and targets were presented in a three-dimensional (3D) environment (MusculoSkeletal Modeling Software (MSMS), Medical Device Development Facility (MDDF), USC, Los Angeles, CA). Hand position data were measured with an infrared reflective bead tracking system (Polaris, Northern Digital, Ontario, Canada). Spike counts were collected by applying a single threshold, set to $-4.5\times$ the root-mean-square of the high-pass filtered spike voltage per electrode [38]. For BCI decoders, the number of electrodes were either 96 (M1) or 192 (M1+PMd), and spike counts were binned in either 25 or 50 ms bins depending on the study (see Table 2.1). Behavioral control and neural decode were run on separate PCs using Simulink/xPC platform (Mathworks, Natick, MA) with communication latencies of 3 ms. This enabled millisecond timing precision for all computations. Neural data were initially processed by the Cerebus recording system (Blackrock Microsystems Inc., Salt Lake City, UT) and were available to the behavioural control system within 5 ± 1ms. Visual presentation was provided via two LCD monitors with refresh rates at 120 Hz, yielding frame updates

Table 2.1: Task and recording settings for each of the three studies. When the number of electrodes is 96, this corresponds to one Utah Array in primary motor cortex. 192 electrodes corresponds to two Utah Arrays, one in primary motor cortex and the other in dorsal premotor cortex.

	Decoder	Radius (cm)	Acceptance window (cm)	Bin width (ms)	# electrodes
Gilja et al. [8]	Hand	8	5	N/A	N/A
	PVKF	12	6	50	96
	ReFIT-KF	8	5	50	96
	VKF	8	5	50	96
Fan et al. [9]	Hand	8	4	N/A	N/A
	PVKF	12	6	50	192
	ReFIT-KF	8	4	50	192
	FIT-KF	8	4	50	192
	VKF	8	4	50	192
Sussillo et al. [16]	Hand	8	4	N/A	N/A
	FORCE	8	4	25	96
	VKF	8	4	50	96

of 7 ± 4 ms. Two mirrors visually fused the displays into a single 3D percept for the user, creating a Wheatstone stereograph. All tasks were restricted to a two-dimensional plane.

2.3.2 Tasks

The three published studies we used to validate our emulator used variants of the center-out-and-back task and the pinball task. The center-out-and-back task was used to collect training data for PVKF, ReFIT-KF, FORCE, and VKF decoders. The pinball task was used to collect training data for FIT-KF decoder. All studies used the center-out-and-back tasks to quantify decoder performance. The center-out-and-back task parameters for each study are summarized in Table 2.1.

2.3.2.1 Center-out-and-back task

In the center-out-and-back task, eight "radial targets" were uniformly placed on the circumference of an 8-cm or 12-cm radius circle. One "center target" was placed at the center of the circle. Each target had a square acceptance window centered around the target, with side length 4, 5, or 6 cm depending on the study. The subject had to hold the cursor within the target acceptance window for 500 contiguous milliseconds to successfully acquire the target. After acquisition of the center target, one of the eight radial targets would be randomly prompted. The target had to be acquired within 3 seconds, or the trial was counted as a failure. After a successful acquisition of a radial target, or following the failure to acquire any radial target, the center target was prompted.

2.3.2.2 Pinball task

In the pinball task implemented by Fan et al. [9], targets were randomly prompted in a 20-by-20 cm workspace. The target's position was randomly sampled each trial. Each target had a 4 cm square acceptance window. The subject had to hold the cursor within the target acceptance window for 750 contiguous milliseconds to successfully acquire the target. The target had to be acquired within 3 seconds, or the trial was counted as a failure. After a successful acquisition or following the failure to acquire any target, the next random target was prompted. There were two additional constraints on the next target location. First, the minimum distance between each random target was 4 cm to avoid two successive trials having overlapping acceptance windows. If the next target was within 4 cm of the previous target, the next target position would be re-sampled until the next target was more than 4 cm away. Second, the if the next target was more than 14 cm away from the previous target, the next target position would be re-sampled until the next target was less than 14 cm away.

2.3.3 BCI metrics

The straight forward way to evaluate decoder performance is using success rate and trial time. Trial time can also be divided into fist-touch time presenting ballistic control and dial-in time presenting fine control. The decoded trajectory is characterized with distance ratio and maximum deviation. All metrics are detailed in the following.

2.3.3.1 Trial time

Trial time is time from when the target shows up to when the cursor last enter the target acceptance window prior to successfully holding the target. Therefore, the holding period is not included in trial time. a shorter trial time means the user can quickly acquire targets.

2.3.3.2 Success rate

Success rate is the ratio of successful trials divided by the total attempt trials. In the center-out-and-back task, we only report the success rate in center-out trials. A good decoder should result in a high success rate.

2.3.3.3 Dial-in time

Dial-in time (DIT) is the time from when the cursor first enters the target acceptance window to when it last enters the target acceptance window prior to successfully holding the target. If the user enters the acceptance window and successfully holds the target without ever leaving the acceptance window, DIT is equal to zero. An illustration of the DIT is shown in Figure 2.2a.

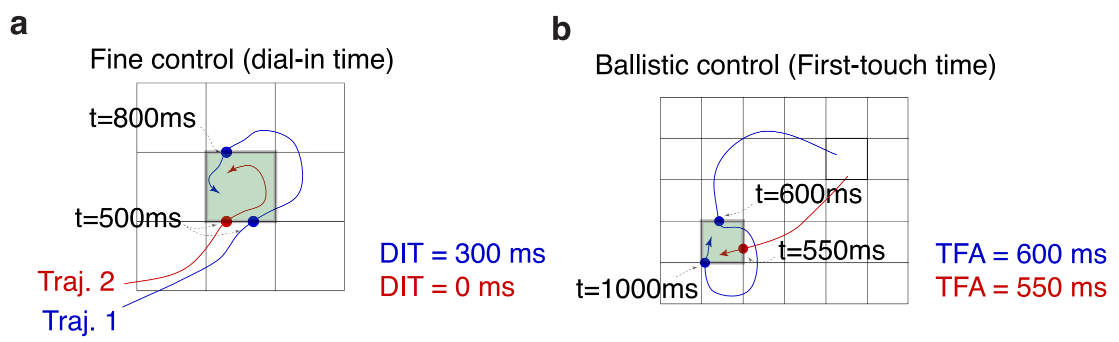


Figure 2.2: Illustration of dial-in time (DIT) and first-touch time (FTT) **(a)** DIT example. **(b)** FTT example. Dial-in time (DIT) represents fine control. In **a**, trajectory 1 (blue) enters the desired target but due to poor fine control, exits and re-enters, leading to DIT = 300 ms. Trajectory 2 (red) enters, and with fine control, is able to acquire the target, leading to DIT = 0 ms. First-touch time (FTT) represents ballistic control. In **b**, trajectory 1 (blue) enters the desired target at FTT = 600 ms. Although it exits and re-enters at 1000 ms, this does not affect the FTT. Trajectory 2 (red) enters the desired target more quickly at FTT = 550 ms.

2.3.3.4 First touch time

First-touch time (FTT) is the time from trial start to when the cursor first enters the target acceptance window. An illustration of FTT is s shown in Figure 2.2b.

2.3.3.5 Distance ratio

Distance ratio, also known as the inverse of path efficiency, is the distance traveled by the cursor divided by the straight line distance (i.e., the shortest trajectory) from the start point to the end point of a trial. Distance ratio quantifies path inefficiency. When the distance ratio is large, then the path taken to acquire the target is less direct.

2.3.3.6 Maximum deviation

Maximum deviation is the maximum distance between the cursor trajectory and the straight line from the start point to the end point of a trial. Maximum deviation therefore quantifies how far the cursor deviates from the shortest trajectory.

2.3.4 BCI decoder algorithms

Three distinct BCI studies were chosen that improved BCI decoders through different innovations including those that require multiple stages of training, as well as linear and nonlinear decoders. Because each BCI decoder algorithm has been well-documented in its published studies, we comment on these more briefly in our Methods, summarizing their innovations and highlighting key differences between the decoders. The FIT-KF, ReFIT-KF, and VKF are all based on a linear dynamical system model where the cursor kinematics at time t, $\mathbf{x}_t$, are the state and binned spike counts at time t, $\mathbf{y}_t$, are the observations. $\mathbf{y}_t$ was a 96D or 192D vector containing the binned spike counts of 96 or 192 channels at time t (depending on the study), and $\mathbf{x}_t$ was a 5D vector representing the cursor's x- and y-positions, velocities, and a bias term of 1 to model a baseline spike count. The dynamical system is:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{A}\mathbf{x}_{t-1} + \mathbf{w}_t && (2.1) \\ \mathbf{y}_t &= \mathbf{C}\mathbf{x}_t + \mathbf{q}_t && (2.2) \end{aligned}$$

where $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{W})$ and $\mathbf{q}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ are Gaussian noise. The parameters of this model, $\theta = \{\mathbf{A}, \mathbf{C}, \mathbf{W}, \mathbf{Q}\}$, were inferred using supervised learning with maximum-likelihood estimation, where datasets comprised the paired cursor kinematics and neural activity, $\{\mathbf{x}_t, \mathbf{y}_t\}_{t=1,\ldots,T}$. For more details, refer to Wu et al. [31], Kim et al. [22], Gilja et al. [8], Kao et al. [39]

To decode, we used the Kalman filter algorithm to infer the state of the dynamical system, $\hat{\mathbf{x}}_t$, given the observed spike counts $\mathbf{y}_t$. This enabled recursive estimation of the state, $\hat{\mathbf{x}}_t$, from a new neural observation, $\mathbf{y}_t$, and the previously decoded state $\hat{\mathbf{x}}_{t-1}$. We calculate the Kalman steady states to have two Matrices, M_1 and M_2, then decode neural data by calculating:

$$\hat{\mathbf{x}}_t = \mathbf{M}_1\hat{\mathbf{x}}_{t-1} + \mathbf{M}_2\mathbf{y}_t \qquad (2.3)$$

where M_1 and M_2 are calculated from the following.

Algorithm 1 Kalman filter recursion

```
Require: $\mathbf{A}, \mathbf{W}, \mathbf{C}, \mathbf{Q}$
Ensure: $\mathbf{M}_1$ and $\mathbf{M}_2$
  Initial $\Sigma_{0|0} = 0$
  repeat
      $\Sigma_{t|t-1} = \mathbf{A}\Sigma_{t-1|t-1}\mathbf{A}^T + \mathbf{W}$
      $\Sigma_{t|t} = \Sigma_{t|t-1} - \Sigma_{t|t-1}\mathbf{C}^T(\mathbf{C}\Sigma_{t|t-1}\mathbf{C}^T + \mathbf{Q})^{-1}\mathbf{C}\Sigma_{t|t-1}$
      $\mathbf{K}_t = \Sigma_{t|t-1}\mathbf{C}^T(\mathbf{C}\Sigma_{t|t-1}\mathbf{C}^T + \mathbf{Q})^{-1}$
  until $\mathbf{K}_t$ converges
  $M_1 = \mathbf{A} - \Sigma_\infty \mathbf{C}\mathbf{A}$
  $M_2 = \Sigma_\infty$
```

2.3.4.1 VKF

The Velocity Kalman filter (VKF) is a linear decoder trained from concurrent cursor kinematics and spiking activity. The "**C**" matrix considers only the contribution from velocity. Across all studies, and in our emulator and simulator, the VKF was trained by collecting approximately 500 trials of a center-out-and-back task where targets were on a 12-cm radius circle with 4 cm square acceptance windows. All studies used VKF with spike counts binned in non-overlapping 50 ms bins. Note, the VKF does not incorporate intention estimation or CLDA.

2.3.4.2 PVKF

The position velocity Kalman filter (PVKF) is a linear decoder trained from concurrent cursor kinematics and spiking activity. The "**C**" matrix considers both contribution from position and velocity. Similar to VKF decoder, PVKF was trained by collecting approximately 500 trials of a center-out-and-back task where targets were on a 12-cm radius circle with 4 cm square acceptance windows. All studies used PVKF with spike counts binned in non-overlapping 50 ms bins. Note, the PVKF does not incorporate intention estimation or

CLDA.

2.3.4.3 FIT-KF

The Feedback Intention Trained Kalman filter (FIT-KF) is a linear decoder trained from concurrent cursor kinematics and spiking activity. The FIT-KF is trained from approximately 500 trials of the pinball task. FIT-KF training includes intention estimation, a form of dataset augmentation where cursor velocities in the training set are rotated to point towards the target at every time step. When the cursor is in the target acceptance window, intention estimation sets the cursor velocity to zero. These changes assume that at every point in time, the BCI user is giving motor commands to move the cursor towards the target, and that when in the acceptance window, the BCI user is attempting to hold the cursor still over the target. The FIT-KF also pre-processes the data to exclude the first 250 ms of data at the start of a trial, since this corresponds to the reaction time of the monkey. It also excludes the dial-in time (between when the user first touched and last touched the target) from training. The FIT-KF inference used a causal intervention in Kalman filter inference, where the covariance of the state position estimate is set to zero, since the user sees the true position of the cursor. Finally, the FIT-KF subtracts the contribution of position to the neural activity through a linear mapping, the details of which are described in Gilja et al. [8].

2.3.4.4 ReFIT-KF

The Recalibrated Feedback Intention-Trained Kalman Filter (ReFIT-KF) is a linear decoder that involves two stages of training, a form of CLDA. Concurrent cursor kinematics and spiking activity were used to train a Position Velocity Kalman Filter (PVKF), using the same task as used to train the VKF. Subsequently, the PVKF was used in closed-loop control to collect 500 trials of a center-out-and-back task where the target radius was 12 cm and each target had a 6 cm square acceptance window. The ReFIT-KF was then trained from the closed-loop 500 PVKF center-out-and-back trials. The PVKF data was augmented to include intention estimation. The ReFIT-KF inference also used the causal intervention and position contribution subtraction in the FIT-KF.

2.3.4.5 FORCE

The FORCE decoder is a nonlinear decoder. The FORCE decoder is a recurrent neural network (RNN) that takes the form of an echostate network (ESN). The parameters of the FORCE decoder were trained using FORCE learning [40]. The FORCE decoder was trained using concurrent cursor kinematics and spiking activity using the same task used to train the VKF. The network was trained on four passes through the data.

2.4 Results

BCI datasets were recorded when monkey performed tasks in each study. BCI datasets include recorded neural activity, cursor kinematics, and hand kinematics. We performed offline analysis on BCI datasets where monkey performed a center-out-and-back task with their hand. We found the delay between hand speed and binned spike counts was 200 ms which is critical when training a neural encoder. Online decoder performance were summarized as ground truth comparisons for our work later.

2.4.1 Monkey data offline analysis

Monkey performs center-out-and-back task with their hand. We analyze the recorded neural activity and hand kinematics. We also train decoders and evaluate decoder performance in an offline manner.

2.4.1.1 A delay due to neural path from motor cortex to arm EMG

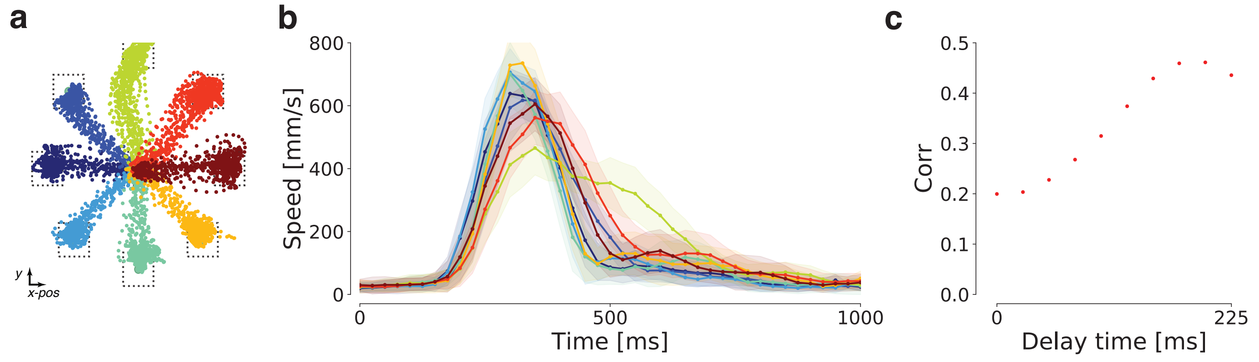


Figure 2.3: Delay estimation between neural activity in motor cortex and hand movements. **(a)** Hand movements and **(b)** speed of center-out trials in a center-out-and-back task. 8 different colors correspond to 8 target directions. **(c)** Correlation coefficient on various delay, and 200 ms delay achieved highest correlation.

Hand position of center-out trials recorded in center-out-and-back task is shown in Figure 2.3a. Each color corresponds to different targets. Due to its body constraints, hand trajectories are not the same for the same target, and not fully straight. Hand speed profile is shown in Figure 2.3b. For each direction, the hand speed profile is a beautiful bell-shape. When target is shown at time 0, hand movement is initiated after 200 ms which is called "response time". Interestingly, the neural activity in motor cortex changes dramatically after time 0 when target is shown (Figure 2.4a). There is a delay from neural activity in motor cortex to arm EMG. We calculate correlation coefficient between hand speed and binned spikes. We found that the 200 ms delay achieves the highest correlation shown in Figure 2.3c. We plot the relationship between hand speed and binned spike in Figure 2.4b and c. The linear relationship in Figure 2.4c is clear than in Figure 2.4b. This 200 ms delay between hand speed and neural activity plays an important role when train neural encoder. We will discuss the detail in chapter 3.

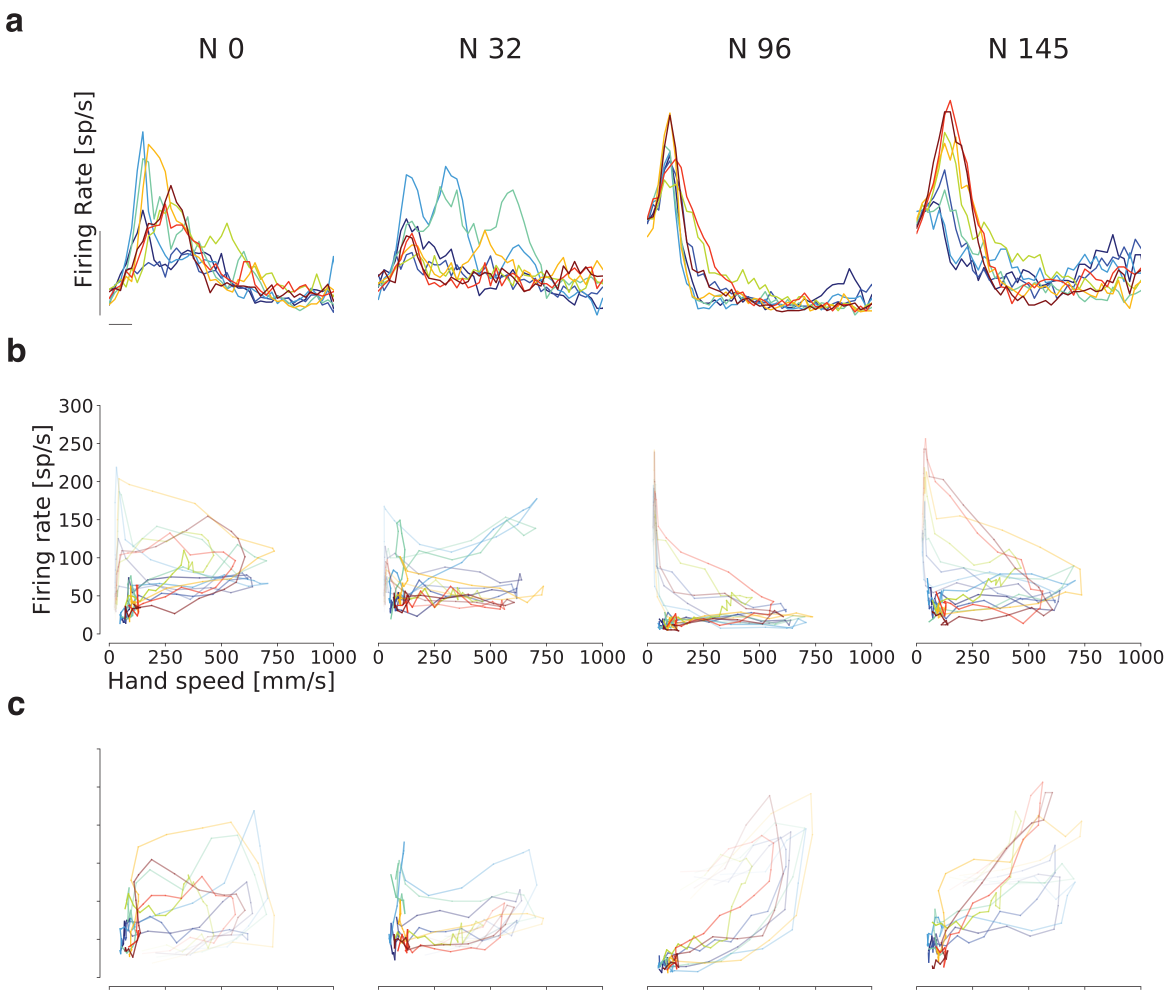


Figure 2.4: Estimate the delay from motor cortex to hand movements. **(a)** PSTH of four neurons. **(b)** Relationship between firing rate and hand speed given no delay. **(c)** Relationship between firing rate and hand speed given 200 ms delay. Hand movements were mainly initiated by neural activity in motor cortex. The neural path from motor cortex to muscle EMG causes the delay which is always ignored when training neural encoders and decoders. We found that M1 and PMd are positively correlated with variability of future hand speed (delay = 200 ms) in average across 192 neurons. As expected, comparing **b** and **c**, the relationship of hand speed and PSTH is much linear when apply 200 ms delay.

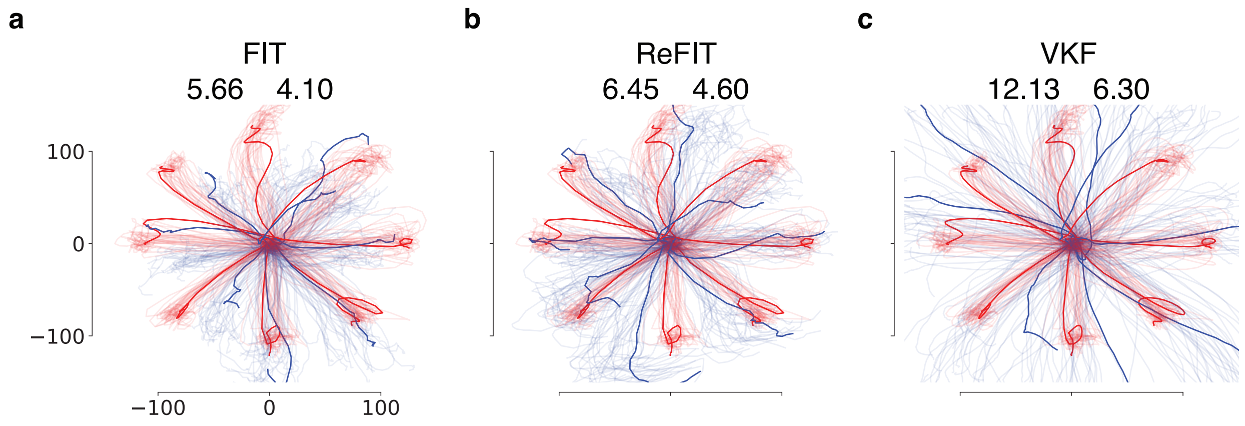


Figure 2.5: Hand movements and decoded movement from open-loop recorded neural activity with three decoders The decoders are **(a)** FIT-KF, **(b)** ReFIT-KF, and **(c)**. In each panel, red lines represent hand trajectories, and blue lines represents decoded trajectories. One randomly chosen trial for each direction is shown in bold lines. NRMSE of decoded positions and velocities are shown in each panel.

2.4.1.2 BCI decoder offline performance

Decoder performance in offline dataset provides insight of which decoder is promising. We evaluate three decoders: FIT-KF, ReFIT-KF, and VKF with offline dataset where monkey performed center-out-and-back task with its hand. We evaluate normalized root mean square error (NRMSE) of position and velocity between hand and decoded cursor. In Figure 2.5, hand trajectory is in red and cursor trajectory is in blue. We found that FIT-KF and ReFIT have similar performance in NRMSE and are better than VKF. FIT-KF decoded trajectory is general shorter followed by ReFIT then VKF. FIT and ReFIT algorithms take intention estimation into consideration by rotating the velocity vector point toward target and set velocity to be zero when the cursor is on the acceptance window. The decoded velocity is therefore shorter than VKF. Overall, these three decoders are decoded into reasonable trajectories. The eventual performance needs to be evaluated in online experiments.

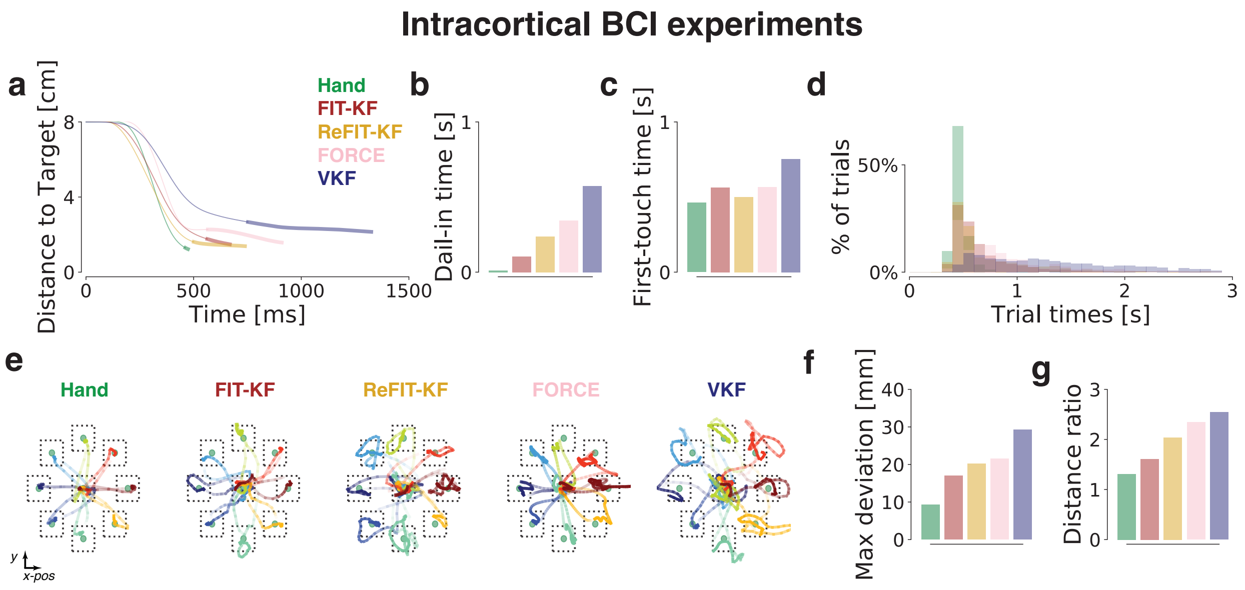


Figure 2.6: **(a)** Distance-to-target plots from intracortical BCI experiments for the various decoders. The bolded line corresponds to DIT. **(b, c)** Average DIT and FTT, respectively, from BCI experiments. **(d)** Distribution of trial times for each decoder. **(e)** Randomly sampled trajectories for each decoder. **(f)** Max deviation from the straight line path for each decoder. **(g)** Distance ratio (cursor path length divided by straight path length) for each decoder. Distance ratio is the inverse of path efficiency. Decoder performance are summarized in Table 2.2.

Table 2.2: Decoder performance in BCI with monkey J. TT, FTT, and DIT are trial time, first-touch time, and dial-in time, respectively.

Decoder	TT [ms]	FTT [ms]	DIT [ms]	distance ratio	max deviation [mm]	success rate	# of center-out trials
hand	473.79	461.88	11.91	1.31	9.27	1.00	741
FIT-KF	668.29	563.5	104.79	1.61	17	0.99	2383
ReFIT-KF	738.42	500.78	237.64	2.04	20.21	0.99	1377
FORCE	909.42	566.18	343.23	2.34	21.48	1.00	1212
VKF	1325.17	752.51	572.65	2.54	29.28	0.81	1368

2.4.2 BCI decoder online performance

Monkey performed center-out-and-back task for several days. Decoders are retrained on the same day recorded dataset. The results of the published studies are combined in Figure 2.6. Each study reported superior performance over the then state-of-the-art VKF, due to superior fine control to successfully hold the cursor over the target and faster time to first touch the target. This also led to smoother kinematic trajectories, with improved path efficiency and less deviation from the straight-line path (Figure 2.6e-g). In detail, FIT-KF has shortest dial-in time, showing the best fine control. ReFIT-KF has shortest first-touch time, showing the best ballistic control. VKF shows circling trajectories while dialing in, indicating the difficulty to stabilize.

2.5 Conclusion

We perform offline analysis on recorded dataset. We found a delay between hand speed and neural activity is 200 ms which is critical and a key innovation in training neural encoder later discussed in Chapter 3. In order to validate our emulator and simulator works, we choose prior studies across diverse innovations and distinct study settings. We therefore choose the BCI experiments that established the (1) FIT-KF [9], (2) ReFIT-KF [8], and (3) FORCE [16] decoders. We quantitatively compare the decoder performance with trial time, first-touch time, dial-in time, distance ratio, max deviation, and success rate. The FIT-KF is a linear state-of-the-art decoder whose high performance is due to the incorporation of control theory inspired dataset augmentation (also known as intention estimation) [9]. The ReFIT-KF, which also incorporates intention estimation, is a decoder that employs two stages of BCI training, a form of closed-loop decoder adaptation (CLDA) [30, 14]. The ReFIT-KF decoder first requires the user to control a position velocity Kalman filter (PVKF), followed by a retraining stage using the neural activity and cursor movements during PVKF control [8]. Finally, the FORCE decoder is a nonlinear decoder that uses a recurrent echostate network (ESN) and outperforms a linear velocity Kalman filter (VKF) [16].

We later show in Chapter 4 and Chapter 5 that our emulation and simulation qualitatively and quantitatively match decoder performance in monkey experiments.

CHAPTER 3

Neural encoder

3.1 Summary

We introduce neural encoders for synthesizing motor cortical neural population activity from kinematic behavior. We trained PPVT neural encoder and various deep-learning-based neural encoders in a supervised fashion using data collected while a monkey performed a reaching task. We demonstrate that RNN neural encoders are significantly better than PPVT neural encoder in reproducing single neuron PSTH variability, neural population dynamics, and neurally decoded movements. Importantly, RNN neural encoder incorporating neural path delay and RNN input weight regularization more faithfully reproduces neural activity and decoded kinematics in validation dataset. Our results indicate that RNN neural encoders may significantly improve the fidelity of BCI simulator.

3.2 Introduction

Generating neural activity has wide implications in systems neuroscience and neural engineering. Neural encoding models (neural encoders) have produced insight into computations in early visual processing stages (e.g., [41, 42, 43, 44]). Neural encoders have been used to argue that the motor cortex does not represent movement, but is rather a dynamical system (e.g., [45, 46]). In engineering applications, neural encoders are proposed to augment memory function through modeling hippocampal activity [47] and deliver realistic sensory sensation through stimulation of sensory cortices [48, 49, 50]. Finally, motor cortical neural

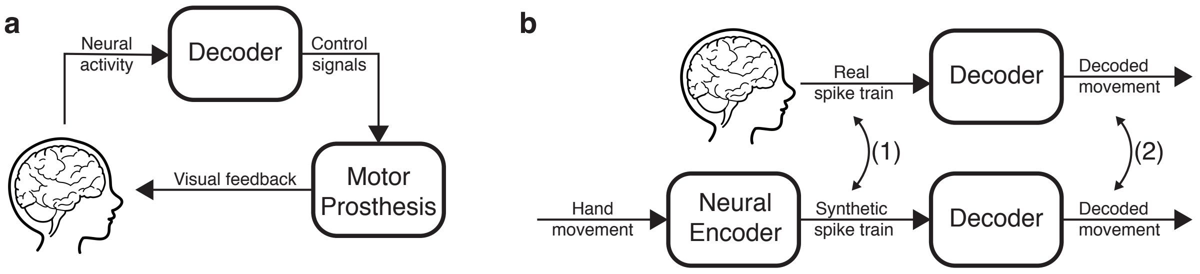


Figure 3.1: BMI system and neural encoder. **a** Neural activity is recorded from electrode arrays implanted in motor cortex. A decoder transforms neural activity into control signals which guides a motor prosthesis. The BCI user receives visual and potentially sensory feedback, adjusting his or her control policy in response to observing the decoded movement. **b** A neural encoder takes user kinematics and produces synthetic neural activity. A good encoder reproduces single neuron variability, neural population dynamics, and decoded movement from real neural activity. Here, we compare (1) the PSTHs and neural dynamics of recorded and synthetic neural activity, and (2) decoded movements from real and synthetic neural activity.

encoders are a critical component for simulating motor brain-computer interfaces (BCIs) [2].

We focus on building motor cortical neural encoders for intracortical motor BCI simulation, with the goal of substantially accelerating clinical translation. In BCI experiment, motor prostheses are guided by a control signal decoded from motor regions of the brain as illustrated in Figure 3.1a. In contrast to closed-loop animal experiments and offline decoding, Cunningham et al. [2] introduced a middle ground: neural activity can be simulated to more rapidly perform experiments that incorporate the closed-loop nature of BCI systems. This online prosthetic simulator (OPS) enables faster decoder evaluation incorporating closed-loop feedback control policies without requiring intracortical recordings from the brain. Their work showed using OPS that BMI performance should increase with smaller bin width, an empirical result they and another study confirmed [10]. In the OPS, spike trains were synthesized via an inhomogeneous Poisson process whose rate was described by a speed-modulated tuning curve model [51]. We call this model the Poisson Process Velocity Tuning (PPVT) model. Several studies have demonstrated that this model does not reproduce complex heterogeneity in neuron firing rates, including changing preferred directions with time [52], nonlinear structure in the neural population activity [53], and neural population dynamics (e.g.,[45, 46]).

Deep learning has achieved advances in a variety of research areas including, but not limited to, computer vision [54, 55, 56], object detection [57, 58, 59, 60], neural signal denoising [53], early visual representations modeling [61], and sensory cortex modeling [62]. Recent work has applied deep learning, specifically convolutional neural networks (CNNs), to model retinal activity. Analysis of these CNNs revealed roles for feedforward inhibition, recurrent lateral connections, and noise in explaining empirical neural responses [63, 64, 62]. Though approximating neural activity accurately may appear challenging, the low-dimensional nature of neural population activity during point-to-point reaches simplifies the problem and enables us to build accurate neural encoders. We trained deep learning based neural encoder in a supervised fashion using data collected while a monkey performed a reaching task, and evaluate synthetic neural activity and decoded movements as illustrated in Figure 3.1b. We demonstrate that our work can better reproduce single neuron PSTH variability, neural population dynamics, and neurally decoded movements. Importantly, we evaluate generalization of our model with validation datasets which were recorded when monkey controlled with BCI decoders. We found that RNN neural encoder trained with a neural path delay and RNN input weights regularization more faithfully reproduces neural activity and decoded kinematics in validation dataset, indicating our RNN neural encoder closely models the kinematic-to-neural relationship and generalizes to the datasets recorded in closed-loop experiments. We later will demonstrate that our RNN neural encoder can reproduce online decoder performance with BCI emulator and simulator in chapter 4 and 5

3.3 Methods

3.3.1 Dataset

We focused on the dataset of Monkey J from the published study [9]. Monkey J had the same experiment setting as described in chapter 2. We used concurrent kinematics and intracortical threshold crossings recorded when Monkey J perform center-out-and-back task with his own hand as training dataset to train neural encoder. We also used the same day recorded online BCI data where Monkey J used FIT-KF and ReFIT-KF to perform the task as validation dataset for evaluating generalization performance of neural encoders.

3.3.2 Neural encoders

We focus on the neural encoder modeling neural activity in motor cortical regions by transforming kinematics, x_t, into binned spike counts, y_t. All presented models generate a neural firing rate. Neural encoders were trained to reproduce the empirical binned spike counts recorded from multi-unit threshold crossing activity. We calculated binned spike counts by treating the neural encoder firing rates as the rate of an inhomogeneous Poisson process. We discuss a neural encoder, Poisson Process Velocity Tunning model, used in prior study [2] and introduce variants of RNN neural encoders.

3.3.2.1 Preferred direction (PD) model

The PD model is based on a tuning curve model where each neuron's firing rate is explained as a function of the reach angle, with the angle eliciting the highest firing rate called the "preferred direction" (PD) [51]. The PD model calculates firing rate based on a cosine tuning curve. Subsequently, binned spike counts, y_t, are generated by treating the PD model firing rate as the underlying rate of an inhomogeneous Poisson distribution. The equation to generate binned spike counts is:

$$\lambda_t \quad = \quad \lambda_o + (\lambda_{max} - \lambda_o)\cos(\theta_t - \theta_{max}) \tag{3.1}$$

with spikes generated according to the equation below.

$$\hat{y}_t|\lambda_t \sim \text{Poisson}\,(\max(0, \lambda_t)) \tag{3.2}$$

where: λ_t is the neural firing rate, y_t is the binned spike counts, θ_t is the reach angle, θ_{max} is the PD, and λ_o is an offset firing rate.

The neuron's modeled firing rate, λ_t, ranges from $2\lambda_o - \lambda_{max}$ (when the reach angle is opposite to the PD) to λ_{max} (reach angle aligned to the PD). The model parameters were found using the technique of [51], where firing rates were averaged from 200ms to 500ms after trial initiation. Because neural firing rates cannot be negative, we draw spike counts with rates lower bounded by 0, i.e., with rate $\max(0, \lambda_t)$.

3.3.2.2 Poisson process velocity tuning (PPVT) model

The PPVT model incorporates a tunning curve model and reach speed. Tunning curve model explaines each neuron's firing rate as a function of reach angle, with the angle eliciting the highest firing rate called the "prefreed direction" (PD). The firing rate is calculated based on a cosine tunning curve and linearly scaled based on the speed of reach. Firing rate is calculated as

$$\lambda_t = \lambda_0 + (\lambda_{max} - \lambda_0)\cos(\theta_t - \theta_{max}) \cdot s_t$$

with spikes generated according to equation 3.2. In this equation, s_t is the scaled movement speed at time t. We scale s_t so that the firing rates, when decoded, produce reasonable trajectories. This scaling is subject dependent, since different subjects may reach with different vigor.

3.3.2.3 Generalized linear models (GLMs)

The GLM model is a flexible generalization of ordinary linear regression [65, 66, 67]. In its most basic form, we calculate the rates as:

$$\lambda_t = k \cdot x_t + \text{noise}, \tag{3.3}$$

where k is a vector of weights and x_t are the kinematic inputs. Unless otherwise stated, x_t is the 2D position and velocity of the hand at time t. The GLM can be extended to incorporate different noise distributions and a link function relating inputs to the rates. We performed maximum likelihood estimation through iteratively reweighted least squares.

3.3.2.4 Multilayer perceptron (MLP) model

The multilayer perceptron (MLP) model is a nonlinear, fully connected feedforward neural network. As a feedforward network, it does not model any firing rate dynamics but enables a data-driven, nonlinear approach to predict firing rates from kinematics. The MLP is a universal function approximator and has higher capacity than the linear neural encoders [68]. The MLP model takes kinematics, x_t, and generates firing rates, λ_t. We denote an MLP layer as

$$\begin{aligned} h_t &= \text{MLP}_N^f(x_t) \quad (3.4) \\ &= f(Wx_t + b), \quad (3.5) \end{aligned}$$

where f is activation function, N is the number of neurons, $W \in \mathbb{R}^{N \times \dim(x_t)}$, $b \in \mathbb{R}^N$, and $h_t \in \mathbb{R}^N$. We used the sigmoid activation function, i.e., $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ in all hidden layers. Because firing rates are non-negative and not bounded by 1, we used an exponential nonlinear activation function in the final layer, i.e., $f(x) = exp(x)$. The overall neural encoder we tested is:

$$\lambda_t \quad = \quad \mathrm{MLP}^{\exp}_{192}(\mathrm{MLP}^{\sigma}_{128}(\mathrm{MLP}^{\sigma}_{64}(\mathrm{MLP}^{\sigma}_{32}(x_t)))), \tag{3.6}$$

with spikes generated according to equation 3.2. To optimize the model, we maximized the log-likelihood of the empirical data by assuming that empirical binned spike counts, y_t, were Poisson distributed and conditionally independent given the firing rate, λ_t. Optimization was performed in batches of trials. The log-likelihood function is,

$$\mathcal{L} = \sum_i \sum_t \left(y_t^i \log \lambda_t - \lambda_t \right), \tag{3.7}$$

where i is iterating over trials in a batch, t is iterating over the time in trial i, and y_t^i is the binned spike counts at time t on trial i. We performed optimization using stochastic gradient descent with the first order Adam optimizer [69].

3.3.2.5 RNN

Our RNN model was trained in a supervised manner with recorded hand kinematics comprising 2D position and velocity (inputs) and binned spike counts (outputs) in monkey center-out-and-back task. Kinematics and binned spike counts were evaluated at 25 ms bin width resolution. The complete RNN neural encoder (Figure 3.2) is

$$\lambda_t = \mathrm{MLP}^{\exp}_{192}(\mathrm{MLP}^{\sigma}_{192}(\mathrm{MLP}^{\sigma}_{192}(\mathrm{RNN}^{\sigma}_{192}(x_t)))) \tag{3.8}$$

with spikes generated according to equation 3.2. The RNN was trained in Seq2Seq fashion by maximizing the log-likelihood of the observed binned spike counts under the assumption that follow the Poisson distribution. Instead of using concurrent hand kinematics

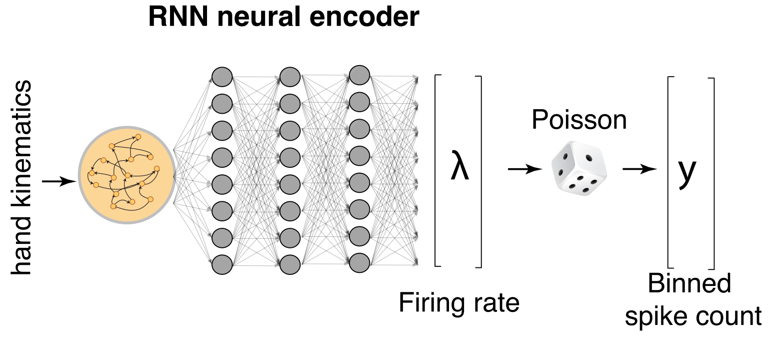


Figure 3.2: Architecture of the RNN neural encoder which takes hand kinematics to predict firing rates of 192 neurons. RNN was trained by maximizing the log-likelihood under the assumption of Poisson distribution. Binned spike counts were sampled under Poisson distribution given estimated firing rates.

and binned spike counts to train neural encoder, we delayed binned spike counts for 200 ms which achieved highest correlation coefficient with respect to hand speed. In addition, regularized RNN neural encoder has regularization on input weights of RNN cell. All RNN neural encoders were optimized with stochastic gradient descent, using the Adam optimizer, and initialized with the Xavier uniform initialization. The final model trained with delay binned spike counts is called delayed-regularized RNN neural encoder, and was used in both simulator and emulator.

3.3.3 Metrics

Contribution ratio estimate the ratio of contribution from external input and internal recurrent state. The contribution is defined as weight multiplied with variable. The ratio is defined as

$$\text{Contribution ratio} = \frac{\text{Input contribution}}{\text{Recurrent contribution}} = \frac{\mathbf{W}_{input}\mathbf{H}^{pv}}{\mathbf{W}_{recurrent}\mathbf{s}} \tag{3.9}$$

where $\mathbf{W}_{input}$ and $\mathbf{W}_{recurrent}$ are input and recurrent weight of RNN cell, respectively. H^{pv} is external input, and s is internal state. A ratio close to zero means that the recurrent contribution dominates the state changes. In contrast, a large ratio means that the external input dominate the state changes.

Pearson correlation coefficient (PCC) of PSTH and neural trajectory in low dimension. We report the PCC between BCI and synthetic PSTHs. The data were binned at intervals of 25 ms. For each channel, we concatenated the PSTH for each of eight target reach conditions into a vector. We then calculated the PCC between these vectors for real and synthetic PSTHs, and average across channels. We also report the PCC between BCI and synthetic neural trajectories from PCA. PCA is an orthogonal transformation of the neural data that maximizes the variability of the data in low dimensional projections. We performed PCA on PSTHs to emphasize across-condition variability. When comparing neural trajectories, we projected both real and synthetic neural population activity into the PCs found from real data.

Normalized root-mean-square error (NRMSE) of PSTH and neural trajectory in low dimension. NRMSE defined as

$$\mathrm{NRMSE}(y, \hat{y}) = \frac{\sqrt{\mathrm{Mean}(\hat{y} - y)^2}}{\mathrm{std}(y)} \tag{3.10}$$

where y is the ground truth, $\hat{y}$ is the predicted output, and "std" stands for standard deviation. In PSTH case, for each channel, we also concatenated the PSTH for each of eight target reach conditions into a vector. We then calculated the NRMSE and average across channels. In neural trajectory case, for each dimension, we also concatenated into a vector in the same way. We then calculated the NRMSE and average across top N dimension. N is the number of PCs that capture over 90 % of the PSTH variance.

jPCA is a rotation of the top PCs that reveals rotational structure in data. We applied jPCA after finding top PCs. This comprised finding a skew symmetric matrix least-squares mapping from the position of the neural trajectory to its velocity, with additional details in Churchland et al. [45]. We report $\mathrm{R}^2_{\mathrm{skew}}$, which is the variance explained in predicting the neural trajectory velocity from its position. A higher R^2_{skew} indicates that the system is better described by rotational dynamics.

NRMSE of decoded velocity from the real and synthetic neural activity is used to quantify the performance of neural encoder in reproducing neural activity.

Neural activity from monkey and neural encoder was decoded into cursor velocity. We then calculate NRMSE and average across trials. NRMSE follows the Equation 3.10.

3.4 Results

We found that deep learning models outperformed representational tuning models, PD and PPVT, in reproducing recorded single electrode activity, neural population activity, and decoded kinematics. Importantly, we validated generalization of neural encoders to closed-loop dataset which it was not trained on. We additionally found three important design considerations. First, we reasoned that a key limitation of tuning and MLP models is that they do not incorporate history over the inputs, i.e., kinematics. Second, incorporating a delay between hand speed and neural activity improves reproduced PSTHs. Third, regularizing RNN input weights improves generalization toward closed-loop datasets. Overall, our "delayed regularized RNN neural encoder" closely matches kinematic-to-neural relationship and reproduces single neuron PSTHs, neural population dynamics and neural population rotations.

3.4.1 RNN neural encoders better reproduce neural activity

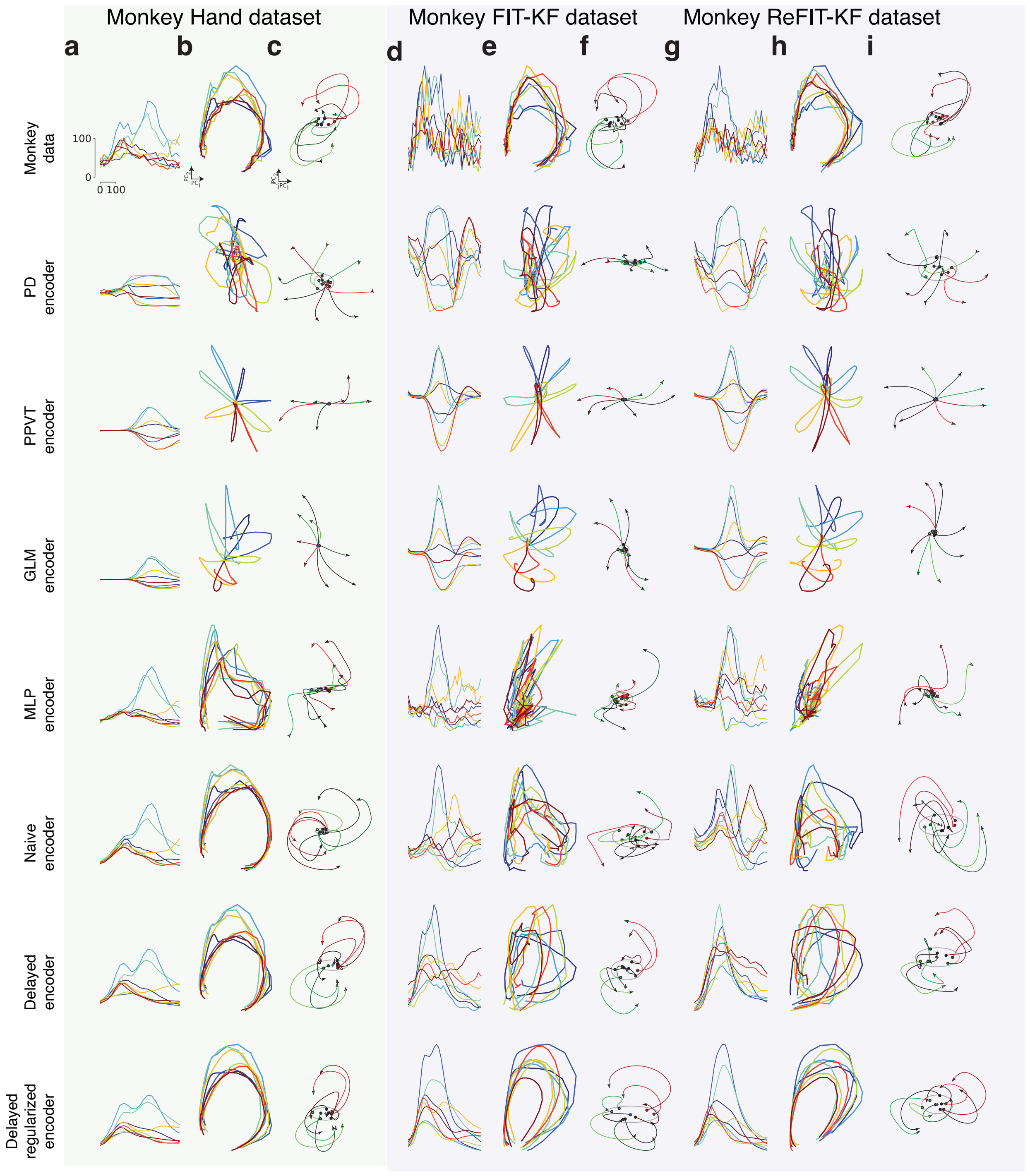

Monkey Hand dataset
Monkey FIT-KF dataset
Monkey ReFIT-KF dataset
a
b
c
d
e
f
g
h
i
Monkey data
PD encoder
PPVT encoder
GLM encoder
MLP encoder
Naive encoder
Delayed encoder
Delayed regularized encoder
100
0
0 100
PC2
PC1
jPC2
jPC1

Figure 3.3: An RNN neural encoder better reproduces neural activity than a PPVT in both training and validation datasets. **(a-c)** Our training dataset was recorded when monkey performed hand-control reaches. **(d-i)** Validation dataset was recorded when monkey performed reaches with FIT-KF and ReFIT-KF decoders. In each dataset, we plot **(a, d, g)** example PSTHs, **(b, e, h)** PCA trajectories in the top 2 PCs, and **(c, f, i)** jPCA trajectories in the top jPC plane. In PSTHs, the vertical bar denotes 100 spike/s, and the horizontal bar denotes 100 ms. We observed that single electrode PSTHs exhibited multiphasic firing pattern and population activity exhibited rotational dynamics in monkey data. A PPVT neural encoder did not reproduce these neural properties. We also trained a naive RNN neural encoder ("Naive encoder"), an RNN neural encoder with a 200ms delay between hand kinematics and neural activity ("Delayed encoder"), and a delayed RNN encoder with L2 regularization on the input weight matrix ("Delayed regularized encoder"). Ultimately, the delayed regularized neural encoder performed the best in reproducing neural activity when validated on BCI datasets. Quantitative results are summarized in Table 3.1.

Table 3.1: Similarity of reproduced neural activity from various neural encoders. Three datasets are included. The "hand" dataset comprises neural data simultaneously recorded with hand reaches. This was the training set for training neural encoders. The "FIT-KF" and "ReFIT" datasets comprise neural data recorded when the monkey controlled the FIT and ReFIT-KF in closed-loop, which we used as validation data. All metrics compare the artificial neural activity to the Monkey's neural activity. PCC stands for Pearson's correlation coefficient. NRMSE stands for normalized root-mean-square error. Higher PCC and lower NRMSE indicate the neural encoder better reproduced neural activity. Bolded numbers indicate the best performance for each dataset.

encoder	dataset	PSTH		PCA		jPCA	
		PCC	NRMSE	PCC	NRMSE	R^2_{skew}	Skew ratio
Monkey J	hand	1	0	1	0	0.33	0.68
	FIT	1	0	1	0	0.29	0.60
	ReFIT	1	0	1	0	0.26	0.58
PD	hand	0.24	16.25	0.28	16.52	<0.01	<0.01
	FIT	0.08	18.41	0.07	19.26	0.01	0.03

	ReFIT	0.12	16.97	0.1	16.43	<0.01	<0.01
PPVT	hand	0.21	16.36	0.3	16.41	<0.01	<0.01
	FIT	0.12	18.01	0.12	18.89	<0.01	<0.01
	ReFIT	0.12	16.69	0.15	15.9	<0.01	<0.01
GLM	hand	0.25	16.05	0.48	13.89	0.01	0.01
	FIT	0.16	17.47	0.25	17.49	0.01	0.01
	ReFIT	0.16	15.68	0.25	15.56	0.03	0.03
MLP	hand	0.79	9.57	0.86	8.25	0.13	0.35
	FIT	0.27	17.58	0.41	17.19	0.05	0.20
	ReFIT	0.15	17.01	0.33	16.06	0.07	0.15
Naive RNN	hand	0.83	7.52	0.94	5.05	0.32	0.46
	FIT	0.43	16.24	0.43	16.83	0.17	0.38
	ReFIT	0.45	14.3	0.42	14.87	0.29	0.46
Delayed RNN	hand	**0.84**	**6.75**	**0.99**	**2.17**	0.32	0.56
	FIT	0.55	14.62	0.61	14.8	0.28	0.57
	ReFIT	0.48	14.46	0.51	14.39	0.27	0.58
Delayed regularized RNN	hand	0.83	7.28	0.98	3.01	0.35	0.55
	FIT	**0.65**	**13.32**	**0.62**	**14.17**	0.37	0.59
	ReFIT	**0.64**	**11.87**	**0.63**	**11.95**	0.29	0.49

We first evaluated how each neural encoder reproduced every electrode's peri-stimulus time histogram (PSTH). The electrode's PSTH is the average firing rate for reaches to each of eight center-out target conditions. This was calculated by averaging empirical firing rates across all trials within a condition, aligned to target onset. We also calculated PSTHs for synthetic spike trains for each encoding model. We subsequently evaluated the similarity between recorded and synthetic neuron PSTHs by calculating the PCC. A model's overall Pearson's r is the average of the Pearson's r across all 192 electrodes for Monkey J.

Figure 3.3 illustrates a representative example, with the recorded PSTHs shown in panel a, d and g. As has been previously described, the PSTHs have a condition-independent increase in activity followed by heterogeneous activity that may be multiphasic [70]. However, consistent with prior work, the PD and PPVT encoders do not capture heterogeneity, including multiphasic behavior, in PSTH activity [45, 46, 52]. We found that the PD model achieved relatively static firing rates This is expected because the PD model's firing rates only vary with reach angle. The PPVT incorporates speed modulation, causing the PSTHs to resemble the speed profile. We next evaluated GLM models, which incorporate position and velocity. We found that GLM model also resemble the speed profile. PD, PPVT, and GLM models achieved comparable performance.

We next evaluated deep-learning-based neural encoders. In contrast to representational tuning and GLM models, we found that deep-learning-based models better reproduced empirically recorded PSTHs. The MLP was capable of reproducing a condition-independent increase and qualitative motifs in the original PSTH, including the relative ordering of condition firing rates. However, it does not capture all multiphasic activity, such as the biphasic activity of the blue PSTH. We reasoned that a key limitation of tuning, GLM, and MLP models is that they do not incorporate history over the inputs, i.e., kinematics. We therefore evaluated the RNN models, incorporating kineamtic history. The RNN can generate different outputs for the same input depending on its internal hidden state, which is a function of past inputs and the RNN's own internal dynamics. We therefore evaluated a naive RNN neural encoder, incorporating a state inference to represent history information. The naive RNN encoder can also reproduce a condition-independent increase and qualitative motifs in the original PSTH but not capture all multiphasic activity.

We further evaluated “delayed RNN encoder” which incorporates a 200 ms delay between hand speed and neural activity found in offline data analysis in Chapter 2.4.1.1. Delayed RNN encoder reproduced biphasic activity in training dataset, however, did not generalize to validation dataset. As discussed in the Methods, we then apply regularization on RNN input weights to force RNN neurons to reduce dependence on external inputs and receive more contribution from internal RNN dynamics (see Chapter 3.4.4. We found this regularization significantly improves reproduced PSTH in validation dataset, such as the PSTH in dark blue in Figure 3.3d and Figure 3.3g.

3.4.2 RNN neural encoders better reproduce population neural activity

As RNN models better reproduce empirical firing rates, we next wondered how well RNN models reproduce neural population activity. To quantify neural population activity, we performed PCA on empirically recorded and synthetically generated PSTHs. When performing PCA on recorded PSTHs, we found 8 dimensions were required to capture over 90% of the recorded neural variance. Tuning models were overly simplistic, demonstrating a smaller dimensionality. When performing PCA on synthetic neural population activity generated by the PD or PPVT outputs, we found that only 2 PCs are required to capture nearly 100% variance. On the other hand, we found that RNN models required 4 or 5 dimensions to capture over 90% of the neural variance, as shown in Figure 3.4. Together, these results demonstrate that RNN models more closely match the dimensionality of real data, reflecting relatively greater variability in the neural population response compared to tuning and linear models.

How similar are neural population trajectories in these low-dimensions? We compared low-dimensional projections via PCA on the synthetic versus recorded neural population. We projected synthetic and neural activity onto the PCs found by performing PCA on the real data. We found that tuning models had very different projections in comparison to real data (compare Figure 3.3b,e, and h). PPVT low-dimensional variance primarily resided on a 2-dimensional axis, reflecting hand speed and moving direction. On the other hand, MLP and RNN models reproduced circling neural trajectories as in real data, achieving PCCs above 80%. Critically, RNN input weight regularization leads to better inference of RNN dynamics that generalizes validation dataset to reproduce the neural population dynamics.

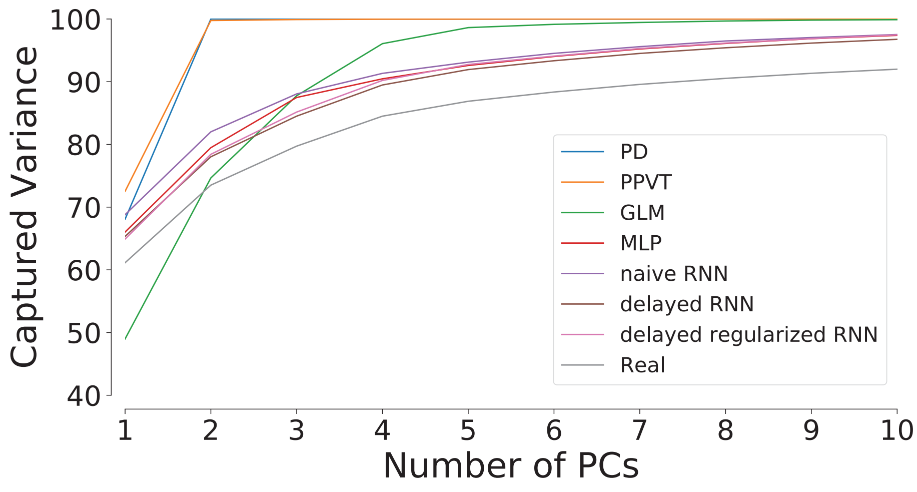


Figure 3.4: Dimensionality of recorded and synthetic PSTHs. 8 dimensions were required to capture over 90 % of the recorded neural variance.

This demonstrates that RNN models better reproduced empirical neural population motifs.

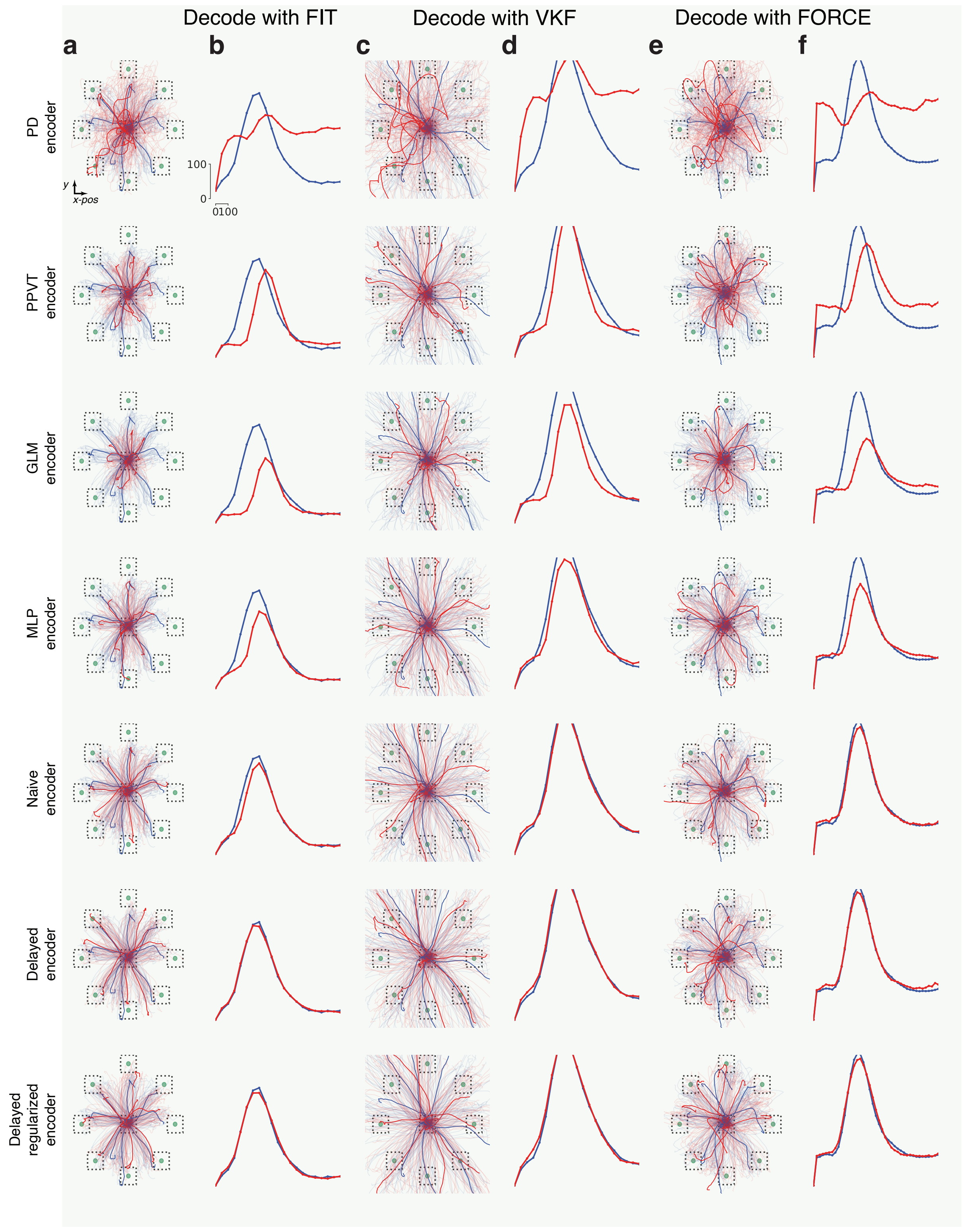
Another way to measure neural population structure is to compare rotational dynamics in the neural population via jPCA(e.g., [46, 2, 45, 71], see Methods). We applied jPCA and found that PD- or PPVT-synthesized activity cannot be well described by rotational dynamics, as reported previously [46, 45]. We found that the degree of rotational dynamics in the encoding model output increased as the model was able to consider historical inputs; the RNN models exhibited more rotational dynamics than the tuning models. We quantify these results with the mean PCC and NRMSE of PC trajectories in the top 8 dimensions (capturing greater than 90% of the variance) and R^2_{skew} and skew ratio in jPCs between synthetic and real neural activity and summarized in Table 3.1.

These results demonstrate that RNN models can better reproduce single electrode PSTHs, and also better reproduce neural population motifs. Together, the data generated by RNN models appears to be a more faithful representation of neural population activity in the motor cortex.

3.4.3 Kinematics decoded from RNN generated neural activity better match decoded kinematics

As our motivating application for generating motor cortical neural signals is BCI simulation, we next assessed if RNN synthetic neural data could better match offline decoding of previously recorded neural data. We therefore decoded neural data recorded from when monkeys performed a center-out-and-back reaching task, and compared these to kinematics decoded from synthetic neural binned spike counts. A more effective neural encoder will produce decodes that more closely resemble decoding real data. While RNN neural encoder more accurately recapitulate PSTHs and population structure, as earlier described, optimal decoding dimensions may have little overlap with the top PCs of the activity [72]. Therefore, it is important to assess if neural variance in these kinematic dimensions is better captured by RNN models. We trained a FIT-KF [9], a VKF [22, 73], and a FORCE [16] decoder to decode both recorded and synthetic neural activity. We show randomly chosen decoding trials in Figure 3.5a,c, and e. The decoded trajectory from PD- and PPVT-synthesized neural activity was not closely matched. For example, FORCE decoder requires neural dynamics which PPVT cannot provide, resulting in strong-biased decoded trajectory shown in Figure 3.5e. In contrast, across all decoders, RNN neural encoders reproduced neural activity that were sufficient to be decoded into cursor kinematics which matched to the one decoded from monkey data, resulting in lower NRMSE of cursor velocities. As our prior results demonstrated that RNN models incorporating kinematic history better reproduce dynamical aspects of motor cortical activity, this result is consistent with the observation that dynamical aspects of motor cortical activity are important for kinematic decoding [74, 75, 53]. Hence, RNN models well encode kinematics in a manner that is more similar to the real neural data than PPVT encoders. Note, while RNN models achieved better training loss and better reproduced PSTHs and population motifs, we did not apply objectives on matching decodes.

As discussed earlier, BCIs are closed-loop systems where the user interacts with the



Decode with FIT
Decode with VKF
Decode with FORCE
a
b
c
d
e
f
PD encoder
PPVT encoder
GLM encoder
MLP encoder
Naive encoder
Delayed encoder
Delayed regularized encoder
y
x-pos
100
0
0100

Figure 3.5: Decoding open-loop neural data and synthetic neural activity. We used neural encoders to generate synthetic neural activity while a monkey performed the center-out-and-back task with the hand. We then compared the decoded trajectories from real neural activity (blue) to synthetic activity (red). **(a, c, e)** We plot multiple decoded center-out trajectories, with a random trajectory bolded for each direction. **(b, d, f)** We also show average cursor speed profile with the vertical bars denote 100 mm/s, and the horizontal bar denotes 100 ms. Decoded kinematics from real and synthetic data are marked in blue and red, respectively. PPVT synthetic neural activity performed especially poorly under FORCE decoding, exhibiting a bias in the up-left direction. In contrast, RNN neural encoders reproduced real neural data decodes more closely. Quantitative results are summarized in Table 3.2.

Table 3.2: NRMSE of cursor velocities decoded from real monkey data and from synthetic data in the training dataset (open-loop hand control). Lower NRMSE indicates the neural encoder better reproduced neural activity with respect to the decoders.

Encoder / Decoder	PD	PPVT	GLM	MLP	Naive RNN	Delayed RNN	Delayed regularized RNN
FIT-KF	171.73	114.24	105.39	87.44	77.21	**72.48**	73.78
VKF	166.35	104.38	93.65	84.81	**81.17**	83.29	83.67
FORCE	167.5	119.72	105.49	91.24	**84.42**	86.37	88.69

decoded output, updating his or her motor commands in response to visual feedback of the decoded output. Hence, the statistics of closed-loop neural population activity may differ from those during naturalistic reaching (e.g., [20, 19, 36, 76]). We therefore evaluated how well RNN models generalized to reproduce closed-loop data, where monkeys update their motor commands in response to imperfect decoding. If RNN models more faithfully reproduce prior decoded closed-loop kinematics, this is an additional evidence that they are more appropriate for closed-loop BCI use.

Neural encoders were trained on a dataset recorded when monkey performed open-loop center-out-and-back task with the native hand. Critically, we subsequently tested how well these encoding models could generate neural activity during BCI control from a previously collected closed-loop BCI dataset. We used the kinematics of the monkey's arm to generate synthetic neural activity, and subsequently decoded this activity post-hoc using the corre-

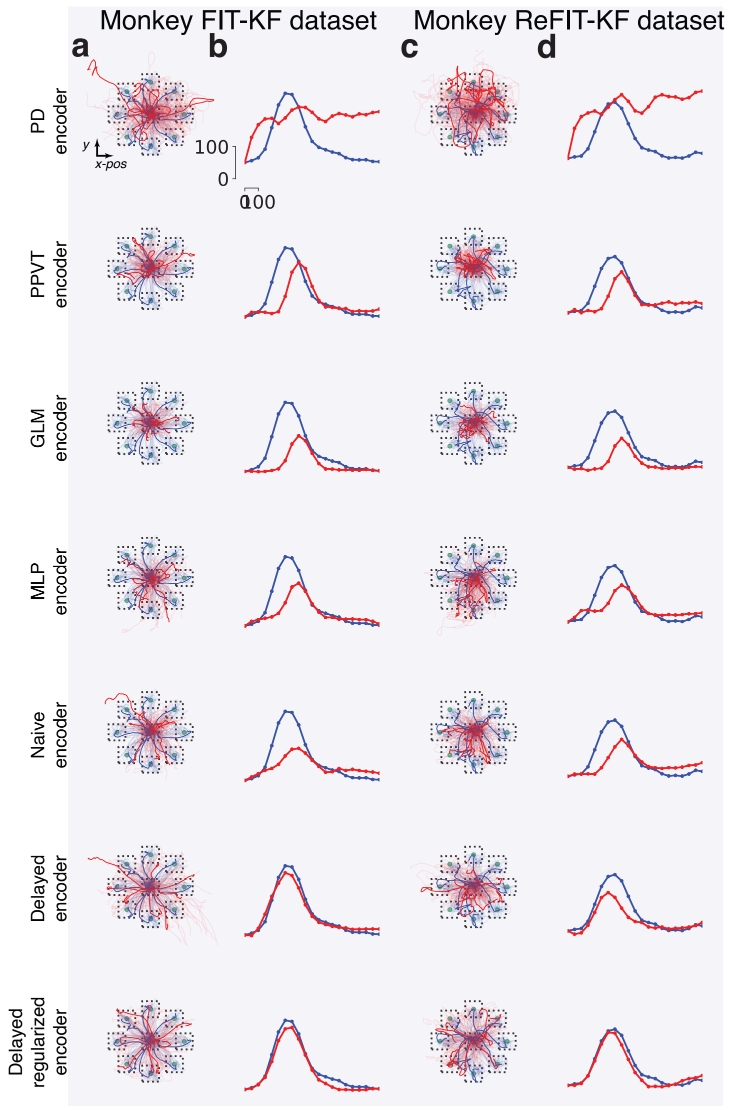


Figure 3.6: Decoding closed-loop neural data and synthetic neural activity. We used neural encoders to generate synthetic neural activity while a monkey controlled the FIT-KF and ReFIT-KF decoders. We then compared the decoded trajectories from synthetic activity (red) to the real trajectories decoded from closed-loop neural activity (blue). We found that incorporating a delay between kinematics and neural activity and regularization on RNN input weights significantly improved generalization in reproducing decoded closed-loop kinematics, which are validation datasets the neural encoder was not trained on. Quantitative results are summarized in Table 3.3.

Table 3.3: NRMSE of cursor velocities decoded from real monkey data and from synthetic data in the validation datasets (closed-loop FIT-KF and ReFIT-KF control). Lower NRMSE indicates the neural encoder better reproduced neural activity with respect to the decoders.

Encoder / Decoder	PD	PPVT	GLM	MLP	Naive RNN	Delayed RNN	Delayed regularized RNN
FIT-KF	159.56	102.55	95.86	106.14	103.92	93.69	**79.09**
ReFIT-KF	208.77	112.65	102.71	124.03	111.02	108.45	**96.45**

sponding FIT-KF and ReFIT-KF decoders used in online experiments. We then compared the post-hoc decoded movements to the empirical movements during closed-loop experiments. These results are summarized in Figure 3.6, demonstrating that even for closed-loop control, the RNN-synthesized activity could be better decoded to produce kinematics than PPVT-synthesized. Together, these results indicate that RNN models reproduce neural population activity better than PPVT when used in closed-loop settings, even though they were only trained in open-loop settings.

3.4.4 Regularize RNN input weights to receive more contribution from internal RNN dynamics

Prior studies have shown that motor cortex does not represent movement, bat rather a dynamical system [45, 46]. We therefore regularize RNN input weights to force RNN neural encoder to rely more on intrinsic recurrent states than external inputs. We analyze how the regularization impacted the RNN dynamics by quantifying the relative importance of external inputs (i.e. hand kinematics) and intrinsic recurrent states. For each recurrent neuron, we examined their contribution as the product of the weight with the activity of the signal for both external inputs and intrinsic recurrent states (see Methods). We computed the ratio between the norm of the local recurrent contribution and the external inputs for each individual RNN neuron. The resulting average contribution ratio was smaller than one indicating the recurrent contribution is larger than the contribution from external inputs in

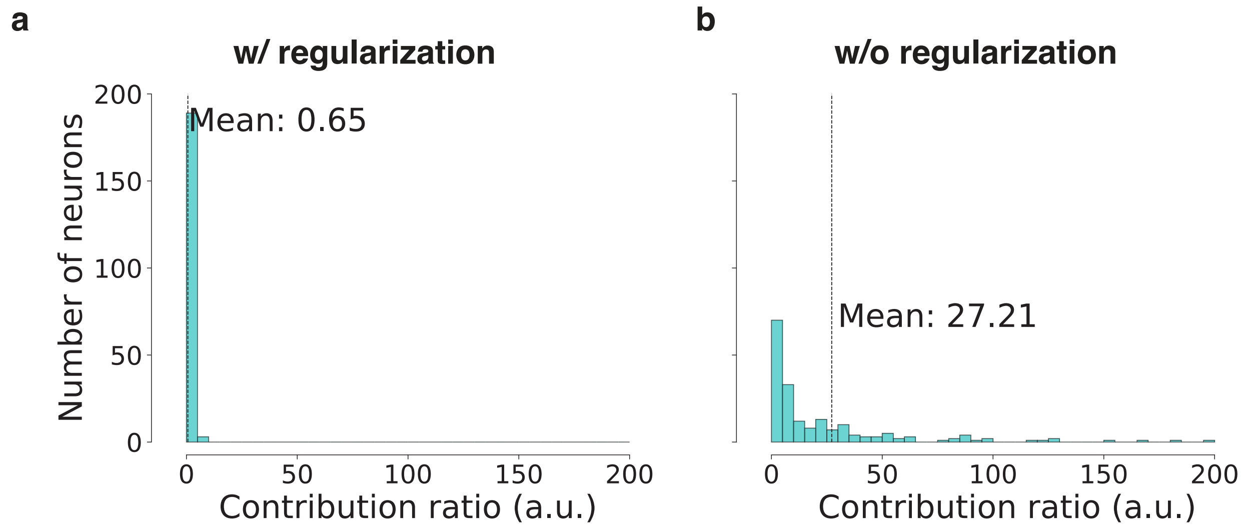


Figure 3.7: Regularizing RNN input weights leads to a stronger contribution from internal RNN dynamics than external inputs.**(a)** Histogram of the contribution ratio (see Methods) in the regularized RNN neural encoder. A lower value indicates a lower contribution from the RNN inputs and a stronger contribution from RNN dynamics. **(a)** Histogram of contribution ratio in a non-regularized RNN neural encoder. This encoder had a significantly higher mean contribution ratio.

most of neurons. In contrast, naive RNN neural encoder had a wide distribution of contribution ratio and high average ratio. Thus, as expected, RNN input weight regularization forces recurrent states to have a substantial impact in RNN dynamics.

3.5 Discussion

Prior work has demonstrated that motor cortical activity is heterogeneous, dynamic, and more complex than previously thought [52, 77, 45, 78]. Neural encoders that do not reproduce neural population statistics will limit the performance of BCI simulators. To overcome this limit, the key innovation was using a high capacity recurrent neural network, and incorporating a delay between hand speed and neural activity, and regularizing on RNN input weights to closely model kinematic-to-neural relationship in the empirical data. Our results demonstrate that these models are better than traditional models in (1) reproducing heterogeneity in single electrode PSTHs, (2) reproducing nonlinear structure and dynamic features in the neural population, and (3) matching decoded kinematics.

We found that PPVT model significantly underestimated the variability of motor cortical neural activity. The PPVT incorporated kinematics, causing the PSTHs to have kinematic-resembling activity. The RNN neural encoder, by implementing a dynamical system, can learn longer temporal dependencies. In contrast, RNN models incorporating history of inputs and nonlinearity more closely reproduced empirical PSTHs. In addition to this, they also reproduced neural population motifs, resembling neural trajectories in PCA and jPCA projections. We also found that decoding RNN generated neural activity more closely resembled kinematics decoded from recorded activity, both in open-loop and closed-loop datasets.

An important limitation of our BCI simulator approach is that if a novel decoder algorithm incorporates newly discovered mechanisms or aspects of neural variance not captured by the neural encoder, then the simulator may not adequately evaluate this decoder's performance. As described previously, FORCE decoder requires neural dynamics which PPVT cannot provide. This limits the scope of testable decoding algorithms to ones that operate on neural variability adequately captured by the neural encoders. In addition, the input to RNN neural encoders were 2D position and velocity. Motor cortex also encodes additional kinematic [48] and kinetic variables including acceleration [73, 79], time [70], distance [80, 81, 82, 83, 84], joint angles [85], and forces [86, 87]. These additional kinematic parameters may improve neural encoders. Future work may improve encoding models by measuring and incorporating additional kinematic and kinetic variables as inputs.

To demonstrate that RNN models generalize for BCI simulators, we also assessed how well these models generated neural activity during closed-loop BCI experiments. Closed-loop BCI data has different statistics than open-loop data. In closed-loop BCI decoding, the user's kinematics are drastically different; where as offline data comprises primarily ballistic reaches, closed-loop BCI decoding incorporates corrective movements, which are typically smaller in extent and fairly precise. However, if an neural encoder generally captures how kinematics can be predictive of neural population activity, we would expect this neural encoder to more accurately reproduce closed-loop BCI decoder performance. We found that, when generalizing to closed-loop BCI kinematics, RNN neural encoders significantly outperformed PPVT neural encoder, indicating that they may generalize better for closed-loop experiments. We will assess how these RNN neural encoders perform in real-time closed-loop experiments in chapter 4 and chapter 5 These experiments would utilize the neural encoder to generate synthetic neural activity and decode this activity in real-time.

There is a growing literature using deep learning in neuroscience. This body of work includes using variational autoencoders to denoise intracortical spike trains [53], and model early visual representations [88], sensory cortex [62], decision-making tasks [89, 90, 91], and motor tasks [92, 78, 46]. Prior studies training RNNs to perform motor tasks have used artificial units that resemble neurophysiological activity and dynamics [92, 78, 46]. However, these studies have important differences to our RNN neural encoders. The goal of those studies is to train RNNs that reproduce animal behavior, so that the artificial RNN can be further analyzed for neuroscientific insight [90]. Along these lines, the inputs to these RNNs are task inputs and the outputs are behavior (such as electromyograms or kinematics). In contrast, our study aimed to generate neural activity as a function of kinematics. As such, the inputs to our network are kinematics (as opposed to task inputs), and the outputs are neural activity (as opposed to EMG or kinematics). Finally, our work is not concerned about the activity of the hidden artificial units, as our goal is to reproduce neural activity at the output.

3.6 Conclusion

Deep-learning-based neural encoders are able to reproduce heterogeneous neural activity better than conventional tuning models. We showed that RNN neural encoders are significantly better than prior PD and PPVT encoders in reproducing PSTHs, neural population motifs, and matching decoded kinematics in open-loop and closed-loop datasets. Our results indicate that RNN neural encoders may significantly improve the fidelity of BCI simulators, which we will show in Chapter 4 and Chapter 5.

CHAPTER 4

BCI emulator

4.1 Summary

We build on a prior emulator in Cunningham et al. [2] that correctly optimized decoder bin width by generating synthetic neural spike counts from hand kinematics using a tuning model, PPVT. As shown in chapter 3, a limitation of prior study is that a tuning model is insufficient to reproduce complexities in neural firing rates and population activity, including multiphasic PSTHs, neural trajectories, and neural dynamics. To address this, we used neural network based encoder to transform hand kinematics to synthetic neural activity in our emulator. We evaluated our emulator by performing three published studies and quantitatively compared the emulator's predictions to the monkey empirical results. We chose these three studies to test: linear decoders (FIT-KF and VKF), a two-stage trained decoder (ReFIT-KF), and a nonlinear decoder (FORCE). Our emulator correctly reproduced the conclusions of these studies, in addition to reproducing precise details of control, including: (1) distance-to-target profiles, closely matching the first touch time (FTT) and the dial-in time (DIT), (2) the distribution of trial times, and (3) cursor trajectories observed in prior monkey online experiments. These results suggest it is feasible to accurately predict BCI performance without neurosurgery, enabling quantitative comparisons between different types of decoding algorithms. We anticipate this system can facilitate and accelerate the development of BCI decoders.

4.2 Introduction

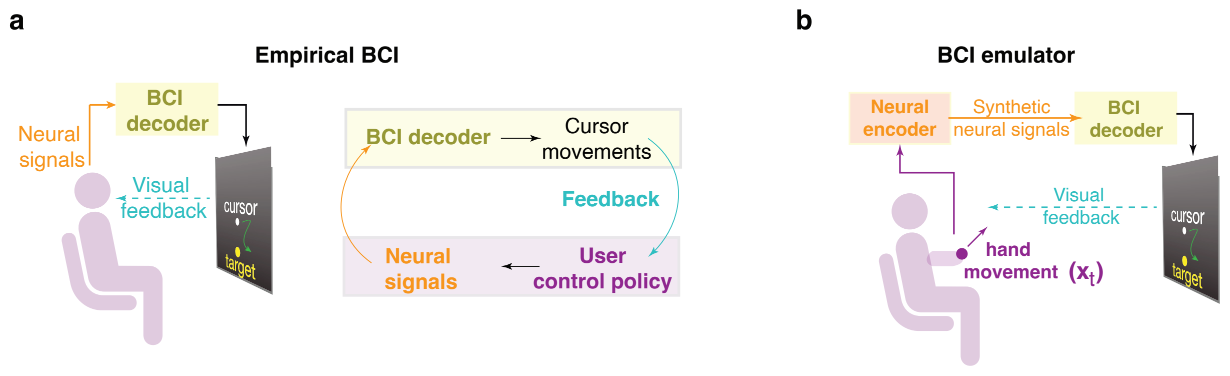


Figure 4.1: Empirical BCIs are closed-loop systems and BCI emulator can circumvent neurosurgery. **(a)** BCI decoders are imperfect, so decoded cursor movements will not fully match the intent of the user. In response, the user will generate updated motor commands and neural signals. **(b)** The BCI emulator circumvents invasive neurosurgery by generating synthetic neural signals with hand kinematics.

Brain-computer interfaces (BCIs) aim to provide naturalistic communication and movement for those with paralysis by decoding neural spikes into actions. BCIs require closed-loop in vivo experiments to develop, design, optimize, and benchmark decoder algorithms (Figure 4.1a). Although offline evaluation provides insight into what algorithms are promising, the discrepancy between offline and online performance may lead to incorrect conclusions that mislead algorithm design. To accelerate BCI research, prior studies have attempted to emulate or simulate BCI systems to accurately characterizes online decoder performance without neurosurgery. An important component is neural encoder, which maps motor commands (recorded hand kinematics) to synthetic neural activity. Cunningham et al. [2] developed a BCI "emulator" where a human subject's hand movements generate synthetic neural activity to control a BCI decoder, removing invasive neurosurgery but still including true control policy from testing subjects (Figure 4.1b). We use the term "emulator" to highlight that this system requires a physical experiment with hardware that mimics the BCI decoding system and a human to provide a control policy.

The prior study by Cunningham and colleagues used a Poisson process velocity tuning (PPVT) model of neural activity , which they found resulted in successful optimization of decoder bin width. However, a PPVT neural encoder fails to capture the rich heterogeneity and dynamics of motor cortical activity [3] (see chapter 3). Also, we performed BCI emulator experiments using the PPVT neural encoder, and found it failed to reproduce published studies, necessitating a better neural encoder. Further details of the BCI emulator are provided in the Methods, and all experiments were approved by the UCLA IRB.

Our goal was to make a general neural encoder that would work well in all decoder settings. We therefore hypothesized that a neural encoder that reproduces the heterogeneity, low-dimensionality, and dynamics of motor cortical neural populations during natural reaches would be capable of reproducing the three published decoder studies. We trained RNN neural encoders to transform hand kinematic inputs to binned spike outputs using data collected from two Utah arrays while a monkey performed center-out-and-back reaches with their hand (details in chapter 3). These encoders better reproduced single neuron PSTHs, neural population trajectories, and neural population rotational dynamics in point-to-point reaches. RNN neural encoders also reproduced neural activity that could be accurately decoded. Together, RNN-based neural encoders reproduced key features of motor cortical neural population activity, providing a better approximation of neural activity than the PPVT.

Although we have shown RNN neural encoders closely match kinematic-to-neural relationship, it is still unclear if these neural encoder can reproduce decoder performance in prior studies. Here, we employs emulator closed-loop experiments to validate the neural encoders. A user moves his or her hand, which generates synthetic activity that is subsequently decoded. Because the user sees the decoded output and adjusts his or her motor commands to better control the decoder, the BCI emulator incorporates a true (not approximated) user control policy. This enables us to therefore evaluate a neural encoder in isolation.

We performed the FIT-KF, ReFIT-KF, and FORCE studies in a BCI emulator using 4 subjects. For each study, we reproduced the task, task parameters, decoder hyper-parameters, and decoder training as described in the published studies [9, 8, 16]. We trained decoders as in the published studies by having subjects perform center-out reaches and performing supervised training with concurrent kinematics and synthetic neural activity generated by the neural encoder (see Methods). In each study, the task was a variant of the center-out-and-back task, with targets 8 cm or 12 cm away from a center target, a 500 ms hold time to successfully acquire each target, and 4 cm or 6 cm acceptance window sizes (see Methods).

4.3 Methods

4.3.1 Emulator setup

We built a BCI emulator in a similar fashion to the "Online Prosthetic Simulator" introduced by Cunningham et al. [2]. In our BCI emulator, endpoint kinematics of the user's hand were recorded via the Polaris Vega system (Northern Digital, Ontario, Canada). This system tracked the position of an infrared bead to a resolution of 0.30 mm at 250 Hz, enabling precise estimation of hand position. These data were processed by a custom real-time system build using the LiCoRICE framework [93], which used 1 ms clock cycles, enabling simulation of spikes-based BCIs. Our real-time system processed hand kinematics and transformed them into spike counts via the neural encoder. The synthetic neural spike counts were decoded to update the cursor positions shown on a screen. In response to this visual feedback, the user made new motor commands to best control the BCI.

The three published studies we used to validate our emulator used variants of the center-out-and-back task and the pinball task (see chapter 2. The center-out-and-back task was used to collect training data for PVKF, ReFIT-KF, FORCE, and VKF decoders. The pinball task was used to collect training data for FIT-KF decoder. All studies used the center-out-and-back tasks to quantify decoder performance. The center-out-and-back task parameters for each study are summarized in Table 2.1. Note, in each of our experiments, we matched the number of electrodes used by the BCI experiments that we compare. FIT-KF, ReFIT-KF, PVKF, and VKF were using 192 electrods, and FORCE was using 96 electrodes.

4.3.2 Experiment procedure

In experiments, we had each subject perform approximately 50 native arm reaches to gain familiarity with the system. We then collected training data for 500 native arm reaches on center-out-and-back and pinball tasks, respectively. This data was used to train decoders, including the FIT-KF, FORCE, VKF and PVKF decoders. To train the ReFIT-KF, we collected 500 reaches in PVKF-controlled center-out-and-back task. Testing subjects practiced decoders in a sequence (arm, FIT, ReFIT, FORCE, then VKF) until they felt confident in controlling decoders. Following this, we performed data collection for decoder performance. In the evaluation session, we follow the same sequence (arm, FIT, ReFIT, FORCE, then VKF), where we collected 60 center-out-and-back reaches for each decoder to analyze. Subjects were notified they could stop the experiment when they felt fatigue. We repeated this sequence until the subject terminated the experiment due to fatigue.

4.4 Results

4.4.1 Emulator with PPVT neural encoder leads to incorrect conclusions.

The prior study by Cunningham and colleagues used a Poisson process velocity tuning (PPVT) model of neural activity, which they found resulted in successful optimization of decoder bin width. Importantly, an empirical result they and another study confirmed [10]. PPVT neural encoder does not reproduce important neural properties in offline analysis. We are still curious how decoders perform in emulator when run with PPVT neural encoder. We intend to reproduce three representative prior studies: linear decoders (FIT-KF and VKF),

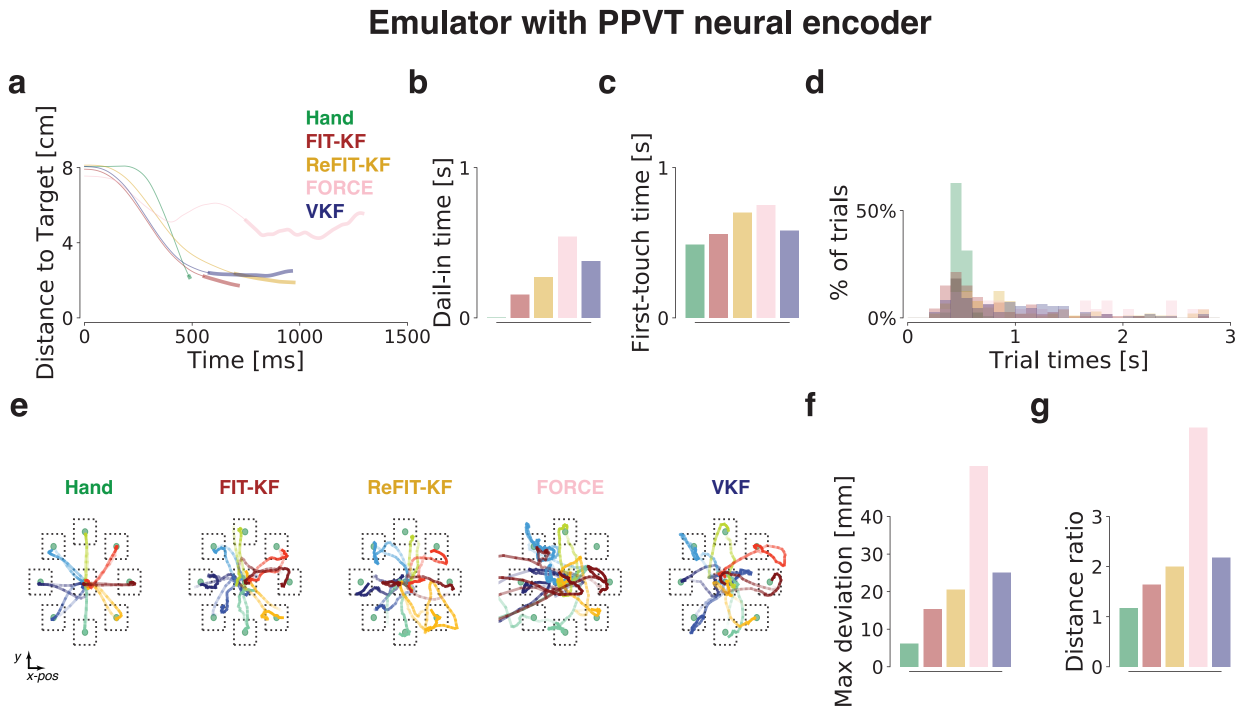


Figure 4.2: An emulator with a PPVT neural encoder does not reproduce prior published studies. **(a)** Distance-to-target plots from emulator experiments for the various decoders. Emulator results were shifted 200 ms to remove visuomotor response time when using the neural encoder. The bolded line corresponds to DIT. **(b, c)** Average DIT and FTT, respectively. **(d)** Distribution of trial times for each decoder. **(e)** Randomly sampled trajectories for each decoder. **(f)** Max deviation from the straight line path for each decoder. **(g)** Distance ratio (cursor path length divided by straight path length) for each decoder. A PPVT neural encoder does not reproduce published studies, especially for the nonlinear FORCE decoder. FORCE was sometimes not controllable and had a strong bias toward left shown in **(e)**, resulting in low success rate (50%). FIT-KF, ReFIT-KF, and VKF all had similar performances, also not consistent with prior studies. Results are summarized in Table 4.1.

Table 4.1: Decoder performance in emulator with PPVT neural encoder. Emulator results were shifted by 200 ms to remove visuomotor response time when using the neural encoder. TT, FTT, and DIT are trial time, first-touch time, and dial-in time, respectively.

Decoder	TT [ms]	FTT [ms]	DIT [ms]	distance ratio	max deviation [mm]	success rate	# of center-out trials
hand	488	486	2	1.17	6.15	1.00	118
FIT-KF	712.97+200	556.78+200	156.19	1.64	15.42	0.97	117
ReFIT-KF	971.90+200	701.71 +200	270.19	2.00	20.57	0.94	111
FORCE	1292.00+200	750.00+200	542.00	4.78	53.49	0.51	49
VKF	959.98+200	580.62+200	379.36	2.18	25.08	0.96	114

a two-stage trained decoder (ReFIT-KF), and a nonlinear decoder (FORCE).

First, as expected, VKF was controllable and performed well in a center-out-and-back task which is consistent with prior study [2]. However, it does not enable us to have comparison across decoders. The ballistic control of ReFIT-KF is the best,resulting in the short first-touch time in empirical result. However, this emulator leads to incorrect conclusion that FIT-KF and VKF have better ballistic control than ReFIT-KF.

RNN neural encoders reproduced neural activity that were sufficient to be decoded into cursor trajectory which matched to the one decoded from monkey data. In contrast, the decoded trajectory from PPVT-synthesized neural activity was not closely matched.

Further, FORCE decoder requires neural dynamics which PPVT cannot provide, resulting in strong-biased decoded trajectory shown in Figure 4.2e. FORCE decoder was basically uncontrollable and achieved low success rate. Therefore, emulator with PPVT neural encoder cannot be used to evaluate nonlinear decoders, and even just for linear decoders, the conclusion is incorrect. found it failed to reproduce published studies

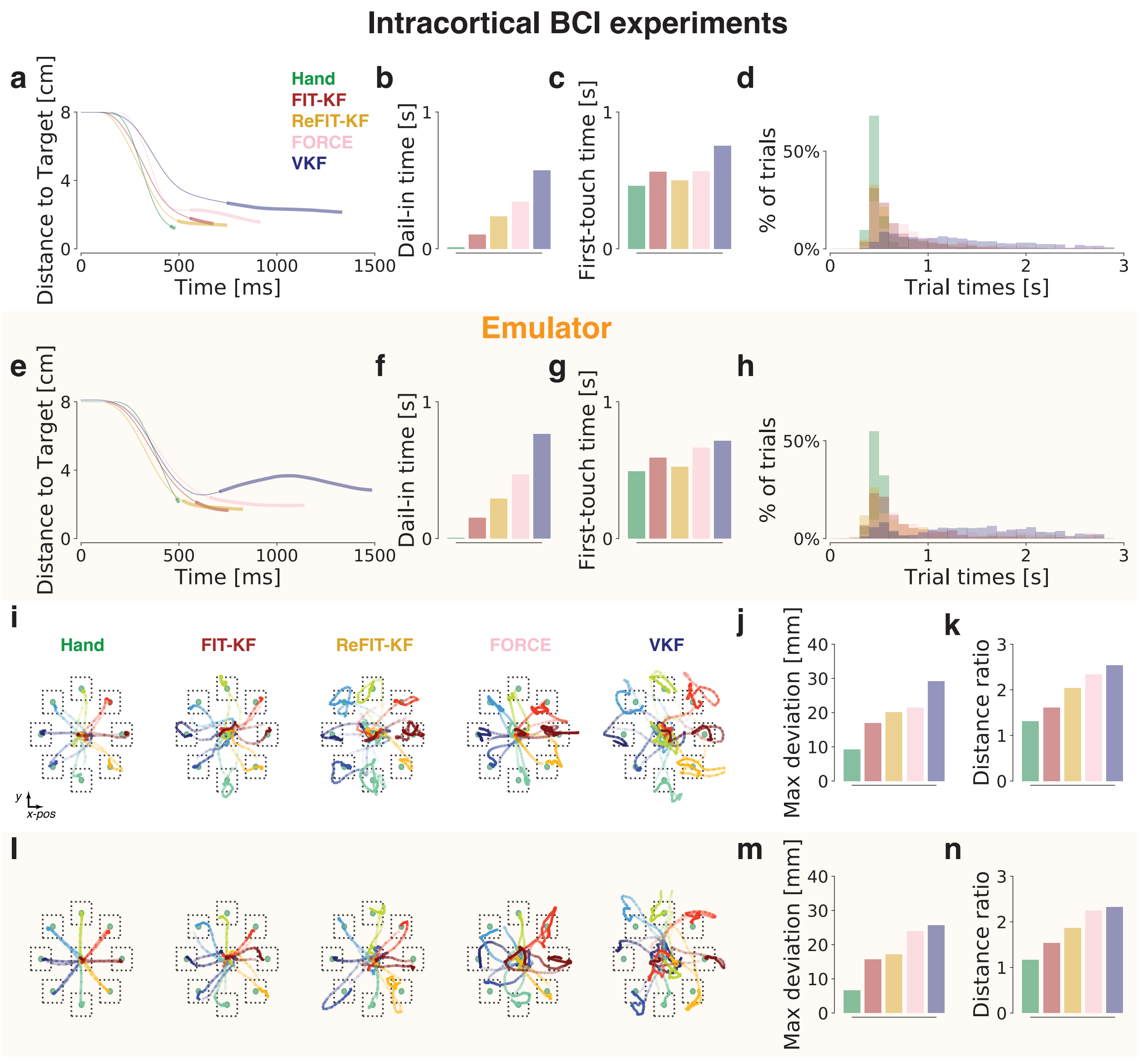


Figure 4.3: BCI emulator reproduces published studies. **(a-d, i-k)** Reproduction of the same panels from Figure 2.6, for ease of comparison of the BCI emulator to intracortical BCI experiments. **(e-h)** Same as **(a-d)** but for the BCI emulator. **(l-n)** Same as **(i-k)** but for the BCI emulator. BCI and emulator results are summarized in Table 2.2 and Table 4.2, respectively. Emulator results were shifted by 200 ms to remove visuomotor response time when using the neural encoder.

Table 4.2: Decoder performance in emulator with the delayed regularized RNN neural encoder. Emulator results were shifted by 200 ms to remove visuomotor response time when using the neural encoder. TT, FTT, and DIT are trial time, first-touch time, and dial-in time, respectively.

Decoder	TT [ms]	FTT [ms]	DIT [ms]	distance ratio	max deviation [mm]	success rate	# of center-out trials
hand	496.1	491.45	4.64	1.16	6.55	1.00	602
FIT-KF	745.19 +200	592.17 +200	153.02	1.53	15.59	1.00	581
ReFIT-KF	817.75 +200	525.28 +200	292.47	1.86	17.16	0.99	584
FORCE	1132.17 +200	664.07 +200	468.10	2.24	23.87	0.93	576
VKF	1477.72 +200	713.55 +200	764.17	2.32	25.65	0.90	551

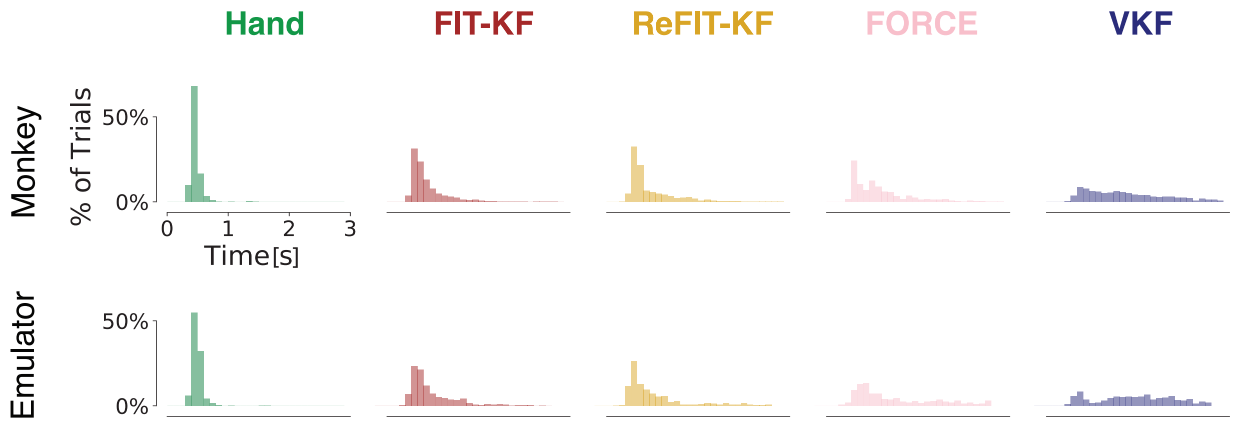


Figure 4.4: The emulator reproduce trial time distributions observed in prior studies. Emulator results in decoding cases were shifted 200 ms to remove visuomotor response time. We observed similar trial time distributions across decoders in comparison to the closed-loop BCI data. FIT-KF and ReFIT-KF had narrower distributions while FORCE and VKF had relatively wider distributions.

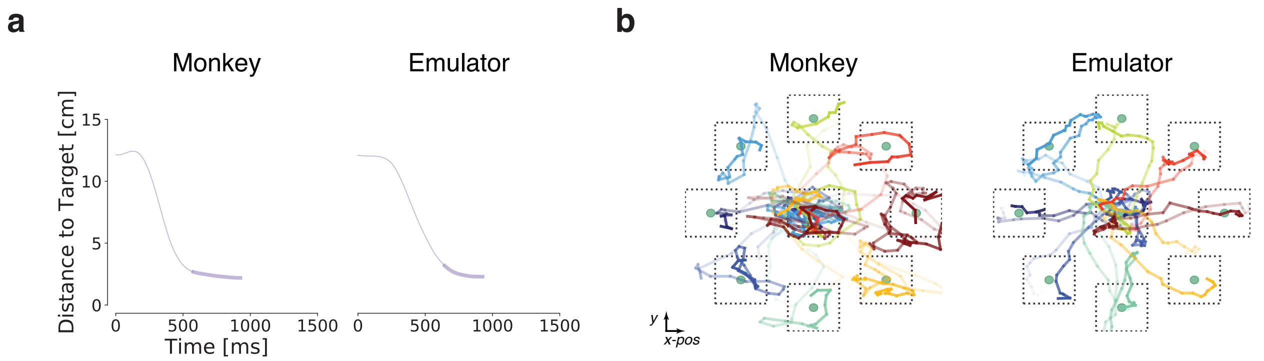


Figure 4.5: Emulator reproduce PVKF decoder performance. **(a)** Distance-to-target plots from BCI emulator experiments. The distance-to-target profiles in emulator qualitatively resembled those in monkey experiments. **(b)** Randomly sampled decoded trajectories for each direction in each experiment. The emulator reproduce jittery and jumpy cursor trajectories during PVKF control. Quantitative results are summarized in Table 4.3. Emulator results were shifted by 200 ms to remove visuomotor response time when using the neural encoder.

Table 4.3: PVKF performance in monkey, emulator, and simulator experiments. The delayed regularized RNN neural encoder was used in both emulator and simulator experiments. Emulator results were shifted by 200 ms to remove visuomotor response time when using the neural encoder. TT, FTT, and DIT are trial time, first-touch time, and dial-in time, respectively.

Exp.	TT [ms]	FTT [ms]	DIT [ms]	distance ratio	max deviation [mm]	success rate	# of center-out trials
Monkey	925.87	576.32	349.55	2.72	35.77	0.99	1011.0
Emulator	925.99 +200	644.16 + 200	281.84	2.11	28.42	1.00	683.0
Simulator	993.40	566.17	427.23	3.30	37.10	0.95	248.0

4.4.2 A BCI emulator using a RNN neural encoder reproduces published BCI studies.

PPVT neural encoder fails to capture the rich heterogeneity and dynamics of motor cortical activity, resulting in incorrect conclusions in emulator experiments. RNN neural encoders reproduced neural activity that were sufficient to be decoded into cursor trajectory which matched to the one decoded from monkey data. The results of the published studies are combined in Figure 4.3a-d, i-k. In brief, each study reported superior performance over the then state-of-the-art VKF, due to superior fine control to successfully hold the cursor over the target (Dial-In Time or DIT, Figure 4.3a,b, for further explanation of DIT, see Figure 2.2a) and faster time to first touch the target (First Touch Time or FTT, Figure 4.3a,c, for further explanation of FTT see Figure 2.2b). This also led to smoother kinematic trajectories, with improved path efficiency and less deviation from the straight-line path (Figures 4.3i-k, see Methods).

Our BCI emulator, using an RNN neural encoder, reproduced the key conclusions of each study. Remarkably, it also produced detailed timing and trajectory results across all studies. The summary of our decoder comparisons are in Figure 4.3e, where we reproduced the distance-to-target plot results reported in all studies. In BCI emulator experiments, we highlight the following: (1) we reproduced that FIT-KF, ReFIT-KF, and FORCE all achieved superior performance to the VKF, as well as the result in Fan et al. [9] that FIT-KF and ReFIT-KF achieve similar trial times with ReFIT-KF having faster FTT than FIT-KF, but FIT-KF having better DIT than ReFIT-KF (Figure 4.3a,e); (2) we reproduced the ordering of fine control DIT across studies, in particular that FIT-KF had the shortest DIT, followed by ReFIT-KF, FORCE, and VKF (Figure 4.3b,f); (3) we reproduced the trends in FTT across studies, where VKF had the slowest FTT, ReFIT-KF had the best FTT, and FIT-KF and FORCE are in between. (Figure 4.3c,g); (4) we achieved similar trial time distributions over single-trials across all decoders, with the VKF achieving the widest trial-time distribution (Figure 4.3d,h and Figure 4.4 for one-by-one comparison); (5) we observed qualitatively similar decoder trajectories (Figure 4.3i, l) and reproduced the decoder ordering of max deviation and distance ratio (Figure 4.3j,k,m,n); and (6) we observed the intermediate PVKF training stage of the ReFIT-KF also closely matched PVKF performance from intracortical experiments (Figure 4.5). Together, these results show that our BCI emulator with an RNN neural encoder reproduced the key timing and kinematic performance statistics of decoders evaluated in intracortical experiments, including timing at the single-trial resolution.

4.5 Discussion

Our goal was to make a general neural encoder that would work well in all decoder settings. We found that a BCI emulator, using a neural encoder that models the kinematic-to-neural relationship closely, reproduced the detailed timing and decoder's trajectory kinematic in online BCI experiments. We emphasize that we did not use any decoder-specific information in the design of the neural encoder. The neural encoder was only trained from open-loop center-out-and-back reaches. Nevertheless, when used in BCI emulation for significantly different studies, we observed this approximate neural data was sufficient for predicting BCI decoder performance.

We emphasize that these studies vary significantly; for example, ReFIT-KF emulation required an intermediate stage of decoding with a PVKF using the neural encoder, followed by subsequent retraining, where as the FORCE and FIT-KF were nonlinear and linear decoders, respectively, trained directly from hand movements. We found that the intermediate PVKF training stage of the ReFIT-KF also closely matched PVKF performance from intracortical experiments. By modeling the kinematic-to-neural relationship as closely as possible, we were able to accurately recapitulate detailed timing and kinematic performance of these decoders in BCI emulation, in contrast to a poorer PPVT model. These results suggest that an RNN neural encoder is sufficient to accurately predict decoder behavior for 2D cursor control.

Testing subject in clinical trials can provide personal experience when control BCI decoders, in contrast to monkey experiments. We also found that our testing subjects can tell the difference between decoder algorithms. For example, ReFIT-KF has unnoticeable delay between movement intention and decoded cursor, and moves faster than FIT-KF, but not easy to dial-in. VKF shows longer delay between movement intention and decoded cursor, causing the trajectory to overshot easily. We also observed the correction intention both in empirical and emulator data. Our emulator can also provide valuable personal experience which may be helpful in designing decoder algorithm.

The hand kinematics reflect the "movement intention" in the motor cortex. Neural encoder transforms hand kinematics into binned spike counts. A natural response time is around 200 ms (see Figure 4.3e, hand-control). Interestingly, in BCI-controlled cases, testing subjects still initiate hand movement after 200 ms response time, however, decoded cursor does not move right after hand movements. Instead, there is a time delay. This is consistent with BCI results. The neural activity changes in motor cortical regions after target shows up, indicating the movement intention initiates. However, in BCI experiments, there is still a time delay even the decoder is controlled by the neural signal directly from the brain. We found that only the neural encoder incorporating a delay between hand speed and neural activity can reproduce this time delay as in empirical experiments.

4.6 Conclusions

We were able to accurately recapitulate detailed timing and kinematic performance of these decoders in BCI emulation, in contrast to a poorer PPVT model. Incorporating the neural delay and regularizing on RNN input weights on RNN neural encoder are necessary to reproduce decoder performance. We chose three significantly different studies to reproduce including FIT-KF, ReFIT-KF, FORCE, VKF, and PVKF. Our emulator correctly reproduced the conclusions of these studies including detailed timing and trajectory results. These results suggest it is feasible to accurately predict BCI performance without neurosurgery, enabling quantitative comparisons between different types of decoding algorithms. We anticipate this system can facilitate and accelerate the development of BCI decoders.

CHAPTER 5

BCI simulator

5.1 Summary

In a BCI system, a user interacts with an imperfect decoder and continuously changes motor commands in response to unexpected decoded movements. This "closed-loop" nature of BCI leads to emergent interactions between the user and decoder that are challenging to model. The gold standard for BCI evaluation is therefore real-time experiments, which significantly limits the community and speed of BCI research. Our BCI emulator reproduced published studies but still requiring experiments with a human-in-the-loop. We therefore replace the user-in-the-loop in BCI emulation with an AI agent. We present a new BCI simulator that enables researchers to accurately and quickly design BCIs entirely in software. Our simulator replaces the BCI user with a deep reinforcement learning (RL) agent that interacts with a simulated BCI system and learns to optimally control it. We demonstrate that our simulator is accurate and versatile, reproducing the published results of three distinct types of BCI decoders: (1) a state-of-the-art linear decoder (FIT-KF), (2) a "two-stage" BCI decoder requiring closed-loop decoder adaptation (ReFIT-KF), and (3) a nonlinear decoder (FORCE). We anticipate this simulator will help democratize and significantly accelerate BCI research.

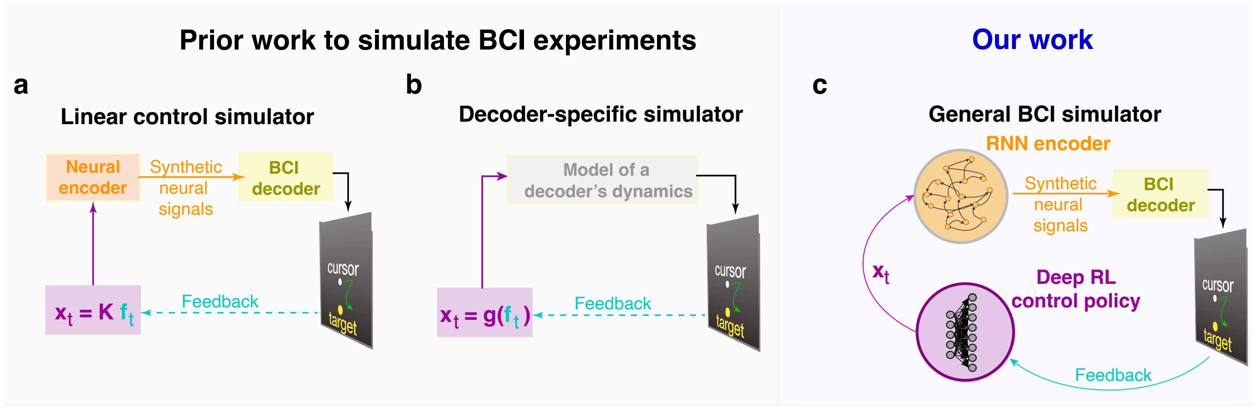


Figure 5.1: **(a)** The linear control simulator replaces the human with a linear control policy. This makes the simulator entirely software, but the control policy is not accurate. **(b)** The simulator by Willett et al. [1] enables hyperparameter optimization of a single decoder by modeling its dynamics (gray box), but cannot generalize to new decoders. **(c)** Our goal is the general solution, a purely software simulator that accurately predicts the performance of any type of decoder.

5.2 Introduction

Intracortical brain-computer interfaces (BCIs) are expensive and time-consuming to design because accurate evaluation traditionally requires real-time experiments. To accelerate BCI research, prior studies have attempted to emulate or simulate BCI systems. Cunningham et al. [2] developed a BCI "emulator" where a human subject's hand movements generate synthetic neural activity to control a BCI decoder, removing invasive neurosurgery but still requiring experiments with a human-in-the-loop. While this approach is useful, its use of human experiments significantly limits its community use and speed To remove the human-in-the-loop and avoid experiments, Lagang and Srinivasan [23] used the linear quadratic regulator from control theory to approximate the human's BCI control policy. Other studies have also used linear policies from control theory in BCI design [24, 25, 26, 27, 28]. However, as shown by Willett and colleagues [29, 1], and by further experiments in this study, linear control policies are a poor approximation of user control policy and result in incorrect conclusions. Finally, Willett et al. [1] designed a simulator that enabled hyperparameter tuning of a VKF. This simulator, however, does not have a neural data model and requires closed-loop experiments to model a decoder's dynamics. This simulator can therefore only optimize decoders already tested in closed-loop intracortical experiments and does not generalize to new, never-before-tested, decoders. It is also limited to optimize decoders of a linear form.

In contrast to prior work, our BCI simulator aims to faithfully model all aspects of BCI control to accurately estimate the performance of any BCI decoder entirely in software. This requires accurately modeling the unique user-decoder interactions without human or monkey experiments. It also requires generating synthetic neural activity that provides sufficient information for downstream neural decoders. Our simulator solves the general problem: it accurately evaluates the performance of representative decoder algorithms, both linear and nonlinear, entirely in software. Our idea fundamentally involves training an artificial intelligence (AI) agent that learns to control new decoders through deep reinforcement learning (RL) and reproduces decoder performance.

We next sought to replace the user-in-the-loop in BCI emulation with an AI agent. The AI agent must interact with the BCI system to acquire targets, as humans did in the BCI emulator. We fixed the neural encoder and focused on agent optimization. Because the BCI emulator reproduced published studies, we reasoned that successful replication of user control policy would lead to a purely software BCI simulator that also reproduces published studies. As the overall system is nonlinear, we found a linear control policy, such as a linear quadratic regulator, failed to control the FORCE decoder.

Even when we relaxed the neural encoder to a PPVT, an LQR agent struggled to control some decoders and did not reproduce prior studies. This is consistent with prior work that showed linear control policies are inadequate for BCI control [1]. We therefore trained nonlinear control policies (Figure 5.2). The AI agent must generate movements resulting in similar online decoder performance as in intracortical experiments. Because the agent had to learn to control each decoder anew, some decoders of which may have never before been tested, we found imitation learning techniques were not sufficient because online control data in general is not available. Further details, including hyperparameters, our use of curriculum learning, the RL reward setting, and a more detailed description of the RL problem, are described in the Methods.

5.3 Methods

5.3.1 Simulator setup

The simulator replaces the user-in-the-loop with a deep RL agent. It therefore allows us to predict the performance of BCIs without human subjects. In this setting, the hand kinematics (input of the neural encoder) are controlled by the deep RL agent, instead of a human. The rest of the simulator mimics the BCI emulator. The hand position and velocities of the agent are updated according to the following kinematic equation:

$$\begin{bmatrix} H^p_{t+1} \\ H^v_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} H^p_t \\ H^v_t \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} H^a_t$$

where H^a_t is the hand acceleration at time t. The synthetic spike counts were synthesized with hand kinematics and decoded by decoders into cursor position as the following:

$$\hat{y}_t = \text{Encoder}(H^p_t, H^v_t)$$
$$C^p_t = \text{Decoder}(\hat{y}_t)$$

Our goal was to therefore have the agent generate hand accelerations, H^a_t, that mimicked how a user would control a BCI. The input to the agent which is the “observation” was the hand’s position and velocity, cursor’s position and velocity, as well as the target. The output of the agent was a hand acceleration that would subsequently be translated into neural activity, $\hat{y}t$, and decoded into a new cursor position, C^p_t.

5.3.2 AI agents

5.3.2.1 LQR

The linear quadratic regulator (LQR) algorithm is an automated way of finding a state-feedback controller based on system dynamics and cost constraints. System dynamics are described by a set of linear differential equations and cost constraints are described by quadratic functions. In our tasks, the observation space was the concatenation of cursor position, cursor velocity and target position. The action space was the hand acceleration.

The LQR assumes that the observations and inputs are related according to a linear dynamical system:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{5.1}$$

In general, when using LQR, the $\mathbf{A}$ and $\mathbf{B}$ matrices are known. However, our system dynamics are nonlinear, probabilistic, and change based on the neural encoder and decoder. Using naive system dynamics leads to a suboptimal feedback controller. To model system dynamics, we therefore used an algorithm to iteratively learn system dynamics $\mathbf{A}$ and $\mathbf{B}$ from the data collected while performing tasks. The data includes observation, $\mathbf{x}_k$, next observation, $\mathbf{x}_{k+1}$, and action, $\mathbf{u}_k$. We approximate the relationship between hand kinematics and cursor kinematics of BCI system with a linear dynamic system as:

Algorithm 2 Iteratively estimate system dynamics

```
Require: $\mathbf{x}$ and $\mathbf{u}$ collected when interacting with BCI systems.
Ensure: $\mathbf{A}^*$ and $\mathbf{B}^*$ linear system approximates
  $\mathbf{A}^* = \underset{\mathbf{A}}{\operatorname{argmin}}\, \mathbb{E}\left[\|\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k\|^2\right]$
  $\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmin}}\, \mathbb{E}\left[\|\mathbf{x}_{k+1} - \mathbf{A}^*\mathbf{x}_k - \mathbf{B}\mathbf{u}_k\|^2\right]$
  repeat
    $\mathbf{A}^* = \underset{\mathbf{A}}{\operatorname{argmin}}\, \mathbb{E}\left[\|\mathbf{x}_{k+1} - \mathbf{A}x_k - \mathbf{B}^*u_k\|^2\right]$
    $\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmin}}\, \mathbb{E}\left[\|\mathbf{x}_{k+1} - \mathbf{A}^*\mathbf{x}_k - \mathbf{B}\mathbf{u}_k\|^2\right]$
  until $\mathbf{A}^*$ and $\mathbf{B}^*$ converge
```

In our algorithm, we assume system state can evolve through its internal dynamics, $\mathbf{A}$, without requiring external input. We therefore initialize $\mathbf{B} = 0$. After inferring $\mathbf{A}^*$ and $\mathbf{B}^*$, we use the LQR algorithm to infer a linear policy, $\mathbf{u}_t = \mathbf{K}\mathbf{x}_t$.

5.3.2.2 Naive PPO

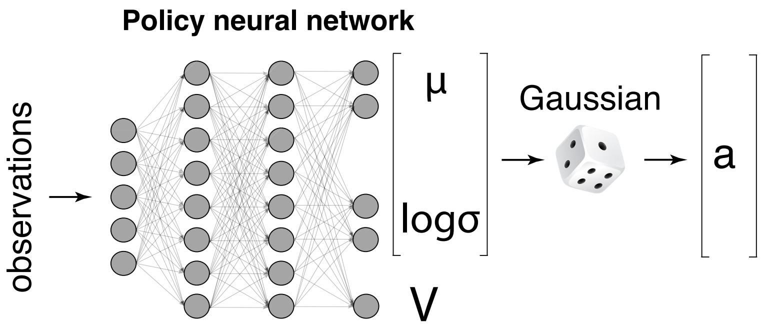


Figure 5.2: Architecture of the deep RL policy which takes observations and generates hand acceleration and state value.

To train the deep RL agent, we used proximal policy optimization (PPO) with new regularizations to constraint agent behavior as humans had. PPO uses a clipped surrogate objective function with a goal of reducing KL divergence between successive gradient updates [94]. To encourage exploration, the entropy of actions given state was added to the objective function. The loss function of PPO is defined as the following:

$$\begin{aligned}
\mathcal{L}_{PPO} &= \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] + \beta H(\pi_\theta(\cdot|s_t)) \\
r_t(\theta) &= \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \\
\hat{A}_t &= \Sigma_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l}^V \\
\delta_t^V &= R_t + \gamma V(s_{t+1}) - V(s_t),
\end{aligned}$$

where r_t is the probability ratio, $\hat{A}_t$ is the advantage value, R_t is the reward from environment, $V(s_t)$ is the output from the value function given s_t, γ and λ are hyperparameters in generalized advantage estimation (GAE) [95]. PPO updates were performed with first-order stochastic gradient descent (SGD) or Adam. The agent outputs were the means and logarithmic standard deviations of Gaussian distributions modeling the hand acceleration. During testing, the actions injected into the tasks were sampled from Gaussian distributions on each dimension of action space.

The PPO algorithm included a policy and a value network. The policy network was a feedforward neural network with two affine layers of 64 neurons, using $\tanh(\cdot)$ as the activation function, followed by a linear layer to output means and logarithmic standard deviation of Gaussian distributions modeling the 2D hand acceleration. The value network shared the same first two 64-neuron affine layers, followed by a linear layer to estimate the value function $V(s_t)$.

5.3.2.3 Constrained PPO

The PPO agent, naively trained, does not exhibit realistic physical behaviors, resulting in unmatched decoder performance (Figure 5.4). For example, it could make arbitrarily large accelerations, something humans cannot do with their arms because of biomechanical constraints. We therefore developed two additional regularization terms to encourage the agent to act more like a BCI user would. We first incorporated a smoothness constraint that penalized the KL divergence on consecutive actions. Additionally, we incorporated a zeroness constraint that penalized changes to the policy that move policy away from a Gaussian distribution with zero mean and unit variance. This encouraged the network to conserve energy, not moving unless it was necessary. The objective function of our constrained PPO agent is therefore:

$$\mathcal{L} = \mathcal{L}_{PPO} + \alpha_{smoothness}\mathrm{KL}(\pi_\theta(.|s_t)\|\pi_\theta(.|s_{t+1})) + \alpha_{zeroness}\mathrm{KL}(\pi_\theta(.|s_t)\|\mathcal{N}(0,1)) \tag{5.2}$$

The policy network, π_θ, outputs mean μ and logarithmic standard deviation, $\log \sigma$, of a Gaussian distribution over the agent's actions. We used the output of the policy network to estimate the KL divergence penalties. Consider two Gaussian distributions, $p(x)$ and $q(x)$, with means, μ_1 and μ_2, and standard deviations, σ_1 and σ_2, respectively. The KL divergence of these two Gaussian distributions is calculated in the following way:

$$\begin{aligned}
\mathrm{KL}(p\|q) &= \int p(x)\log\frac{p(x)}{q(x)}dx = \int p(x)\log p(x)dx - \int p(x)q(x)dx \\
\int p(x)\log q(x)dx &= \int p(x)\log\left(\frac{1}{(2\pi\mu^2)^{\frac{1}{2}}}\exp^{-\frac{(x-\mu_2)^2}{2\mu_2^2}}\right)dx \\
&= -\frac{1}{2}\left(\log 2\pi\mu_2^2\right) - \int p(x)\left[\frac{x^2-2x\mu_2+\mu 2^2}{2\mu_2^2}\right]dx \\
&= -\frac{1}{2}\left(\log 2\pi\mu_2^2\right) - \frac{\mu_1^2+(\mu_1-\mu_2)^2}{2\mu_2^2} \\
\int p(x)\log p(x)dx &= -\frac{1}{2}(1+\log 2\pi\sigma^2) \\
\mathrm{KL}(p\|q) &= \log\frac{\mu_2}{\mu_1} + \frac{\mu_1^2+(\mu_1-\mu_2)^2}{2\mu_2^2} - \frac{1}{2}
\end{aligned}$$

where, we used $\mathrm{Var}(x) = \mathbb{E}[x^2] - \mathbb{E}[x]^2$ to replace $\int p(x)x^2dx = \mu^2 - \sigma^2$.

5.3.2.4 Train RL agents with curriculum learning

We first trained an agent to perform the center-out-and-back task in a bypass mode which has cursor position equal to hand position. We adjusted the zero constraint and smooth constraint until the resulting behavior was close to Monkey J's behavior. All other agents learned from this pretrained control policy. Although reward shaping facilitates the learning process, agents' behaviors were also determined by the shaped reward. We therefore only trained agents with a simple reward setting which is 1 if agents successfully acquired targets and 0 others. We applied curriculum learning to facilitate the learning process by starting from a larger target acceptance window, 120 mm, and progressively shrunk it down to 30 mm if the overall target success rate was more than 90%.

5.4 Results

5.4.1 Linear control policy and linear neural encoder lead to incorrect conclusions

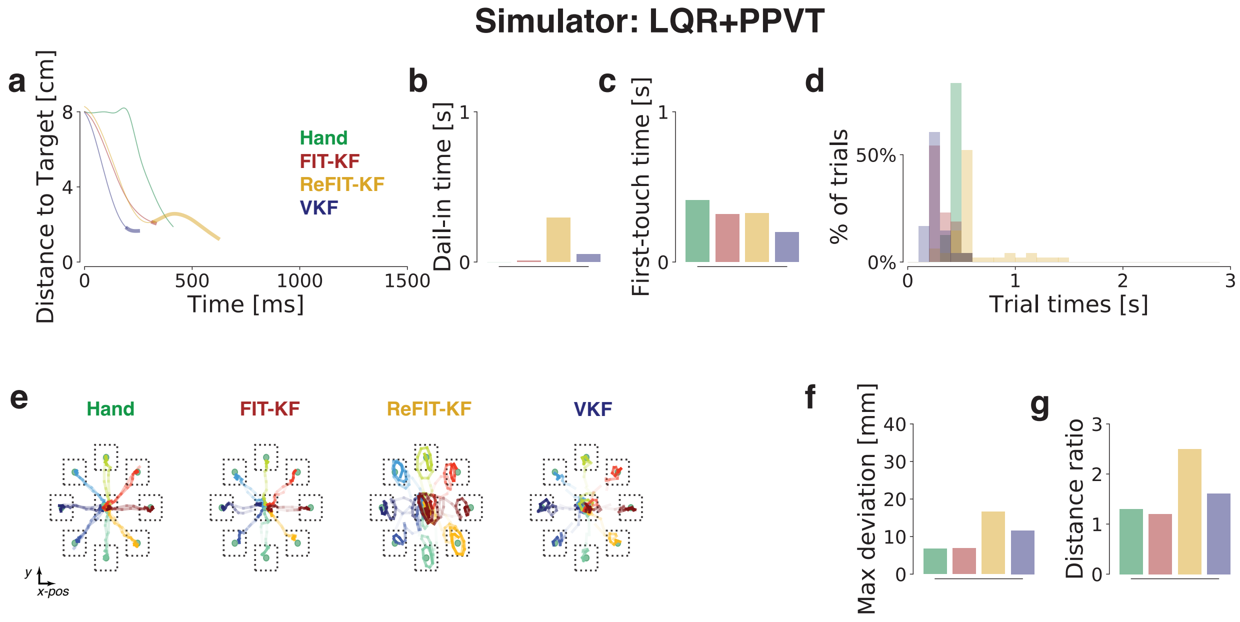


Figure 5.3: A BCI simulator with an LQR agent and PPVT neural encoder. The LQR agent does not reproduce prior studies with PPVT neural encoder. We do not show the results of the FORCE decoder because it was uncontrollable, consistent with what we observed in the emulator. Quantitative results are summarized in Table 5.1.

The linear quadratic regulator (LQR) algorithm is an automated way of finding a state-feedback controller based on system dynamics and cost constraints. System dynamics are described by a set of linear differential equations and cost constraints are described by quadratic functions. In general, when using LQR, the system dynamics **A** and **B** matrices are known or can be linearly approximated from a data-driven method (see Methods).

Table 5.1: Decoder performance in simulator using a LQR agent and PPVT neural encoder. TT, FTT, and DIT are trial time, first-touch time, and dial-in time, respectively.

Decoder	TT [ms]	FTT [ms]	DIT [ms]	distance ratio	max deviation [mm]	success rate	# of center-out trials
hand	411.46	411.46	0	1.3	6.74	1.00	48
FIT-KF	327.08	317.71	9.38	1.2	6.91	1.00	48
ReFIT-KF	620.83	326.04	294.79	2.5	16.62	1.00	48
VKF	248.96	197.92	51.04	1.61	11.66	1.00	48

Although LQR can control linear decoders including FIT-KF, ReFIT-KF, and VKF, LQR agent did not reproduce prior studies. PPVT-synthesized neural signal does not recapitulate the complexity of real neural signal, resulting in a simple BCI system dynamics. The simple VKF decoder results in the best decoder performance which is not consistent with prior studies. Therefore, a better neural encoder is needed to recapitulate important neural properties to closely simulate the BCI system dynamics. BCI system dynamics are nonlinear, probabilistic, and change based on the neural encoder and decoder. As the overall system is nonlinear, a nonlinear control policy is needed to interact with nonlinear BCI systems.

5.4.2 Unconstrained RL agent leads to incorrect conclusions

Table 5.2: Decoder performance in simulator using an unconstrained PPO agent and delayed regularized RNN neural encoder. TT, FTT, and DIT are trial time, first-touch time, and dial-in time, respectively.

Decoder	TT [ms]	FTT [ms]	DIT [ms]	distance ratio	max deviation [mm]	success rate	# of center-out trials
hand	373.79	370.26	3.53	1.58	8.43	1.00	248.0
FIT-KF	482.26	392.94	89.31	1.77	18.05	1.00	248.0
ReFIT-KF	399.60	322.78	76.81	1.58	16.08	1.00	248.0
FORCE	532.42	393.50	138.92	1.77	17.34	0.99	248.0
VKF	686.03	435.63	250.40	2.02	21.10	1.00	248.0

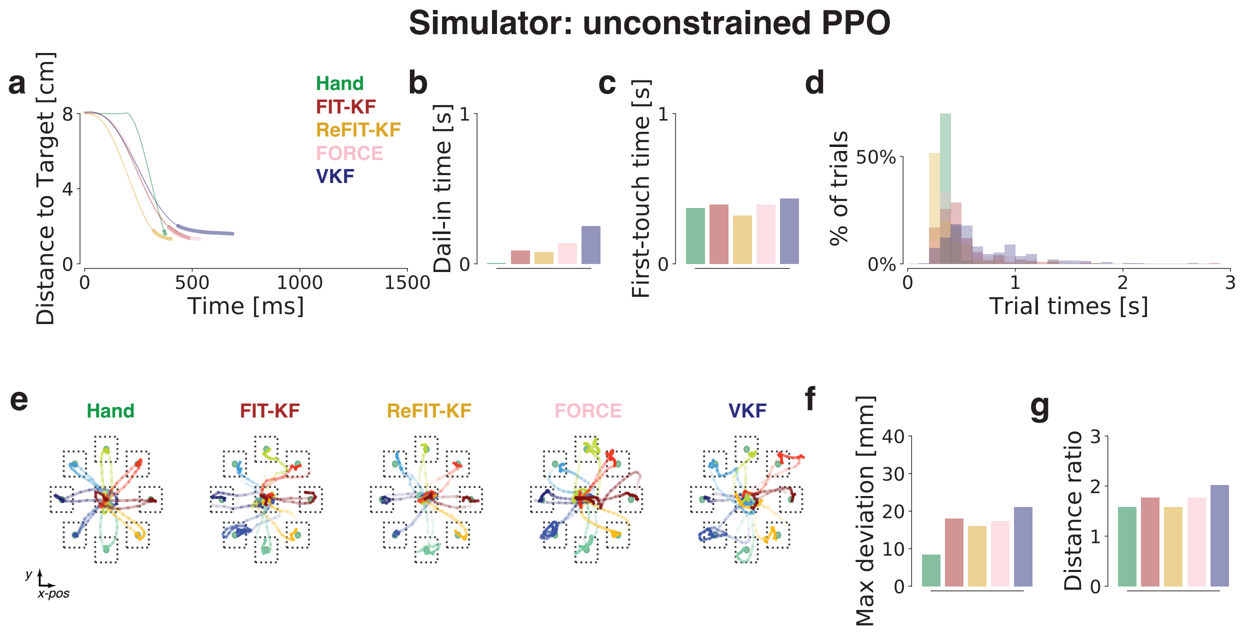


Figure 5.4: A BCI simulator with an unconstrained PPO agent and RNN neural encoder. Without the smoothness and zero regularizations we incorporated in RL training, the unconstrained PPO agent was able to generate more unphysical behavior and find superior strategies to controlling the decoders. The first-touch times and dial-in times are all shorter than in prior studies, and the VKF first-touch time is not significantly worse than the other decoders. The ordering of max deviation and distance ratio are also mismatched from intracortical BCI data. Quantitative results are summarized in Table 5.2.

We trained nonlinear control policies with proximal policy optimization (PPO) to interact with nonlinear BCI systems. Because control policy is decoder dependent, the deep RL agent was trained to learn the emergent closed-loop interactions helpful for controlling each decoder. We performed the FIT-KF, ReFIT-KF, and FORCE studies in a BCI simulator using trained RL agents. We found that PPO agents successfully performed tasks and the decoder performance is shown in Figure 5.4 and summarized in Table 5.2.

The ordering of first-touch time were reproduced, indicating RNN neural encoder reproduced critical neural activity with respect to each decoder. However, the PPO agent intends to acquire targets in a minimum time because the objective function is to maximize the discounted cumulative reward. The PPO agent, naively trained, could make arbitrarily changes and large accelerations, something humans cannot do with their arms because of biomechanical constraints. We found that the dial-in times were too short and the overall decoder performance does not match. Therefore, to closely reproduce decoder performance in monkey experiments, the behavior of PPO agent needs to be constrained to be similar to natural behavior.

5.4.3 A software BCI simulator with a constrained RL agent reproduces published BCI studies.

Table 5.3: Decoder performance in simulator using a constrained PPO agent and the delayed regularized RNN neural encoder. TT, FTT, and DIT are trial time, first-touch time, and dial-in time, respectively.

Decoder	TT [ms]	FTT [ms]	DIT [ms]	distance ratio	max deviation [mm]	success rate	# of center-out trials
hand	415.42	415.42	0	1.37	6.93	1.00	248
FIT-KF	577.62	522.18	55.44	1.68	19.42	1.00	248
ReFIT-KF	612.15	464.37	147.77	1.89	19.34	1.00	248
FORCE	892.36	563.81	328.56	2.31	25.24	0.96	248
VKF	1138.24	631.22	507.01	2.7	28.6	0.91	243

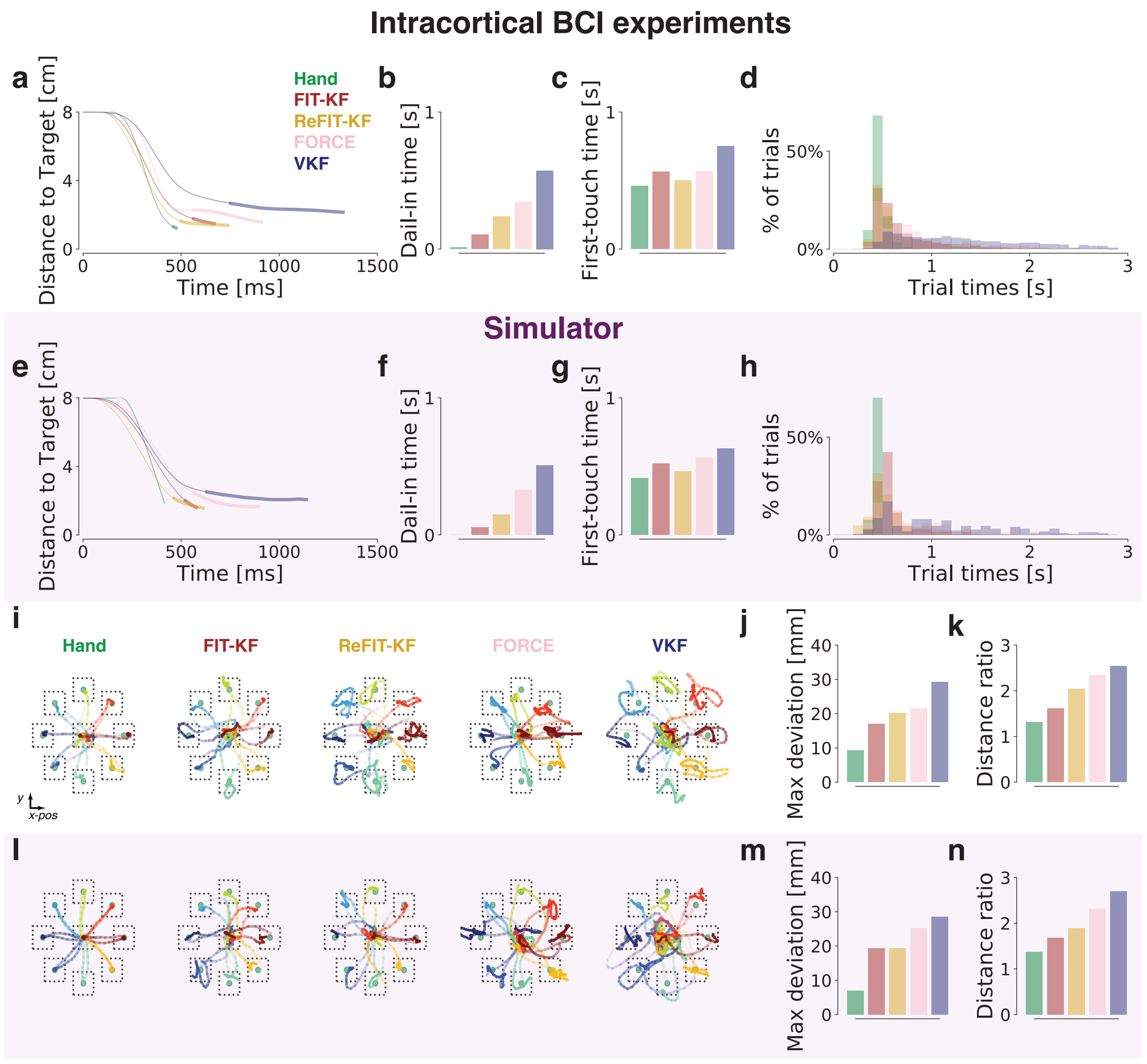


Figure 5.5: BCI simulator reproduces published studies. **(a-d, i-k)** Reproduction of the same panels from Figure 3, for ease of comparison of the BCI simulator to intracortical BCI experiments. **(e-h)** Same as **(a-d)** but for the BCI simulator. **(l-n)** Same as **(i-k)** but for the BCI simulator. BCI and simulator results are summarized in Table 2.2 and Table 5.2, respectively.

We trained the deep RL agent to control each of the decoders in the published studies. We found that the deep RL agent exhibited qualitatively similar decoder dependent control policies to monkeys controlling intracortical BCIs and humans controlling the BCI emulator (FIT-KF average "hand" trajectory length: 284 mm, VKF: 373 mm, $p < 10^{-7}$, Wilcoxon rank-sum test).

We performed the FIT-KF, ReFIT-KF, and FORCE studies in a BCI simulator using trained RL agents. We found the BCI simulator reproduced all the key conclusions of these studies, like in the emulator (Figure 5.5 and Figure 5.7 for PVKF), including detailed timing and kinematic differences in decoder comparisons. This included correctly predicting the trends in FTT and DIT across all decoders, as well as kinematic trends in distance ratio and max deviation. Together, these results show the BCI simulator, using a deep RL agent, and an RNN neural encoder, reproduced the key timing and kinematic performance statistics of decoders, including timing at the single-trial resolution. Importantly, our BCI simulator arrived at the same conclusions as prior intracortical studies that required months to years of invasive experiments, but did so entirely in software simulation.

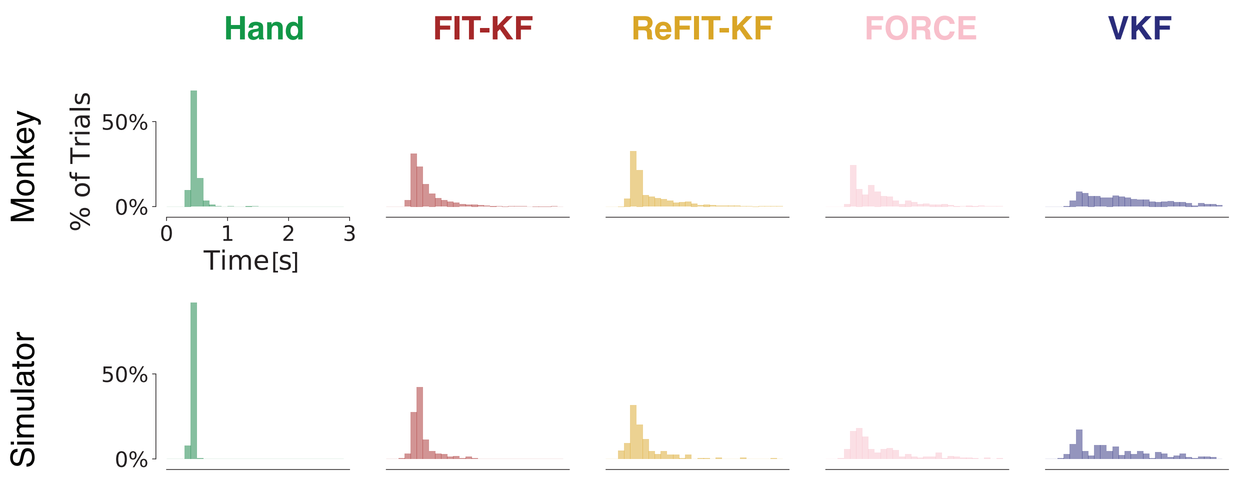
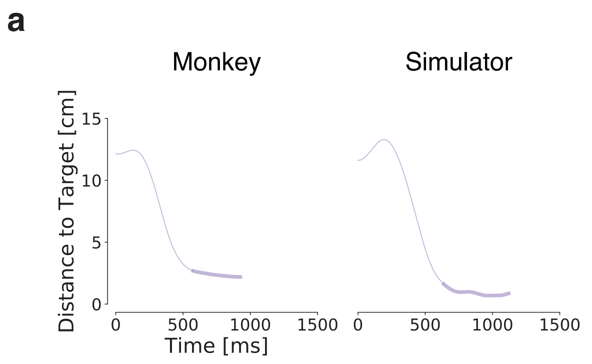


Figure 5.6: The simulator reproduce trial time distributions observed in prior studies. We observed similar trial time distributions across decoders in comparison to the closed-loop BCI data. FIT-KF and ReFIT-KF had narrower distributions while FORCE and VKF had relatively wider distributions.



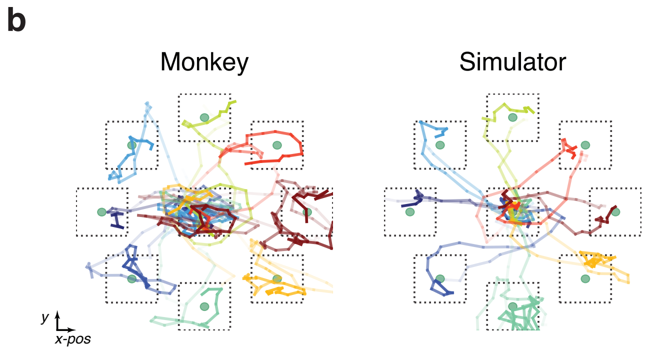


Figure 5.7: Simulator reproduce PVKF decoder performance. **(a)** Distance-to-target plots from BCI and simulator experiments. The distance-to-target profiles in simulator qualitatively resembled those in monkey experiments. **(b)** Randomly sampled decoded trajectories for each direction in each experiment. The simulator reproduce jittery and jumpy cursor trajectories during PPVT control.

5.5 Discussion

We incorporated a deep-learning-based neural encoder and deep RL agent to build a BCI simulator entirely in software without requiring invasive neurosurgery or user-in-the-loop experiments. Previously, we found that a BCI emulator, using a neural encoder that models the kinematic-to-neural relationship closely, reproduced the detailed timing and decoder's trajectory kinematic in online BCI experiments. Here, we further were able to replace the user-in-the-loop with a deep RL agent trained with constraints to implement a purely software simulator. In general, we found that high capacity neural networks, in both the neural encoder and deep RL agent, were necessary to accurately simulate key prior studies, and we expect that our work can predict the performance of never-before-tested decoders.

Critically, our work significantly extends prior attempts at BCI simulation and emulation by not requiring additional physical experiments. Further, because our BCI simulator is implemented entirely in software, it is straightforward to simulate experiments which require online user-decoder interaction such as designing and optimizing decoders, including algorithm testing, hyperparameter sweeps, task parameter sweeps, and other adjustable variables, and also add-on algorithms such as decoder stabilization and AI argumentation. Importantly, this pure software platform can used to benchmark algorithms, a current limitation that has complicated algorithm comparisons in prior BCI literature. We expect that BCI algorithms will be faster to design and optimize, as opposed to taking months of experiments. We further have released the code, making this software accessible to all researchers. We emphasize that, by reproducing three distinct studies, we found that our simulator accurately estimates performance of linear and nonlinear decoders, in addition to those that require multiple training stages, as in closed-loop decoder adaptation.

The key innovation was using neural network based RL agent to replace user-in-the-loop in BCI emulator. Because the overall BCI system includes nonlinearities, including from movement intention to neural activity, or even in the decoder algorithm, we found linear feedback controllers like LQR that rely on linear system dynamics ultimately failed. In contrast, deep RL agents trained with constrained PPO were able to implement control strategies that reproduced published studies. Future work can further incorporate physical constraints such as muscle skeleton model to better simulate natural movement intention. Also, along with new technical changes in continual learning, meta learning, or off-policy methods from RL community, we believe RL agents can be easier to train and more general.

BCI community takes two decades to have significant advances in 2D plane decoding, and is tackling high-degree-of-freedom decoding such as BCIs those control robotic arm or fingers. Our future work may consider how to simulate high DOF neural encoder and tasks. A key consideration will be to train the neural encoding model, which would require simultaneous neural recordings and high DOF kinematics or kinetics. After training a neural encoder, we anticipate a deep RL agent will be capable of learning to control these higher-DOF decoders to simulate BCI performance.

We believe our BCI simulator can be used to develop a generalized neural decoder with respect to all neural encoders without having multiple subjects performing online experiments repeatedly. BCI community takes two decades to have significant advances in 2D plane decoding, and is tackling high-degree-of-freedom decoding such as BCIs those control robotic arm or fingers. Our future work may consider how to simulate high DOF neural encoder and tasks. A key consideration will be to train the neural encoding model, which would require simultaneous neural recordings and high DOF kinematics or kinetics. After training a neural encoder, we anticipate a deep RL agent will be capable of learning to control these higher-DOF decoders to simulate BCI performance.

5.6 Conclusions

We were able to accurately recapitulate detailed timing and kinematic performance of these decoders in BCI simulation. PPO agent with nonlinear function policy can control nonlinear BCI system which Linear-quadratic regulator (LQR) agent cannot control. Incorporating smoothness and zeroness constraints is critical to reproduce BCI decoder performance. We replace the human-in-the-loop in BCI emulator experiments with constrained PPO agents. Our simulator correctly reproduced the conclusions of BCI experiments. These results suggest it is feasible to accurately predict BCI performance without neurosurgery, enabling quantitative comparisons between different types of decoding algorithms. We anticipate this system can further facilitate and accelerate the development of BCI decoders.

CHAPTER 6

Conclusion

Throughout this dissertation, we have demonstrated steps toward a BCI simulator and proved the concept that synthetic neural activity and deep RL agents can be combined to accurately simulate cursor control BCIs. Our design approach uses published BCI experiments as ground truth decoder comparisons, physical BCI emulator experiments to optimize the neural encoder in isolation, and software BCI simulator to train AI agents. We chose to replicate three distinct BCI studies that improved BCI decoders through different innovations, namely: (1) a state-of-the-art linear decoder (FIT-KF), (2) a "two-stage" BCI decoder requiring closed-loop decoder adaptation (ReFIT-KF), and (3) a nonlinear decoder (FORCE). Both our BCI emulator and simulator can accurately recapitulate detailed timing and kinematic performance of these decoders in monkey BCI experiments. These results suggest that our emulator and simulator are accurate and versatile.

This dissertation shows how BCI emulator design can benefit profoundly from a synergy between deep learning and neuroscience. Ideas from neuroscience is important for solving specific problems encountered in designing neural encoders. Concretely, we demonstrated in Chapter 3 that RNN neural encoder incorporating neural path delay and RNN input weight regularization more faithfully reproduces neural activity and decoded kinematics in validation dataset. Critically, in Chapter 4, we demonstrated that our emulator correctly reproduced the conclusions of these studies, in addition to reproducing precise details of control, including: (1) distance-to-target profiles, closely matching the first touch time (FTT) and the dial-in time (DIT), (2) the distribution of trial times, and (3) cursor trajectories observed in prior monkey online experiments.

However, while BCI emulator is useful, and we used it to validate a neural encoder, its use of human experiments significantly limits its community use and speed. In Chapter 5, we replaced the BCI user with a deep reinforcement learning (RL) agent that interacts with a simulated BCI system and learns to optimally control it. We demonstrated that nonlinear control policy is needed to control BCI systems which are naturally nonlinear. Further, we demonstrated the importance of the regularizations we introduced to enforce smoothness and low-energy actions from the agent; without these, the agent could make non-physical arbitrary accelerations and decelerations. These results suggest it is feasible to accurately predict BCI performance without neurosurgery and physical experiments, enabling quantitative comparisons between different types of decoding algorithms. We anticipate this system can facilitate and accelerate the development of BCI decoders.

The work in this dissertation highlights the need for nonlinear approaches for designing BCI simulator. By combining approaches from deep learning with basic science insights from systems neuroscience, we addressed critical hurdles to BCI simulator. We demonstrated that by modeling the kinematic-to-neural relationship as closely as possible, we were able to accurately recapitulate detailed timing and kinematic performance of representative decoder algorithms. I believe that approaches that continue to use insights from deep learning and systems neuroscience in BCI simulator will play a important role in accelerating BCI research. Some natural applications and extensions of this work include:

- Developing neural decoders on BCI emulator and simulator is the most direct application. We can build multiple encoder models across days to simulate the non-stationary neural distribution. It is also possible to build encoder models from multiple subjects, and benchmark decoders across separate encoding models. We believe our BCI simulator can be used to develop a generalized neural decoder with respect to all neural encoders without having multiple subjects performing online experiments repeatedly. Robust decoders should generalize well across all neural encoders.

- Generalizing to BCIs driven by other neural recording modalities (e.g., ECoG and EEG), or Extending to high-degree-of-freedom decoding such as BCIs those control robotic arm or fingers. A key consideration will be to train the neural encoding model, which would require simultaneous neural recordings and high DOF kinematics or kinetics. After training a neural encoder, we anticipate a deep RL agent will be capable of learning to control these higher-DOF decoders to simulate BCI performance. In addition, this new BCI system can incorporate potential biomechanical models to constrain agent's behavior.

Bibliography

[1] F. R. Willett, D. R. Young, B. A. Murphy, W. D. Memberg, C. H. Blabe, C. Pandarinath, S. D. Stavisky, P. Rezaii, J. Saab, B. L. Walter, J. A. Sweet, J. P. Miller, J. M. Henderson, K. V. Shenoy, J. D. Simeral, B. Jarosiewicz, L. R. Hochberg, R. F. Kirsch, and A. Bolu Ajiboye, "Principled BCI decoder design and parameter selection using a feedback control model," *Sci. Rep.*, vol. 9, no. 1, p. 8881, Jun. 2019.

[2] J. P. Cunningham, P. Nuyujukian, V. Gilja, C. A. Chestek, S. I. Ryu, and K. V. Shenoy, "A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces," *J. Neurophysiol.*, vol. 105, no. 4, pp. 1932–1949, Apr. 2011.

[3] K.-F. Liang and J. C. Kao, "Deep learning neural encoders for motor cortex," *IEEE Trans. Biomed. Eng.*, vol. 67, no. 8, pp. 2145–2158, Aug. 2020.

[4] P. R. Kennedy and R. A. E. Bakay, "Restoration of neural output from a paralyzed patient by a direct brain connection," *Neuroreport*, vol. 9, no. 8, pp. 1707–1711, Jun. 1998.

[5] P. R. Kennedy, R. A. E. Bakay, M. M. Moore, K. Adams, and J. Goldwaithe, "Direct control of a computer from the human central nervous system," *IEEE Trans. Rehabil. Eng.*, vol. 8, no. 2, pp. 198–202, Jun. 2000.

[6] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3D neuroprosthetic devices," *Science*, vol. 296, no. 5574, pp. 1829–1832, Jun. 2002.

[7] L. R. Hochberg, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, and J. P. Donoghue, "Neuronal ensemble control of prosthetic devices by a human with tetraplegia," *Nature*, vol. 442, no. 7099, pp. 164–171, Jul. 2006.

[8] V. Gilja, P. Nuyujukian, C. A. Chestek, J. P. Cunningham, B. M. Yu, J. M. Fan, M. M. Churchland, M. T. Kaufman, J. C. Kao, S. I. Ryu, and K. V. Shenoy, "A high-performance neural prosthesis enabled by control algorithm design," *Nat. Neurosci.*, vol. 15, no. 12, pp. 1752–1757, Nov. 2012.

[9] J. M. Fan, P. Nuyujukian, J. C. Kao, C. A. Chestek, S. I. Ryu, and K. V. Shenoy, "Intention estimation in brain-machine interfaces," *J. Neural Eng.*, vol. 11, no. 1, p. 016004, Feb. 2014.

[10] M. M. Shanechi, A. L. Orsborn, H. G. Moorman, S. Gowda, S. Dangi, and J. M. Carmena, "Rapid control and feedback rates enhance neuroprosthetic control," *Nat. Commun.*, vol. 8, p. 13825, 2017.

[11] J. C. Kao, P. Nuyujukian, S. I. Ryu, M. M. Churchland, J. P. Cunningham, and K. V. Shenoy, "Single-trial dynamics of motor cortex and their applications to brain-machine interfaces," *Nat. Commun.*, vol. 6, no. May, pp. 1–12, 2015.

[12] J. C. Kao, P. Nuyujukian, S. I. Ryu, and K. V. Shenoy, "A high-performance neural prosthesis incorporating discrete state selection with hidden markov models," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 4, pp. 935–945, Apr. 2017.

[13] K. Ganguly and J. M. Carmena, "Emergence of a stable cortical map for neuroprosthetic control," *PLoS Biol.*, vol. 7, no. 7, p. e1000153, Jul. 2009.

[14] A. L. Orsborn, H. G. Moorman, S. A. Overduin, M. M. Shanechi, D. F. Dimitrov, and J. M. Carmena, "Closed-Loop decoder adaptation shapes neural plasticity for skillful neuroprosthetic control," *Neuron*, vol. 82, no. 6, pp. 1380–1393, Jun. 2014.

[15] D. B. Silversmith, R. Abiri, N. F. Hardy, N. Natraj, A. Tu-Chan, E. F. Chang, and K. Ganguly, "Plug-and-play control of a brain-computer interface through neural map stabilization," *Nat. Biotechnol.*, vol. 39, no. 3, pp. 326–335, Mar. 2021.

[16] D. Sussillo, P. Nuyujukian, J. M. Fan, J. C. Kao, S. D. Stavisky, S. I. Ryu, and K. V. Shenoy, "A recurrent neural network for closed-loop intracortical brain-machine interface decoders," *J. Neural Eng.*, vol. 9, no. 2, p. 026027, Apr. 2012.

[17] D. Sussillo, S. D. Stavisky, J. C. Kao, S. I. Ryu, and K. V. Shenoy, "Making brain–machine interfaces robust to future neural variability," *Nat. Commun.*, vol. 7, p. 13749, Dec. 2016.

[18] P. Nuyujukian, J. M. Fan, V. Gilja, P. S. Kalanithi, C. A. Chestek, and K. V. Shenoy, "Monkey models for brain-machine interfaces: the need for maintaining diversity," in *Proceedings of the 33rd Annual Conference of the IEEE EMBS*, vol. 2011. Boston, Massachusetts: IEEE, Jan. 2011, pp. 1301–1305.

[19] S. M. Chase, A. B. Schwartz, and R. E. Kass, "Bias, optimal linear estimation, and the differences between open-loop simulation and closed-loop performance of spiking-based brain-computer interface algorithms," *Neural Netw.*, vol. 22, no. 9, pp. 1203–1213, 2009.

[20] S. Koyama, S. M. Chase, A. S. Whitford, M. Velliste, A. B. Schwartz, and R. E. Kass, "Comparison of brain-computer interface decoding algorithms in open-loop and closed-loop control," *J. Comput. Neurosci.*, vol. 29, no. 1-2, pp. 73–87, Aug. 2010.

[21] J. M. Carmena, M. A. Lebedev, R. E. Crist, J. E. O'Doherty, D. M. Santucci, D. F. Dimitrov, P. G. Patil, C. S. Henriquez, and M. A. L. Nicolelis, "Learning to control a brain-machine interface for reaching and grasping by primates," *PLoS Biol.*, vol. 1, no. 2, p. E42, Nov. 2003.

[22] S.-P. Kim, J. D. Simeral, L. R. Hochberg, J. P. Donoghue, and M. J. Black, "Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia," *J. Neural Eng.*, vol. 5, no. 4, pp. 455–476, Dec. 2008.

[23] M. Lagang and L. Srinivasan, "Stochastic optimal control as a theory of brain-machine interface operation," *Neural Comput.*, vol. 25, no. 2, pp. 374–417, Feb. 2013.

[24] S. Gowda, A. L. Orsborn, S. A. Overduin, H. G. Moorman, and J. M. Carmena, "Designing dynamical properties of Brain–Machine interfaces to optimize Task-Specific performance," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 22, no. 5, pp. 911–920, Sep. 2014.

[25] M. M. Shanechi, A. L. Orsborn, and J. M. Carmena, "Robust brain-machine interface design using optimal feedback control modeling and adaptive point process filtering," *PLoS Comput. Biol.*, vol. 12, no. 4, p. e1004730, 2016.

[26] M. M. Shanechi, Z. M. Williams, G. W. Wornell, R. C. Hu, M. Powers, and E. N. Brown, "A real-time brain-machine interface combining motor target and trajectory intent using an optimal feedback control design," *PLoS One*, vol. 8, no. 4, pp. 23–32, 2013.

[27] M. Benyamini and M. Zacksenhouse, "Optimal feedback control successfully explains changes in neural modulations during experiments with brain-machine interfaces," 2015.

[28] Y. Zhang and S. M. Chase, "Optimizing the usability of Brain-Computer interfaces," *Neural Comput.*, vol. 30, no. 5, pp. 1323–1358, May 2018.

[29] F. R. Willett, C. Pandarinath, B. Jarosiewicz, B. A. Murphy, W. D. Memberg, C. H. Blabe, J. Saab, B. L. Walter, J. A. Sweet, J. P. Miller, J. M. Henderson, K. V. Shenoy, J. D. Simeral, L. R. Hochberg, R. F. Kirsch, and A. Bolu Ajiboye, "Feedback control policies employed by people using intracortical brain–computer interfaces," *J. Neural Eng.*, vol. 14, no. 1, p. 016001, Nov. 2016.

[30] S. Dangi, A. L. Orsborn, H. G. Moorman, and J. M. Carmena, "Design and analysis of closed-loop decoder adaptation algorithms for brain-machine interfaces," *Neural Comput.*, vol. 25, no. 7, pp. 1693–1731, Jul. 2013.

[31] W. Wu, Y. Gao, E. Bienenstock, J. P. Donoghue, and M. J. Black, "Bayesian population decoding of motor cortical activity using a kalman filter," *Neural Comput.*, vol. 18, no. 1, pp. 80–118, Jan. 2006.

[32] L. R. Hochberg, D. Bacher, B. Jarosiewicz, N. Y. Masse, J. D. Simeral, J. Vogel, S. Haddadin, J. Liu, S. S. Cash, P. van der Smagt, and J. P. Donoghue, "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm," *Nature*, vol. 485, no. 7398, pp. 372–375, May 2012.

[33] J. L. Collinger, B. Wodlinger, J. E. Downey, W. Wang, E. C. Tyler-Kabara, D. J. Weber, A. J. C. McMorland, M. Velliste, M. L. Boninger, and A. B. Schwartz, "High-performance neuroprosthetic control by an individual with tetraplegia," *Lancet*, vol. 381, no. 9866, pp. 557–564, Feb. 2013.

[34] B. Wodlinger, J. E. Downey, E. C. Tyler-Kabara, A. B. Schwartz, M. L. Boninger, and J. L. Collinger, "Ten-dimensional anthropomorphic arm control in a human brain-machine interface: difficulties, solutions, and limitations," *J. Neural Eng.*, vol. 12, no. 1, p. 016011, Feb. 2015.

[35] F. R. Willett, D. T. Avansino, L. R. Hochberg, J. M. Henderson, and K. V. Shenoy, "High-performance brain-to-text communication via handwriting," pp. 249–254, 2021.

[36] J. P. Cunningham, P. Nuyujukian, V. Gilja, C. A. Chestek, S. I. Ryu, and K. V. Shenoy, "A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces," *J. Neurophysiol.*, vol. 105, no. 4, pp. 1932–1949, Apr. 2011.

[37] V. Gilja, C. Pandarinath, C. H. Blabe, P. Nuyujukian, J. D. Simeral, A. A. Sarma, B. L. Sorice, J. A. Perge, B. Jarosiewicz, L. R. Hochberg, K. V. Shenoy, and J. M. Henderson, "Clinical translation of a high-performance neural prosthesis," *Nat. Med.*, vol. 21, no. 10, pp. 1142–1145, 2015.

[38] C. A. Chestek, V. Gilja, P. Nuyujukian, J. D. Foster, J. M. Fan, M. T. Kaufman, M. M. Churchland, Z. Rivera-Alvidrez, J. P. Cunningham, S. I. Ryu, and K. V. Shenoy, "Long-term stability of neural prosthetic control signals from silicon cortical arrays in rhesus macaque motor cortex," *J. Neural Eng.*, vol. 8, no. 4, p. 045005, Aug. 2011.

[39] J. C. Kao, S. D. Stavisky, D. Sussillo, P. Nuyujukian, and K. V. Shenoy, “Information systems opportunities in brain-machine interface decoders,” *Proc. IEEE*, vol. 102, no. 5, pp. 666–682, 2014.

[40] D. Sussillo and L. F. Abbott, “Generating coherent patterns of activity from chaotic neural networks,” *Neuron*, vol. 63(4), pp. 544–557, 2009.

[41] T. Gollisch and M. Meister, “Eye smarter than scientists believed: neural computations in circuits of the retina,” *Neuron*, vol. 65, no. 2, pp. 150–164, Jan. 2010.

[42] A. D. Huberman, M. Manu, S. M. Koch, M. W. Susman, A. B. Lutz, E. M. Ullian, S. A. Baccus, and B. A. Barres, “Architecture and activity-mediated refinement of axonal projections from a mosaic of genetically identified retinal ganglion cells,” *Neuron*, vol. 59, no. 3, pp. 425–438, Aug. 2008.

[43] J. W. Pillow, J. Shlens, L. Paninski, A. Sher, A. M. Litke, E. J. Chichilnisky, and E. P. Simoncelli, “Spatio-temporal correlations and visual signalling in a complete neuronal population,” *Nature*, vol. 454, no. 7207, pp. 995–999, Aug. 2008.

[44] N. C. Rust, O. Schwartz, J. A. Movshon, and E. P. Simoncelli, “Spatiotemporal elements of macaque v1 receptive fields,” *Neuron*, vol. 46, no. 6, pp. 945–956, Jun. 2005.

[45] M. M. Churchland, J. P. Cunningham, M. T. Kaufman, J. D. Foster, P. Nuyujukian, S. I. Ryu, and K. V. Shenoy, “Neural population dynamics during reaching,” *Nature*, vol. 487, no. 7405, pp. 51–56, Jul. 2012.

[46] J. A. Michaels, B. Dann, and H. Scherberger, “Neural population dynamics during reaching are better explained by a dynamical system than representational tuning,” *PLoS Comput. Biol.*, vol. 12, no. 11, p. e1005175, 2016.

[47] R. E. Hampson, D. Song, B. S. Robinson, D. Fetterhoff, A. S. Dakos, B. M. Roeder, X. She, R. T. Wicks, M. R. Witcher, D. E. Couture, A. W. Laxton, H. Munger-Clary,

G. Popli, M. J. Sollman, C. T. Whitlow, V. Z. Marmarelis, T. W. Berger, and S. A. Deadwyler, “Developing a hippocampal neural prosthetic to facilitate human memory encoding and recall,” *J. Neural Eng.*, vol. 15, no. 3, p. 036014, Jun. 2018.

[48] S. J. Bensmaia and L. E. Miller, “Restoring sensorimotor function through intracortical interfaces: progress and looming challenges,” *Nat. Rev. Neurosci.*, vol. 15, no. 5, pp. 313–325, 2014.

[49] V. Gilja, C. A. Chestek, I. Diester, J. M. Henderson, and K. V. Shenoy, “Challenges and opportunities for next-generation intracortically based neural prostheses,” *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 7, pp. 1891–1899, 2011.

[50] J. E. O’Doherty, M. A. Lebedev, P. J. Ifft, K. Z. Zhuang, S. Shokur, H. Bleuler, and M. A. L. Nicolelis, “Active tactile exploration using a brain-machine-brain interface,” *Nature*, vol. 479, no. 7372, pp. 228–231, Nov. 2011.

[51] A. P. Georgopoulos, J. F. Kalaska, R. Caminiti, and J. T. Massey, “On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex,” *J. Neurosci.*, vol. 2, no. 11, pp. 1527–1537, Nov. 1982.

[52] M. M. Churchland and K. V. Shenoy, “Temporal complexity and heterogeneity of single-neuron activity in premotor and motor cortex,” *J. Neurophysiol.*, vol. 97, pp. 4235–4257, 2007.

[53] C. Pandarinath, D. J. O’Shea, J. Collins, R. Jozefowicz, S. D. Stavisky, J. C. Kao, E. M. Trautmann, M. T. Kaufman, S. I. Ryu, L. R. Hochberg, J. M. Henderson, K. V. Shenoy, L. F. Abbott, and D. Sussillo, “Inferring single-trial neural population dynamics using sequential auto-encoders,” pp. 805–815, 2018.

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*,

F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016*, ser. Lecture Notes in Computer Science. Springer, Cham, Oct. 2016, pp. 630–645.

[57] G. Cheng, P. Zhou, and J. Han, "Learning Rotation-Invariant convolutional neural networks for object detection in VHR optical remote sensing images," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 12, pp. 7405–7415, Dec. 2016.

[58] J. Han, D. Zhang, G. Cheng, L. Guo, and J. Ren, "Object detection in optical remote sensing images based on weakly supervised learning and High-Level feature learning," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 6, pp. 3325–3337, Jun. 2015.

[59] D. Zhang, J. Han, C. Li, J. Wang, and X. Li, "Detection of co-salient objects by looking deep and wide," *Int. J. Comput. Vis.*, vol. 120, no. 2, pp. 215–232, Nov. 2016.

[60] J. Han, D. Zhang, X. Hu, L. Guo, J. Ren, and F. Wu, "Background Prior-Based salient object detection via deep reconstruction residual," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 8, pp. 1309–1321, Aug. 2015.

[61] J. Lindsey, S. A. Ocko, S. Ganguli, and S. Deny, "A unified theory of early visual representations from retina to cortex through anatomically constrained deep CNNs," Jan. 2019.

[62] D. L. K. Yamins and J. J. DiCarlo, "Using goal-driven deep learning models to understand sensory cortex," *Nat. Neurosci.*, vol. 19, no. 3, pp. 356–365, Mar. 2016.

[63] L. T. McIntosh, N. Maheswaranathan, A. Nayebi, S. Ganguli, and S. A. Baccus, "Deep learning models of the retinal response to natural scenes," *Adv. Neural Inf. Process. Syst.*, vol. 29, pp. 1369–1377, 2016.

[64] H. Wen, J. Shi, Y. Zhang, K.-H. Lu, J. Cao, and Z. Liu, "Neural encoding and decoding with deep learning for dynamic natural vision," *Cereb. Cortex*, vol. 28, no. 12, pp. 4136–4160, Dec. 2018.

[65] S. Gerwinn, J. H. Macke, and M. Bethge, "Bayesian inference for generalized linear models for spiking neurons," *Front. Comput. Neurosci.*, vol. 4, p. 12, May 2010.

[66] W. Truccolo, U. T. Eden, M. R. Fellows, J. P. Donoghue, and E. N. Brown, "A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects," *J. Neurophysiol.*, vol. 93, no. 2, pp. 1074–1089, Feb. 2005.

[67] A. S. Benjamin, H. L. Fernandes, T. Tomlinson, P. Ramkumar, C. VerSteeg, R. H. Chowdhury, L. E. Miller, and K. P. Kording, "Modern machine learning as a benchmark for fitting neural responses," *Front. Comput. Neurosci.*, vol. 12, p. 56, Jul. 2018.

[68] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, Nov. 2016.

[69] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[70] M. T. Kaufman, J. S. Seely, D. Sussillo, S. I. Ryu, K. V. Shenoy, and M. M. Churchland, "The largest response component in motor cortex reflects movement timing but not movement type," *eNeuro*, vol. 3, no. August, pp. ENEURO.0085–16.2016, 2016.

[71] G. F. Elsayed and J. P. Cunningham, "Structure in neural population recordings: an expected byproduct of simpler phenomena?" *Nat. Neurosci.*, vol. 20, no. 9, pp. 1310–1318, Sep. 2017.

[72] N. Even-Chen, S. D. Stavisky, J. C. Kao, S. I. Ryu, and K. V. Shenoy, "Augmenting intracortical brain-machine interface with neurally driven error detectors," *J. Neural Eng.*, vol. 14, no. 6, p. 066007, Dec. 2017.

[73] W. Wu, M. J. Black, Y. Gao, E. Beinenstock, M. D. Serruya, A. Shaikhouni, and J. P. Donoghue, "Neural decoding of cursor motion using a kalman filter," in *Advances in Neural Info. Proc. Sys.*, 2003.

[74] M. Aghagolzadeh and W. Truccolo, "Inference and decoding of motor cortex low-dimensional dynamics via latent state-space models," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 4320, pp. 1–1, 2015.

[75] J. C. Kao, P. Nuyujukian, S. I. Ryu, M. M. Churchland, J. P. Cunningham, and K. V. Shenoy, "Single-trial dynamics of motor cortex and their applications to brain-machine interfaces," *Nat. Commun.*, vol. 6, p. 7759, Jul. 2015.

[76] J. C. Kao, S. D. Stavisky, D. Sussillo, P. Nuyujukian, and K. V. Shenoy, "Information systems opportunities in brain machine interface decoders," *Proc. IEEE*, vol. 102, no. 5, pp. 666–682, May 2014.

[77] M. M. Churchland, J. P. Cunningham, M. T. Kaufman, S. I. Ryu, and K. V. Shenoy, "Cortical preparatory activity: representation of movement or first cog in a dynamical machine?" *Neuron*, vol. 68, no. 3, pp. 387–400, Nov. 2010.

[78] D. Sussillo, M. M. Churchland, M. T. Kaufman, and K. V. Shenoy, "A neural network that finds a naturalistic solution for the production of muscle activity," *Nat. Neurosci.*, vol. 18, no. 7, pp. 1025–1033, 2015.

[79] L. Paninski, M. R. Fellows, N. G. Hatsopoulos, and J. P. Donoghue, "Spatiotemporal tuning of motor cortical neurons for hand position and velocity," *J. Neurophysiol.*, vol. 91, no. 1, pp. 515–532, Jan. 2004.

[80] M. M. Churchland, G. Santhanam, and K. V. Shenoy, "Preparatory activity in premotor and motor cortex reflects the speed of the upcoming reach," *J. Neurophysiol.*, vol. 96, no. 6, pp. 3130–3146, Dec. 2006.

[81] Q. G. Fu, J. I. Suarez, and T. J. Ebner, "Neuronal specification of direction and distance during reaching movements in the superior precentral premotor area and primary motor cortex of monkeys," *J. Neurophysiol.*, vol. 70, no. 5, pp. 2097–2116, Nov. 1993.

[82] J. Messier and J. F. Kalaska, "Covariation of primate dorsal premotor cell activity with direction and amplitude during a memorized-delay reaching task," *J. Neurophysiol.*, vol. 84, no. 1, pp. 152–165, Jul. 2000.

[83] D. W. Moran and A. B. Schwartz, "Motor cortical representation of speed and direction during reaching," *J. Neurophysiol.*, vol. 82, no. 5, pp. 2676–2692, Nov. 1999.

[84] A. Riehle and J. Requin, "Monkey primary motor and premotor cortex: single-cell activity related to prior information about direction and extent of an intended movement," *J. Neurophysiol.*, vol. 61, no. 3, pp. 534–549, Mar. 1989.

[85] J. A. Pruszynski, I. Kurtzer, J. Y. Nashed, M. Omrani, B. Brouwer, and S. H. Scott, "Primary motor cortex underlies multi-joint integration for fast feedback control," *Nature*, vol. 478, no. 7369, pp. 387–390, Sep. 2011.

[86] D. R. Humphrey, E. M. Schmidt, and W. D. Thompson, "Predicting measures of motor performance from multiple cortical spike trains," *Science*, vol. 170, no. 3959, pp. 758–762, Nov. 1970.

[87] M. Hepp-Reymond, M. Kirkpatrick-Tanner, L. Gabernet, H. X. Qi, and B. Weber, "Context-dependent force coding in motor and premotor cortical areas," *Exp. Brain Res.*, vol. 128, no. 1-2, pp. 123–133, Sep. 1999.

[88] J. Lindsey, S. A. Ocko, S. Ganguli, and S. Deny, “A unified theory of early visual representations from retina to cortex through anatomically constrained deep CNNs,” Jan. 2019.

[89] V. Mante, D. Sussillo, K. V. Shenoy, and W. T. Newsome, “Context-dependent computation by recurrent dynamics in prefrontal cortex,” *Nature*, vol. 503, no. 7474, pp. 78–84, Nov. 2013.

[90] H. F. Song, G. R. Yang, and X. J. Wang, “Training excitatory-inhibitory recurrent neural networks for cognitive tasks: a simple and flexible framework,” *PLoS Comput. Biol.*, vol. 12, no. 2, pp. 1–30, 2016.

[91] W. Chaisangmongkon, S. K. Swaminathan, D. J. Freedman, and X. J. Wang, “Computing by robust transience: how the fronto-parietal network performs sequential, category-based decisions,” *Neuron*, vol. 93, no. 6, pp. 1504–1517.e4, 2017.

[92] G. Hennequin, T. P. Vogels, and W. Gerstner, “Optimal control of transient dynamics in balanced networks supports generation of complex movements,” *Neuron*, vol. 82, no. 6, pp. 1394–1406, Jun. 2014.

[93] P. Mehrotra, S. Dasgupta, S. Robertson, and P. Nuyujukian, “An open-source realtime computational platform (short WIP paper),” *ACM SIGPLAN Notices*, vol. 53, no. 6, pp. 109–112, 2018.

[94] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” Jul. 2017.

[95] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” 2015. [Online]. Available: https://arxiv.org/abs/1506.02438