Towards a Neural Co-Processor Which Restores Movement After Stroke: Modeling a Proof-of-Concept

Matthew J Bryan[1], **Linxing Preston Jiang**[1], **Rajesh P N Rao**[1]

[1] Neural Systems Laboratory, Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA 98105, USA

E-mail: matthew.bryan@dell.com

September 2021

Abstract. *Objective* Brain co-processors[1] are devices which use artificial intelligence (AI) for closed-loop neurostimulation, to shape neural activity and to bridge injured neural circuits for targeted repair and rehabilitation. The co-processor framework offers a flexible approach to learning closed-loop stimulation policies that optimize for (a) specific regimes of neural activity, or (b) external task performance. For example, it may seek to learn to stimulate the motor cortex of a stroke patient, conditioning the stimulation on upstream visual information, aiding the patient's attempt to grasp an object. Through the use of artificial neural networks (ANNs) and deep learning, the co-processor co-adapts with the neural circuit, allowing it to seek optimal stimulation policies, and adapt them as the neural circuit changes. The results presented here demonstrate a neural co-processor for the first time, through the use of a simulation. We explore some of the core algorithms that may allow co-processors to successfully learn and to adapt to non-stationarity in both the brain and sensors. *Approach* We provide the first proof-of-concept of a neural co-processor that leverages deep learning, through the use of a simulated neural circuit. That circuit performs reach-to-grasp task, based on visual input, and is designed to closely resemble a similar circuit in a primate brain [2]. We simulate a variety of lesions by altering the model, and then demonstrate our co-processor's ability to restore lost function through "stimulation" of that model. We further test the ability of our co-processor to adapt its stimulation as the simulated brain undergoes changes. *Main results* Our simulated co-processor successfully co-adapts with the neural circuit to accomplish the external reaching task. The co-processor framework demonstrated here adapts to a variety of lesion types, and to ongoing changes in the simulated brain. *Significance* The proof-of-concept here outlines a co-processor model, as well as our approach to training it, leading to insights on how such a model may be developed for *in vivo* use. We believe this co-processor design will allow for learning complex stimulation policies that help restore function to a stroke victim.

Keywords: brain-computer interface, neural co-processor, ai, machine learning, stimulation

1. Introduction

Aided in part by application of advanced AI techniques, brain-computer interfaces (BCIs) have made advancements over the last several decads, allowing for decoded brain signals to be used for control of a wide variety of virtual and physical prostheses [3, 4, 5, 6]. Separately: advances in stimulation techniques and modeling have allowed us to probe neural circuit dynamics (e.g. [7]) and learn to better drive neural circuits towards target dynamics, by encoding and delivering information through stimulation [8, 9, 10, 11, 12, 13, 14, 15]. Recently, there has been increasing interest in building on these advances to combine decoding and encoding in a single system, for closed-loop stimulation of a neural circuit. Bi-directional BCIs (BBCIs) allow stimulation to be conditioned by decoded brain activity as well as external sensor data (e.g. camera), which can allow for the application of real-time, fine-grained control of neural circuits and prosthetic devices, e.g. Nicolelis et al. [16]. These may lead, for example, to neuro-prostheses that are capable of restoring movement which was lost due to traumatic brain injury (TBI), to a degree not previously possible.

Motivated by that progress, we demonstrate here a flexible framework for combining encoding and decoding, which we term "neural co-processors" [1]. Neural co-processors leverage AI and deep learning to identify optimal, closed-loop stimulation patterns. The approach is flexible enough to optimize not only for particular neural activities, but also for tasks external to the subject. For example, they may be able to aid a stroke victim by finding a stimulation pattern of the motor cortex which helps restore lost limb function. Likewise, the framework generalizes enough to condition stimulation on both brain activity, and external sensors, e.g. cameras or LIDAR, in order to incorporate feedback for realtime control.

Additionally, the co-processor framework allows a neuro-prosthesis to actively adapt to a neural circuit as it changes with time. This framework is capable of co-adapting with the circuit, i.e. brain, by updating its stimulation regime, while at the same time the brain is updating its response to the stimulation, and changing due to natural plasticity, aging, etc. This allows the co-processor to continually optimize for the intended cost function, despite the signficant non-stationarity of the target circuit.

Here we provide a proof-of-concept in simulation for a co-processor that restores movement to a limb, after a subject has suffered a stroke affecting its ability to use that limb. It combines:

- A stimulation model, which models the relationship between decoded brain activity, stimulation, and task performance.
- An AI agent which determines the stimulation to apply in a closed-loop fashion, in real time.

Significant advances have been made in modeling the effects of electrical stimulation of the brain, some of which can be leveraged for our co-processor design, as we outline below. Researchers have explored how information can be biomimetically or artificially

encoded and delivered via stimulation to neuronal networks in the brain and other regions of the nervous system for auditory [8], visual [9], proprioceptive [10], and tactile [11, 12, 13, 14, 15] perception. Advancements have also been made in modeling the effects of stimulation over large scale, multi-region networks, and across time [17]. Some have additionally designed models which can adapt to ongoing changes in the brain, including changes due to the stimulation itself [18]. In our proof-of-concept outlined below, we will use a stimulation model, not unlike those cited here, which seeks to account for both network dynamics and non-stationarity. In addition to training the model to have a strong ability to predict the effect of stimulation, we additionally train it to be useful for then learning an optimal stimulation policy, which is a property somewhat distinct from predictive power alone.

Advances have also been made in both open- and closed-loop stimulation for treating a variety of disorders. Open loop stimulation has been effective in treating Parkinson's Disease [19], as well as various psychiatric disorders [20, 21, 22]. More directly related to this paper, we see in Khanna et al. [23], the use of open loop stimulation in restoring dexterity after a lesion occurs affecting a primate's motor cortex. The authors demonstrate that the use of low-frequency alternating current, applied epidurally and set to certain phases, can improve grasp performance.

While open loop stimulation techniques have yielded clinically useful results, their results in many domains have been mixed, such as use in visual prostheses [24], and use in invoking somatosensory feedback [15]. Likely this is due to the stimulation not being conditioned on the ongoing dynamics of the circuit being stimulated. Moment-to-moment and throughout the day, the circuit will respond differently to the same stimulus, as a result of differing inputs and ongoing activity. Stimulation therefore needs to be proactively adapted in response. This need is even greater over longer time scales as the effects of plasticity and ageing change the connectivity of the brain.

Closed-loop stimulation conditions stimulation on observations of brain activity, possibly allowing it to shape the neural activity more precisely, and to adapt to changes in the circuit over time. This opens the door to real-time, targeted control of the neural circuit. It has been used to aid in learning new memories after some impairment [25, 26], to replay visually-invoked activations [18], and for optogenetic control of a thalamocortical circuit [27], among others.

Something that remains unclear is how to leverage closed-loop control for real-time co-adaption with the brain to accomplish an external task. "Co-adaption" here refers to the ability of a neuro-prosthesis to adapt its stimulation regime to the ongoing changes in the circuit it is stimulating, and to adapt with that circuit to accomplish the external task, such as grasping. The neural co-processor we present here provides one potential model for accomplishing that. Through the use of deep learning, the co-processor model we present co-adapts an AI agent, which governs the stimulation, with both a stimulation model, and the neural circuit being stimulated.

For a neurologically complex task such as grasping, we cannot identify *a priori* a real time controller of the neural circuit. That is due in large part to the variability of

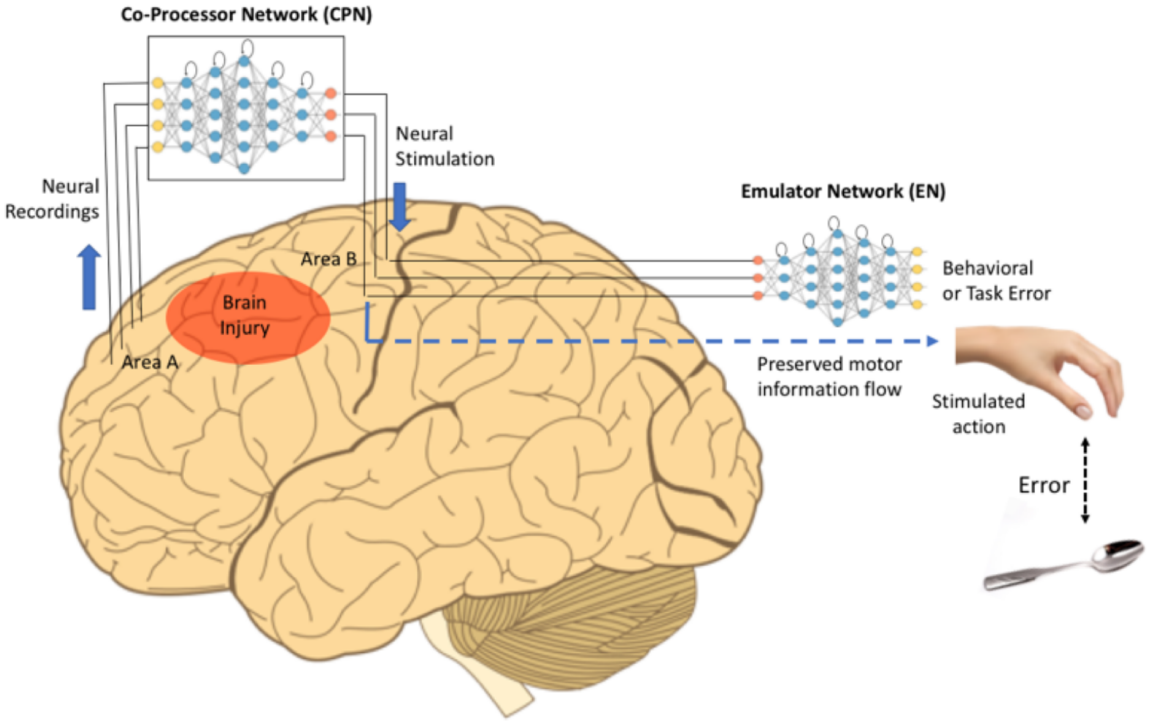


Figure 1: Using a co-processor to drive external task performance after a traumatic brain injury

circuits from subject-to-subject, as well as variations in the placement of sensors and stimulators in the brain. The only plausible path to such a real time controller is to parameterize it in a subject- and time-specific way. Our model seeks to accomplish that using deep learning, together with a data-efficient approach to training.

Before attempting *in vivo* experiments using such a model, we first demonstrate here a number of crucial design elements of it, through the use of a simulated grasping circuit, presented previously by Michaels et al. [2]. We explore:

- The properties of the artificial neural networks (ANNs) that are needed to successfully adapt to the long-running dynamics of a stimulated neural circuit, as well as to adapt to that circuit's ongoing changes.
- Data-efficient methods for training these models, to better ensure we can train them with a biologically-realistic amount of data.
- How we must train our stimulation model to make it effective in later training our stimulation agent.

2. Method

2.1. Architecture Overview

First, we present the architecture of our co-processor design. This design aims to solve two fundamental challenges in using neural stimulation to improve external task performance. First, neural networks exhibit long-running and nonlinear dependencies, necessitating the need for a stimulation agent to account for far-distant effects of the stimulation it applies. Second, with neurologically complex tasks, such as grasping, the mapping between neural activity as-measured and the external task cannot be determined *a priori*. As a result, the co-processor must somehow learn what stimulation is appropriate for aiding the user in the external task they are attempting to perform.

Our co-processor attempts to solve these with a pair of artificial neural networks:

- A stimulation and neural dynamics model, known as an "Emulator Network" (EN). It models the relationship between the stimulation, neural dynamics, and external task. It's purpose is for training the second network.
- A stimulation agent, known as the "co-processor network" (CPN), which maps neural activity, and possibly data from external sensors, to stimulation parameters.

We co-train the EN and CPN, with the goal of training a CPN whose output stimulation parameters improve task performance. By continually training both, they adapt to the brain as it changes, effectively allowing for brain-stimulator co-adaptation. The EN is a tool for training the CPN, giving us a way to back-propagate task error to the CPN. It outputs task-relevant metrics - a prediction of muscle velocities in our case - given measurements of neural activity, and the stimulation parameters output from the CPN. If the EN is trained in a particular way, and to a sufficient level of precision, it can be used as a function approximator relating stimulation and a task, and do so in a way that allows us to train the CPN with it. When training the CPN, we in-effect treat the EN's output as the true task performance, or a related metric, and then backpropagate the loss defined in terms of that metric in order to train the CPN. See Fig. 1. We will illustrate the details of the training algorithm below.

In our present demonstration, the EN is constituted as a single layer, fully connected, long short-term memory (LSTM) recurrent neural network (RNN), with hyperbolic tangent ($tanh$) activations, and a linear readout. The general notion of a co-processor does not require this precise architecture, but for our example, we found that the LSTM approach allows the network to continuously adapt to long-running dependencies in the simulated neural dynamics, far better than a vanilla RNN. The CPN is constituted as an almost identical network, though with a different dimensionality, as we explain below. There is no strict requirement for the EN and CPN to be so similar, but we found this simple architecture to work well in our example.

Note that the EN effectively constitutes a stimulation model. Its design is somewhat motivated by the common linear time-invariant state space model of stimulation, as in e.g. Yang et al. [17], which is a helpfully simple (linear) approach. That approach seeks

to model neural network dynamics in terms of a stationary linear model, which, due in part to its simplicity and the number of techniques developed for it, lends itself to useful interpretations and analyses of those neural dynamics.

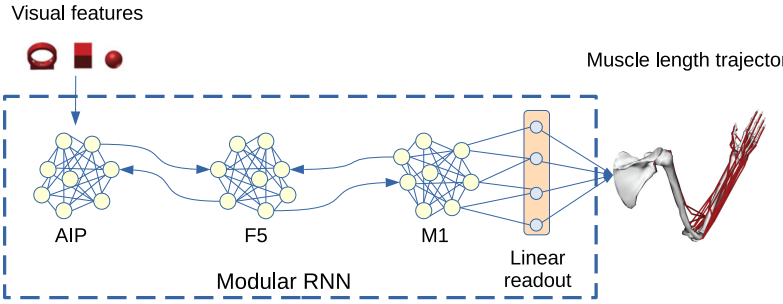
There are some key differences between that approach and the EN architecture we present here. Our EN's purpose is simply to train the CPN, so we pick an architecture we find to be best suited to that job. First, our EN architecture is not linear, through the use of a *tanh* activation function. That allows us to escape some of the limits in representational power of linear models, though we didn't deeply test a strictly linear approach. Second, LSTM cells are designed to better capture long-running network dynamics, and in our case we found that they were crucial for that reason: when using vanilla RNN cells, our co-processor wasn't able to train well. Compare that to the simpler cells commonly used in vanilla RNNs, which yield functionally the same approach as the linear state space model, less the use of a nonlinear activation function.

2.2. Simulation Overview

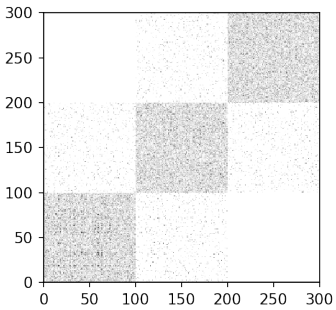
We demonstrate our approach here using a simulated grasping circuit. A detailed simulation such as this allows us to explore some of the critical architectural details and training algorithms for a co-processor of this type. By first doing such exploration in simulation, we are able to rapidly and cheaply iterate on our design, prior to any *in vivo* experiments. In order for the simulation to be admissible, we need to ensure it has some properties that allow it to strongly indicate if our design is improving in a direction that will later allow for real deployments. Otherwise, our design may be adapting to the pecularities of the simulation, without becoming more useful for the real world.

An example of such a simulation approach is Bernal et al. [28]. In this work, the authors use a simulated spiking neural network to train a stimulation agent, much like the work we present here. Their stimulation agent sought to restore the network's control of a simulated arm, to reach a target, after a simulated lesion was applied. As they point out, we have a limited ability to probe a neural circuit *in vivo* in order to perform learning. As a result, we first need to design our approach through the use of an admissible simulation. The authors in this case simulated lesions by effectively removing parts of their simulated network, or by cutting connections between parts of the network. Likewise here, we use an established design for an artificial neural circuit which performs a grasping task, whose architecture and training methods were designed specifically to result in naturalistic dynamics.

The simulated circuit, from Michaels et al. [2], was trained to resemble the grasping circuits of monkey subjects engaged in a delayed reach-to-grasp task. Its design draws on a body of literature focused on architectures and training methods for RNNs which seek to create artificial neural circuits that have activation dynamics similar to natural circuits, including delayed grasping tasks [29]. The Michaels circuit consists of a "modular" vanilla RNN (mRNN), and a linear readout layer. Each "module" consists of 100 vanilla RNN neurons, with a non-linearity applied on the outputs. The modules are



(a) Michaels Modular RNN (mRNN), a simulated grasping circuit. The emergent dynamics of the three modules correspond well to natural neural activity measured from primate AIP, F5, M1 regions, respectively, from the same task. Visual (VGGNet) features forward propagate through the network, conditioning the grasp for the object of the particular size, shape, and location.



(b) Connectivity matrix J. Note the within-module connections (on the diagonal) are fully connected, and connections to adjacent modules are sparse 10%.

Figure 2: Architecture of the Michaels mRNN

internally fully connected, and are connected to each other sparsely (10% connectivity). The inputs are visual features intended to capture the view the monkeys had during the task, specifically VGGNet features from 3D renderings of the same objects which the monkeys grasped. The outputs are muscle length velocities for the should, arm, and hand of the monkey during the trial. The natural velocities were captured with a motion capturing glove, and the artificial neural network was trained to recapitulate those grasping motions. Data and trained models from this work were supplied to us by the lead author Jonathan Michaels, for the purpose of our present simulation. We re-implemented his models' logic in PyTorch, and loaded his trained parameters into it, for one of his subjects, arbitrarily chosen. See Fig. 2.

The design of the circuit is intended to resemble a vision-to-grasp pipeline, essentially representing the visual processing needed to reach the hand to the appropriate position for the grasp, and to form the hand properly for grasping the particular shape of object. The emergent dynamics of the artificial network's "modules", once trained, correspond roughly to the AIP, F5, and M1 portions of the monkey subjects' brains. 'Correspond' here, notably, refers to the fact that the activity of the module receiving the visual inputs resembles the AIP activity of the monkey subject from the same trials. Likewise, the second and third modules resemble the natural activity from the monkey's F5 and M1 regions, respectively. The emergent network dynamics show a number of other correspondences to the monkeys' natural brain activity as well, as detailed in the paper.

Importantly, the simulated circuit's activity shows a relatively clear separation of the object shapes. That is - the visual information input to the network that differentiates one trial from another is leveraged by the circuit to properly condition the hand shape trajectory for grasping the object of that trial's particular shape, size,

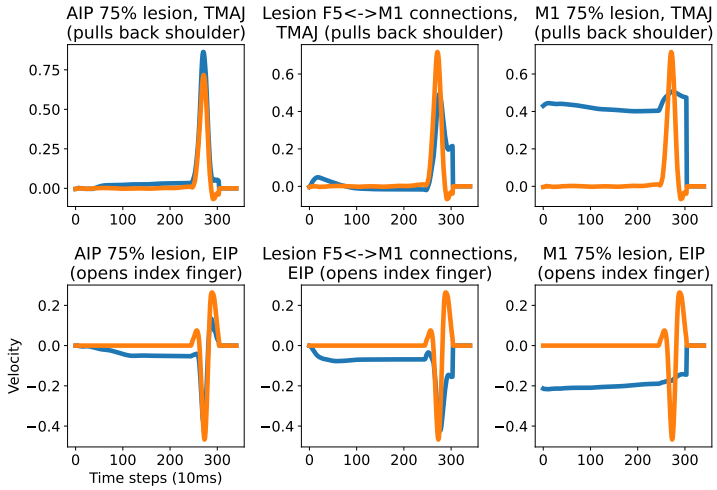
and location. The visual information forward-propagates through the network, through successive processing steps, where it is eventually leveraged to recapitulate the grasp appropriate for the object represented by the input visual features. It follows, then, that the classes are separable in some way by observing the distinct ways that the mRNN is activated by the classes' corresponding visual inputs. As we will note below: in the absense of this visual information forward propagating, the circuit can at-best learn a loss-minimizing grasp, stereotyped across all object sizes and shapes. Such a grasp bears some resemblance to all grasps the given monkey performed, due to all trials being a reach-to-grasp preceded by a waiting period, but performance suffers drastically.

2.2.1. Simulated lesions cause real world failure modes Simulating a brain lesion in terms of this artificial network results in error modes that resemble a natural lesion of certain regions of a primate brain. For example, if we alter the network by zeroing the outputs of some of the first (input, or AIP) module's neurons, the reaching motion generally succeeds, but the finger muscle velocities show a high degree of error - effectively implying that the subject can somewhat reach to grasp, but cannot form a grasp appropriate for the object. That suggests that losing a portion of the cortical machinery needed to translate or forward-propagate object shape information to movement-related cortex can result in a reduced ability to form the hand properly for the grasp, though positioning of the hand may still roughly succeed. Muscle spasticity of the hand is also a common symptom of certain strokes in primates, and indeed many who suffer from it are still able to position their hand, even while being unable to form it properly for a grasp [23]. In that sense, the error mode of this simulated lesion closely resembles natural stroke symptoms, though the simulation is not intended to constitute a physical model of a lesion, i.e. to directly explain the connection between hand spasticity and the lesion. Conversely, if we lesion of a portion of the output module of the Michaels mRNN, roughly corresponding to M1, we see a more wholesale loss of movement, affecting even the ability to reach for the grasp. Finally, if we "disconnect" communications between the F5 and M1 modules, we see a failure similar to an AIP lesion: movement is generally achieved, but we see a disproportionate impact on hand pose. See Fig. 3 for examples.

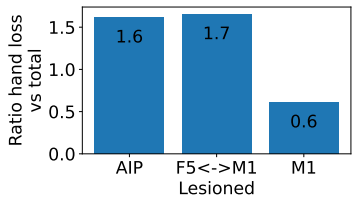
The co-processor's task in our simulation will be to identify the appropriate information, read from the mRNN's activations, for conditioning the stimulation. The co-processor seeks to effectively bridge across the lesion, forward propagating the object shape information, indirectly, through the use of stimulation.

In our experiments below, we demonstrate our co-processor design on three types of simulated lesions:

- **Output-based (AIP):** we force the output of some proportion of "AIP" neurons to zero, effectively removing them from the network. This results in a loss of object shape information and, as we will see below, causes the co-processor to be unable to differentiate objects.



(a) Example muscle trajectories during a single trial, for one hand and shoulder muscle



(b) Ratio of hand muscle MSE losses vs. overall MSE losses, for different lesion types

Figure 3: Lesion designs which prevent forward propagation of object shape information differentially impact hand pose. Loss here is an L2 distance measured relative to the circuit's trajectories prior to the lesion.

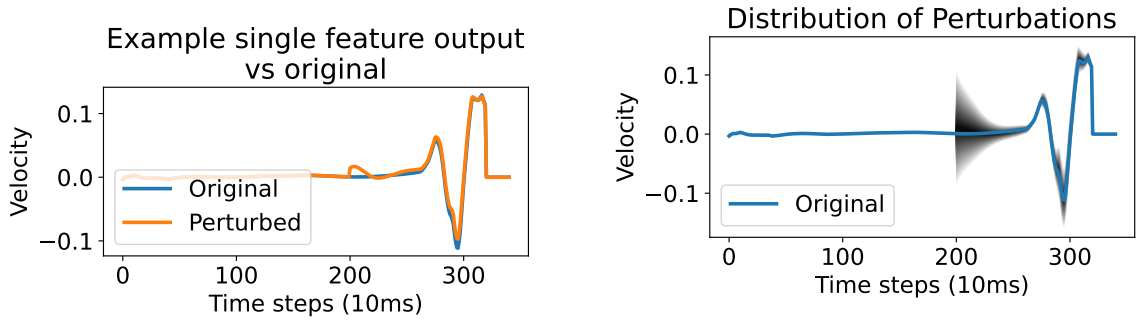
- **Connection-based:** we prevent the forward and backward propagation of information between the "F5" and "M1" modules, effectively representing a severing of the connections between the two.
- **Output-based (M1):** we force the output of some proportion of "M1" neurons to zero, causing a more holisitic loss of movement.

2.2.2. Simulated network exhibits long running dynamics Just as the co-processor must adapt to the lesion, it must also adapt to the dynamics of the network it is stimulating. Natural neural networks as well as our simulated network exhibit long running dynamics, which our CPN and EN must account for in their learning. A perturbation of a network (i.e. due to stimulation) will cause changes in neuron activations long after the stimulation is applied, sometimes far from the site of stimulation. Our simulated grasping circuit exhibits the same behavior.

To illustrate this, suppose we applied a small, one-time, instantaneous perturbation of the hidden state of 10 randomly chosen neurons in the output module at some point in time during a trial. If we repeat that experiment many times, we can see what the distribution of long- running effects tends to look like.

In Fig. 4 we can see that even a single, one-time perturbation in the network has effects dozens of time steps later. Our co-processor will need to account for these, since stimulation is intended to cause perturbations in a network. To successfully aid in grasping, the co-processor will need to account for these dynamics. As we will show in the next subsection, the problem the co-processor faces is in-fact even harder than this,

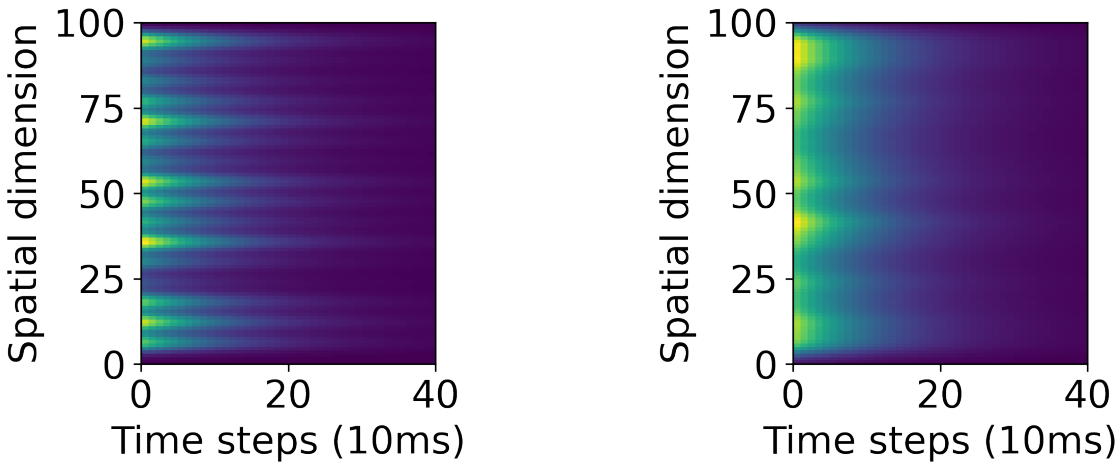
due to a stimulation model that includes both spatial and temporal smoothing.



(a) Example trial

(b) Distribution of effects across n=1000 samples

Figure 4: Instantaneous perturbations of a random sample (n=10) of M1 neurons at time t=200 results in long-running effects on output.



(a) $\sigma = 1.875$

(b) $\sigma = 2.875$

Figure 5: Spatial and temporal smoothing of a single 16 dimensional, randomized θ, onto 100 simulated neurons. Color values indicate the magnitude of the value summed into each neuron's hidden state in that time step. After $t = 0$, θ is the zero vector.

2.2.3. Stimulation model In order to inject another aspect of realism into our simulation, we subject the co-processor to a stimulation model, rather than allowing it to directly influence the hidden state of the simulated network's neurons. Our stimulation model includes aspects of both spatial and temporal smoothing.

In our experiments, we stimulate only the output module, since this co-processor's purpose is to improve external task performance, which it is able to do with only

stimulation of the output module of the network. It is conceivable that a co-processor could stimulate other areas of the brain to improve task performance downstream, or to probe the brain to better reveal the user's intent (i.e. object shape), but we did not explore those possibilities in this paper.

The stimulation function S receives as inputs the stimulation parameters θ provided by the CPN, which it accumulates in an internal memory. Each time step, it outputs a modifier to the internal states of all neurons in the output module, based on the current state of memory in the function object. That memory allows the stimulation function to perform temporal smoothing. Specifically, we used a simple exponential decay model where the function reads its current state of memory, sums in the new parameters, and then decays each memory element towards 0.0 at some rate. Roughly speaking, this design intends to approximate the notion of dissipation: the effect of stimulation is not instantaneous, but rather decays with time, as charge dissipates into the surrounding area.

Likewise, the stimulation function maps the relatively low dimensional stimulation parameter vector θ onto the M1 cells which it stimulates. In our experiments, our stimulation parameters θ had 16 dimensions, which we found was sufficient to allow significant improvement in task performance. The stimulation function uses a simple Gaussian smoothing to map those parameters onto the simulated M1 neurons. Each parameter represents, in a sense, an electrode located along a single spatial dimension, whose stimulation affects the neurons in its vicinity more than it affects others, according to each neuron's Mahalanobis distance from it. The neurons are aligned along that dimension arbitrarily. We fix the width of the Gaussians arbitrarily to $\sigma = 1.75$. Note that we do not attempt to model the physical layout of our simulated neurons, but simply want to disallow our co-processor from having an unrealistically high degree of spatial resolution.

Thus, the governing equations of the stimulation become:

$$\alpha_t = \tau\alpha_{t-1} + \theta_t \tag{1}$$

$$s_t = C(\alpha_t) \tag{2}$$

- α: the 16 dimensional internal activation, or memory of our stimulation
- τ: our decay rate, which we set arbitrarily to 0.7
- C: a precalculated $100x16$ matrix which provides our Gaussian smoothing
- s_t: the stimulation we apply to each neuron at the given time step.

The governing equations of the simulated network then become:

$$x_{t+1} = Jx_t + Iv_t + s_t \tag{3}$$

$$a_t = tanh(x_t) \tag{4}$$

$$y_t = La_t + b \tag{5}$$

- x: the hidden state of each neuron
- J: the recurrence weight matrix
- I: the input response matrix
- L, b: the parameters of the linear readout
- y: the output of the network

See Fig. 5 for a visual depiction of an example where θ is non-zero at $t = 0$, and zero for all other times, in order to see how stimulation is applied across space and time. In Fig. 6, we see stimulation applied during the simulation, in a real trial. In this case, the trial is one where 50% of M1 neurons are lesioned to have zero output. Observations of the network's hidden state (explained in the next section) are taken from the first two modules, and stimulation is applied to the last module. This fact is intended to capture the difficulties with observing the same cortex that you are stimulating, due to stimulation artifacts. The trial we show depicts stimulation supplied by a highly trained co-processor, where task performance has been largely restored. We explain this experiment in more detail below.

2.2.4. Observation model Our observation model likewise relies on a notion of electrodes spread along a single spatial dimension. As with stimulation, this is not intended to capture true physical relationships between neurons and electrodes, but rather to act as a dimensionality reduction that isn't designed to specifically to favor task success, such as PCA. We treat the neurons of each module as occupying their own space, with 20 electrodes arrayed along each.

We use a similar Gaussian-based approach as with stimulation, but in this case our approach looks much like a Gaussian convolution, where the Gaussian kernel is centered at each electrode position. Each 'electrode' is read out as a weighted average of all neurons in the given module, with weights being provided by the Gaussian kernel. As above, refer to Fig. 6 for a visual example.

2.2.5. Brain co-adaptation To demonstrate the co-processor's ability to adapt to the non-stationarity of the brain, we also change it throughout the co-processor's training. Specifically, we simulate the brain's co-adaptation with the stimulation to cooperatively solve the problem, as the stimulation is also changing through the CPN's training.

We achieve this through a simple error backpropagation, using PyTorch's implementation of the *Adam* optimizer. With each trial, task loss is calculated and backpropagated into the mRNN, allowing it to adapt to its inputs, one of which is the stimulation being applied. The learning rate for the optimizer is set arbitrarily to a relatively low rate of $1e-7$. The learning rate was not chosen in a principled way, since we are not aware of a principled way of choosing it. However, we reason that it should be lower than the co-processor by orders of magnitude due to the rapidity by which an ANN can be retrained.

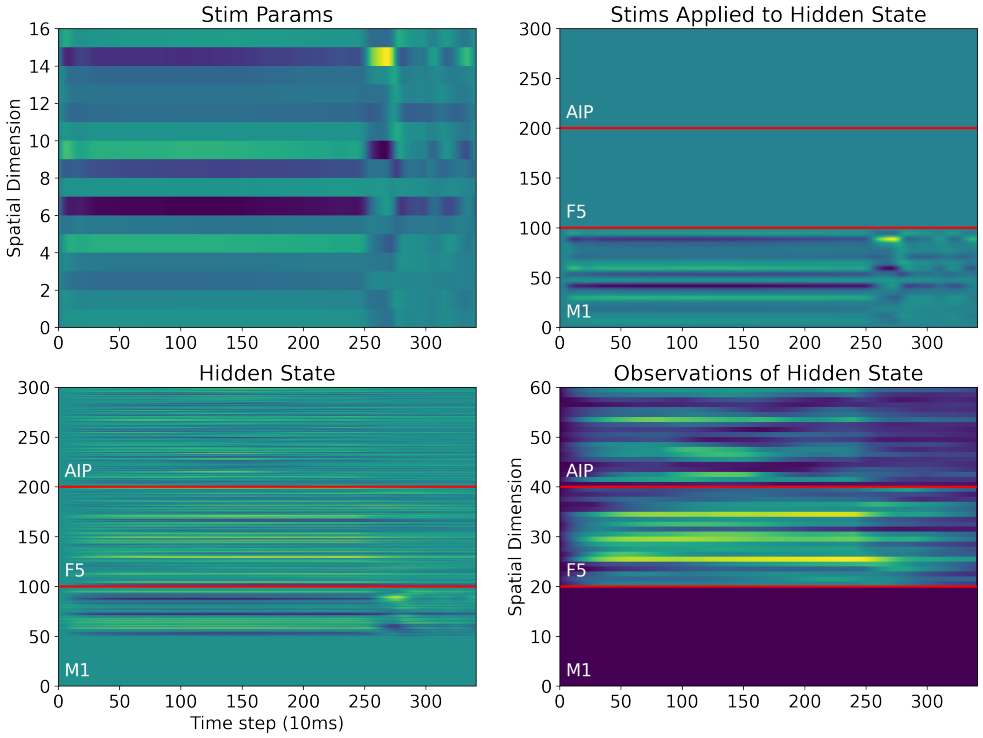


Figure 6: Example of stimulation and observation for a single trial. Here, M1 has been lesioned 50%, as seen in the zero (light blue) hidden states. We gather observations only from the AIP and F5 modules, as we explain in Section 3.1. Stimulation is applied to M1, to drive the network output.

2.2.6. Simulating recovery prior to co-processor use We sought to additonally pre-train the mRNN after applying the lesion in order to simulate stroke recovery. For simulated lesions which zero the outputs of neurons, the mRNN is, unfortunately, not an appropriate model. We know that strokes can cause damage from which the subject, in many cases, cannot recover. The mRNN model has sufficient redundancy built into it that lesioning the outputs of neurons leaves enough remaining degrees of freedom that a nearly full recovery can occur, unless so many neurons are lesioned that no stimulation could be effective. We go into this further in the Discussion section below.

We can, however, simulate pre-recovery in the case of a lesion that excludes communication between the F5 and M1 modules. In that case, object shape information cannot propagate forward in the network, in order to condition the hand for grasping. The mRNN can, in pre-recovery, learn a stereotyped grasp, at which point the co-processor's job will be to forward-propagate that information. We explore this in one of our experiments, explained below.

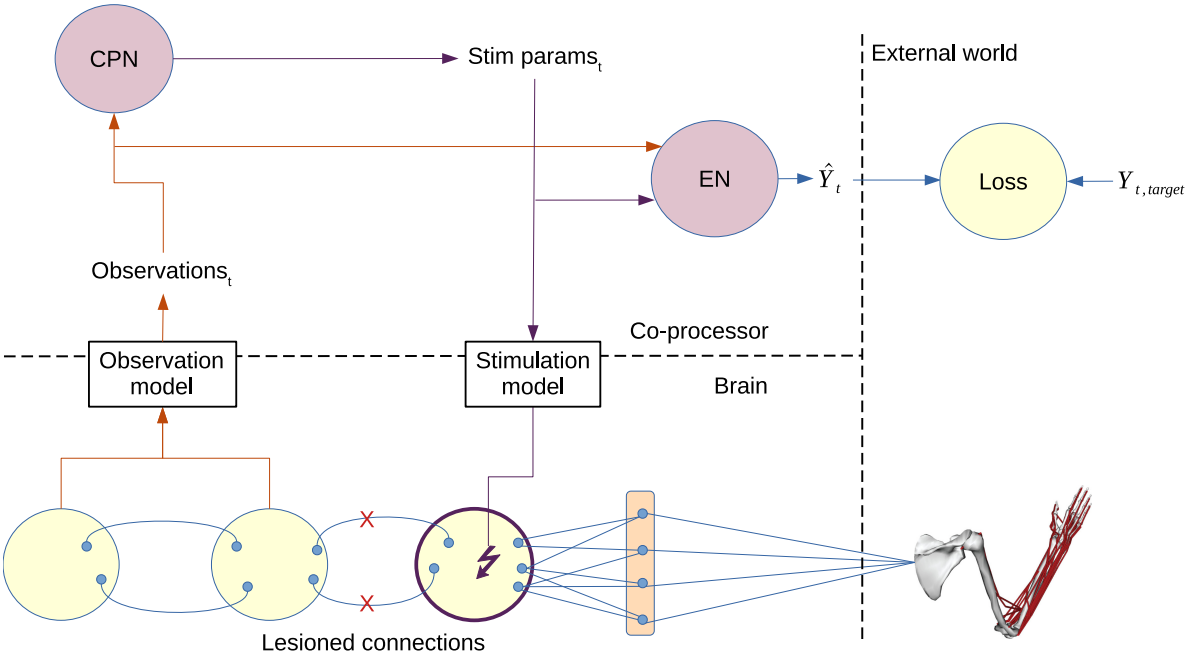


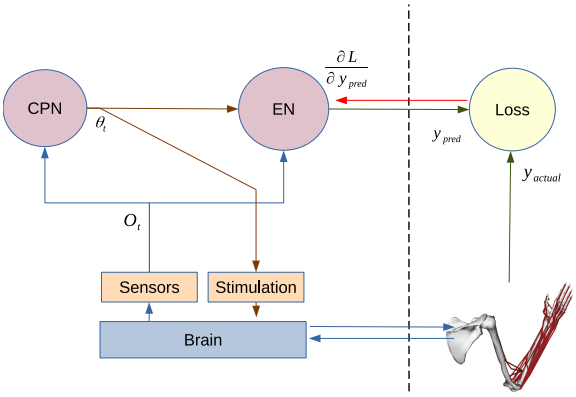
Figure 7: Experiment architecture overview

2.3. Training Algorithm

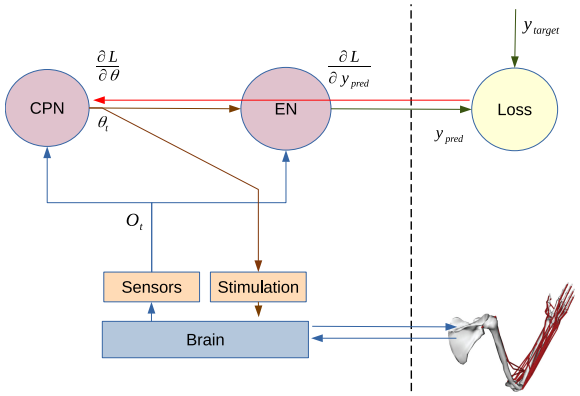
In all experiments, training the co-processor requires a careful interleaving of EN and CPN training epochs. EN training epochs concentrate on updating the EN, based on observations of the effect of stimulation on the simulated network's output. The loss function when training the EN is the MSE loss between the actual and predicted output of the mRNN (i.e. muscle velocities). We then use the EN to train the CPN during the CPN training epochs. During that time, we backpropagate the MSE loss between the EN's predicted output, and target output, to the CPN. See Fig. 8.

2.3.1. EN training For the EN to be useful, it must accurately predict the effect of stimulation produced by the CPN. In addition to that, backpropagating through it must yield gradients which train the CPN to output more useful stimulus. We discovered that this latter property does not necessarily occur simply by virtue of the former. We have found that an EN can be trained to extreme levels of predictive power, even on white noise stimulation - to orders of magnitude lower loss than the task loss - while at the same time backpropagation through it yields gradients which are not useful for training the CPN. As a result, the CPN exhibits unstable training. We have thus far been unable to discover a way to measure and optimize for this second property, though pursuing it may be a fruitful subject of future ML research. Instead of attempting to train an EN directly to exhibit this property, we discovered a training regime that reliably makes it appear.

The central concept of our training regime is to provide a carefully chosen variety of training examples, and to use heavy weight regularization, in order to force the EN



(a) EN training phase: backpropagate actual vs predicted



(b) CPN training phase: backpropagate actual vs target MSE loss, via EN, to CPN. This effectively treats the EN output as the actual, and thus the EN must be trained first.

Figure 8: Two Phase Training Regime

to generalize well to stimulation which it hasn't yet seen. We hypothesize that the crux of our EN training problem is one of over-fitting: that the EN may be trained to great predictive power on a set of stimulation examples, but that the effective dimensionality of the problem it is learning is comparatively quite high, due to both the dimensionality of our stimulation parameters θ, and the dynamics of the network being stimulated. As a result, the hypersurface which it has fit to the training examples may exhibit gradients not reflective of reality.

First, and most significantly, we structure the training data set for each EN training epoch in a very specific way. Each epoch, we include examples from the current CPN, the current CPN with Gaussian-distributed mean-0 noise added to its parameters, and examples of white noise stimulation. We developed this structure under the believe that we needed to cover a sufficient variety of examples to prevent overfit, and to do so in a way that emphasizes the neighborhood in CPN parameter space of the current CPN. We initially attempted to pad our dataset with white noise examples alone, but that was not sufficient to stabilize CPN training, despite the EN's ability to reach a high predictive power. Adding in the addition of samples from the parameter space near the current CPN appears to help ensure that the EN is localized enough to give useful gradients for that area. As we will explain below, this produces an additional problem that the EN is to specialized to the local area of CPN parameter space, but we then solve that problem using EN retraining, interleaved with the CPN, as we will explain further below.

Thus, we composed each batch of training data in the following way:

Second, we use of PyTorch's *AdamW* optimizer, which includes an option for weight regularization, and a carefully chosen learning rate schedule. Our intuition is to leverage regularization as one of the standard mitigations for overfit. To converge, the learning rate schedule also needs to be chosen carefully: a relatively high learning rate at first,

Source	**Proportion of dataset**
Examples produced by the current CPN	10%
Examples produced by a copy of the current CPN, with mean-0 random noise added to its parameters	60%
Examples of white noise stimulation	30%

dropping to a very low rate soon thereafter. Using too-low or too-high of a learning rate in any phase causes the EN learning to not converge. EN training proceeds until it reaches a threshold of prediction error, defined as a fraction of the current CPN's task loss.

2.3.2. CPN training Once we discovered an effective way to train an EN, CPN training was straightforward. For this phase, We generate training examples using the current CPN alone. The EN generates predictions of the effect of those stimulations, and we backpropagate through the EN to generate training gradients for the CPN. As with the EN, we constitute the dataset with a large random sample of trials from the original Michaels task. See Fig. 8.

As with the EN, the choice of learning rate schedule for training the CPN is somewhat important. The CPN appears to train in two phases. In the first phase, it is largely learning the structure of the reach-to-grasp task, e.g. that the muscles need to stay still until the reach begins. During that phase, the learning rate can be quite high. Later, the CPN begins to learn the mapping between the object shape information observed from the simulated brain, and how that should map onto stimulation. That phase of the training takes much longer, and requires a learning rate 2-3 orders of magnitude lower than in the first phase. We will cover this in some more detail in Section 3.

2.3.3. Interleaved CPN/EN training, and adapting to non-stationarity Having defined the training epochs for the EN and CPN, we can define a training algorithm on the basis of those two. We train the two in alteration, creating a new EN each time we enter an EN training period. We explored the possibility of reusing an existing EN by retraining it, for the purpose of training efficiency. However, we found that retraining an EN was no more fast than training a new one, and in-fact usually took longer, despite significant experimentation with learning rates.

The key challenge we face is determining when an EN is no longer suitable for training the current CPN. Expiring an EN at the right time accomplishes two things. First, it ensures that our current EN is trained properly for the current CPN; without that, the CPN training will become unstable. Second, expiring the EN is how we adapt our learning to the brain's non-stationarity. At some point, the brain will have changed sufficiently for our EN to be out-of-date, and therefore requiring replacement.

We solve this challenge through a simple set of metrics, which, when satisfied,

indicate we need to transition to an EN training epoch. Our first metric simply captures the EN's declining predictive power. When the EN's prediction error reaches a point which is sufficiently above some ratio to the CPN's task loss, we expire the EN immediately. Second, we expire the EN if CPN loss does not improve (or gets worse) across a sufficient number of recent training steps. This resembles common stopping conditions in iterative learning: one stops when learning is no longer improving the model. Together, these two metrics appear effective in ensuring we expire the EN at the right time.

Unfortunately, because we expire the EN rather than retraining it, and because we need to expire it so often, training is less efficient than it otherwise could be. We explore this some in Section 3. In the next subsection, we illustrate one way we mitigate training inefficiency.

TODO: pseudocode

2.3.4. Optimization: data reuse ...

3. Results

3.1. Experiments

Using the simulation described above, we perform three experiments to demonstrate the co-processor's ability to learn:

- 50% AIP loss
- 100% connectivity lesion between the F5 and M1 modules
- 50% M1 loss

Additionally, we perform each of the ...
Summary of tunables:

- Learning rate schedules
- Activation functions
- Obs function dimensionality
- Stim dimensionality
- Dimensionality of EN, CPN

- Task performance improves drastically.
  - Fig: normalized task loss vs CPN training epoch
  - Fig: normalized task loss vs training epoch
  - Fig: normalized task loss vs training epoch
  - Split by experiment
- Fine tuning takes a long time
- Object classes separate. Fig: sep vs experiment vs epoch

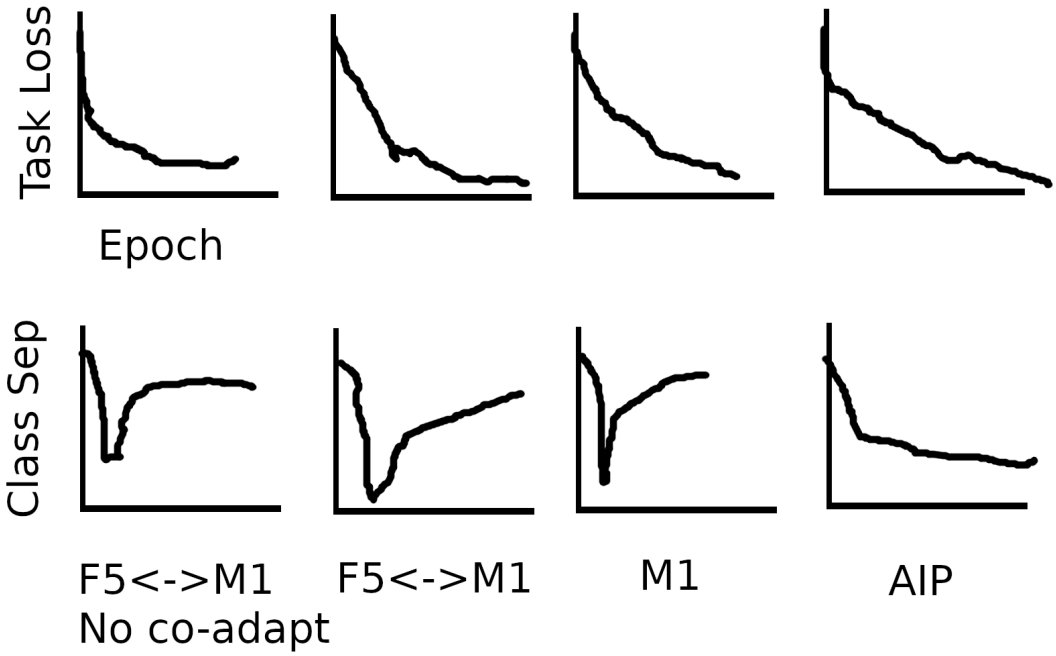


Figure 9: Loss and class separation vs training epoch

- Can co-adapt with the brain, as it changes
- Training efficiency analysis. Show num epochs with and without recycling.

4. Discussion and Conclusion

- Training efficiency
- Spectrum from simple low dimensional stimulation vectors today to higher dimensional future
- Toward in vivo application

5. Acknowledgements

Ganguly, Priya, Anca, Justin, Luciano

6. Ethical Statement

asdf

7. References

[1] RPN R 2019 *Current Opinion in Neurobiology* **55** 142–151

[2] Michaels J, Schaffelhofer S, Agudelo-Toro A and Scherberger H 2020 *Proceedings of the National Academy of Sciences* **117** 32124–32135 ISSN 0027-8424 (*Preprint* https://www.pnas.org/content/117/50/32124.full.pdf) URL https://www.pnas.org/content/117/50/32124
[3] Rao R 2013 *Brain-Computer Interfacing: An Introduction* (Cambridge University Press) ISBN 9780521769419
[4] Wolpaw J and EW W 2012 *Brain-Computer Interfaces: Principles and Practice* (Oxford University Press)
[5] Moritz C, Ruther P, Goering S, Stett A, Ball T, Burgard W, Chudler E and Rao R 2016 *IEEE transactions on bio-medical engineering.* **63**(7) 1354–1367
[6] Lebedev M and Nicolelis M 2017 *Physiological reviews* **97**(2) 767–837
[7] Walker E e a 2019 *Nature Neuroscience* **22**(12) 2060–2065 URL https://doi.org/10.1038/s41593-019-0517-x
[8] Niparko J 2009 *Lippincott Williams and Wilkins* (Oxford University Press)
[9] Weiland J and Humayun M 2014 *IEEE transactions on bio-medical engineering* **61**(5) 1412–1424
[10] Tomlinson T and Miller L 2016 *Advances in experimental medicine and biology* **957** 367–388
[11] Tabot G A, Dammann J F, Berg J A, Tenore F V, Boback J L, Vogelstein R J and Bensmaia S J 2013 *Proceedings of the National Academy of Sciences* **110** 18279–18284 ISSN 0027-8424 (*Preprint* https://www.pnas.org/content/110/45/18279.full.pdf) URL https://www.pnas.org/content/110/45/18279
[12] Tyler D 2015 *Current opinion in neurology* **28**(6) 574–581
[13] Dadarlat M, O'Doherty J and Sabes P 2015 *Nature neuroscience* **18**(1) 138–144
[14] Flesher S N, Collinger J L, Foldes S T, Weiss J M, Downey J E, Tyler-Kabara E C, Bensmaia S J, Schwartz A B, Boninger M L and Gaunt R A 2016 *Science Translational Medicine* **8** 361ra141–361ra141 (*Preprint* https://www.science.org/doi/pdf/10.1126/scitranslmed.aaf8083) URL https://www.science.org/doi/abs/10.1126/scitranslmed.aaf8083
[15] Cronin J, Wu J, Collins K, Sarma D, Rao R, Ojemann J and Olson J 2016 *IEEE transactions on haptics* **9**(4) 515–522
[16] O'Doherty J, Lebedev M, Ifft P, Zhuang K, Shokur S, Bleuler H and Nicolelis M 2011 *Nature* **479**(7372) 228–231 URL https://doi.org/10.1038/nature10489
[17] Yang Y, Qiao S, Sani O, Sedillo J, Ferrentino B, Pesaran B and Shanechi M 2021 *Nature Biomedical Engineering* **5**(4) 324–345 URL https://doi.org/10.1038/s41551-020-00666-w
[18] Tafazoli S, MacDowell C, Che Z, Letai K, Steinhardt C and Buschman T 2020 *Journal of Neural Engineering* **17** 056007 URL https://doi.org/10.1088/1741-2552/abb860
[19] Benabid A 2003 *Current opinion in neurobiology* **13**(6) 696–706
[20] Holtzheimer P and Mayberg H 2011 *Annual review of neuroscience* **34** 289–307
[21] Kisely S, Hall K, Siskind D, Frater J, Olson S and Crompton D 2014 *Psychological medicine* **44**(16) 3533–3542
[22] Fraint A and Pal G 2015 *Frontiers in Neurology* **6** 170 ISSN 1664-2295 URL https://www.frontiersin.org/article/10.3389/fneur.2015.00170
[23] Khanna P, Totten D, Novik L, Roberts J, Morecraft R and Ganguly K 2021 *Cell* **184**(4) 912–930 URL https://pubmed.ncbi.nlm.nih.gov/33571430/
[24] Bosking W, Beauchamp M and Yoshor D 2017 *Annual review of vision science* **3** 141–166
[25] Berger T, Song D, Chan R, Marmarelis V, LaCoss J, Wills J, Hampson R, Deadwyler S and Granacki J 2012 *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society* **20**(2) 198–211
[26] Kahana M J e a 2021 *medRxiv* (*Preprint* https://www.medrxiv.org/content/early/2021/05/22/2021.05.18.212569 URL https://www.medrxiv.org/content/early/2021/05/22/2021.05.18.21256980
[27] Bolus M, Willats A, Rozell C and Stanley G 2021 *Journal of neural engineering* **18**(3)
[28] Dura-Bernal S, Li K, Neymotin S, Francis J, Principe J and Lytton W 2016 *Frontiers in Neuroscience* **10** 28 ISSN 1662-453X URL

https://www.frontiersin.org/article/10.3389/fnins.2016.00028
[29] Sussillo D, Churchland M, Kaufman M and Shenoy K 2015 *Nature Neuroscience* **18**(7) 1025–1033