PATHFINDER: Designing Stimulus for Neuromodulation Through Data-Driven Inverse Estimation of Non-Linear Functions

Chaitanya Goswami, Student Member, IEEE, and Pulkit Grover, Senior Member, IEEE

Abstract—Objective: Using data-driven methods to design stimuli (e.g., electrical currents) which evoke desired neural responses in different neuron-types for applications in treating neural disorders. Methods: The problem of stimulus design is formulated as estimating the inverse of a many-to-one non-linear “forward” mapping, which takes as input the parameters of waveform and outputs the corresponding neural response, directly from the data. A novel optimization framework “PATHFINDER” is proposed in order to estimate the previously mentioned inverse mapping. A comparison with existing data-driven methods, namely conditional density estimation methods and numerical inversion of an estimated forward mapping is performed with different dataset sizes in toy examples and in detailed computational models of biological neurons. Results: Using data from toy examples, as well as computational models of biological neurons, we show that PATHFINDER can outperform existing methods when the number of samples is low (i.e., a few hundred). Significance: Traditionally, the design of such stimuli has been model-driven and/or uses simplistic intuition, often aided by trial-and-error. Due to the inherent challenges in accurately modeling neural responses, as well as the sophistication of stimuli’s effect on neural membrane potentials, data-driven approaches offer an attractive alternative. Our results suggest that PATHFINDER can be applied for optimizing stimulation parameters in experiments and treatments of neural disorders due to it requiring low number of data points.

Index Terms—Neuromodulation, neuroengineering, deep learning, machine learning, neural networks, inverse estimation.

NEUROMODULATION refers to altering neural activity through targeted delivery of a stimulus (e.g., electrical, chemical, ultrasound), and it is one of the fastest-growing areas of medicine, impacting millions of patients [1]. Many neurological disorders result from atypical neural activity in the brain and can be treated by providing appropriate stimuli that can “correct” this atypical neural activity. In experiments, controlling the neural activity through stimuli has shown promise in treating Parkinsonian symptoms [2], facilitating stroke rehabilitation [3], regulating depression [4], etc. Indeed, the ability to systematically design stimuli that produce desired neural activity is key to treating several neurological disorders.

Typically, a stimulus is characterized by a set of parameters. For example, in [2], the authors inject electrical currents (stimuli) into the brain to selectively stimulate a particular type of neuron (controlling neural activity) to treat Parkinsonian symptoms in mice. Their stimulus consisted of three parameters: amplitude, frequency, and duration of the electrical currents. Commonly, the relation between the parameters of stimuli and the neural activity/responses is mathematically modelled through a “forward mapping” that takes as input the parameters and outputs the neural response [5]. The problem of designing stimuli that produce target responses can be viewed as one of inverting the forward mapping. This would allow one to obtain the desired set of stimulus parameters by plugging in the target response as an input to the inverse.

Broadly, there are three major challenges in estimating the inverse of the forward mapping. First, as the forward mapping depends upon the parameters being explored, for novel parameters, the forward mapping is generally unknown and needs to be estimated from the data [2]. Second, in most cases of interest, multiple parameter values lead to the same neural response. For example, in electrical stimulation of the brain, many stimuli may produce the same neural firing rate [6]. That is, the forward mapping is often many-to-one, and hence, non-invertible. Therefore, instead of estimating an inverse, we need to estimate a “pseudoinverse” of the forward mapping (see Section II for details). Third, the amount of data available in such settings is limited, and in general, the data-collection process is quite expensive (e.g. in [2], authors could only collect dataset sizes of ~300). So, it is desirable to estimate the pseudoinverse in a data-efficient manner.

Manuscript received 28 September 2022; revised 26 March 2023 and 11 May 2023; accepted 24 May 2023. Date of publication 1 June 2023; date of current version 22 November 2023. This work was supported in part by the Naval Information Warfare Center (NIWC) Atlantic and the Defense Advanced Research Projects Agency (DARPA) under Grant N65236-19-C-8017. The work of Chaitanya Goswami is also supported by a fellowship awarded by the Centre of Machine Learning and Health (CMLH) at Carnegie Mellon University. (Corresponding author: Chaitanya Goswami.)

Chaitanya Goswami is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: cgoswami@andrew.cmu.edu).

Pulkit Grover is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, USA.

This article has supplementary downloadable material available at https://doi.org/10.1109/TBME.2023.3282155, provided by the authors. Digital Object Identifier 10.1109/TBME.2023.3282155
In this work, we are interested in purely data-driven approaches to designing stimuli through pseudoinverse estimation, which has received significant recent attention [7], [8], [9], [10]. A main advantage of data-driven approaches over traditional stimuli design approaches is that they allow end users (e.g., clinicians or neuroscientists) to explore a larger number of parameters as compared to traditional approaches, creating the possibility of discovering novel stimuli, e.g. for improved clinical relevance (see Section VII for details). Broadly, two methodologies have been explored in literature for estimating the pseudoinverse for the purpose of stimuli design. One is to use conditional density estimation (CDE) methods to learn the conditional density of waveform parameters conditioned on firing rates, and then use the conditional mode as the pseudoinverse [7], [11]. The other is to estimate the forward mapping (e.g. using a neural network) and then numerically invert it [8], [9]. We provide a detailed discussion regarding both approaches in Section III. Briefly, CDE-based methods are known to require more data than their regression counterparts [12], and NI approaches suffer because numerical inversion blows up even small errors in forward models (see Section VII). These observations are reflected in our results in Section VI.

To address these limitations, we propose PATHFINDER: a novel pseudoinverse estimation framework that adapts regression techniques\(^1\) to directly estimate a pseudoinverse, thereby circumventing the need of inverting an estimated forward model, while still requiring less data than CDE methods (regression techniques typically require less data than their CDE counterparts). Section IV provides a detailed description of PATHFINDER. The key insight utilized by PATHFINDER is that a non-invertible function can still be inverted over a restricted domain. If such a restricted domain were known a priori, the inverse mapping can be estimated using traditional regression methods. PATHFINDER jointly learns a restricted domain and the inverse mapping over it. To do so, PATHFINDER uses a weighted \(l_2\) loss, where the weights are also learned from data. On convergence, the weights approximate the indicator function over the restricted domain, effectively learning it (theoretically justified in Section VI).

In Section VI-B and Section VI-C, we compare the performance of PATHFINDER to two CDE methods: Masked Autoregressive Flows (MAF) [14] and Mixture Density Networks (MDN) [15], as well as a naïve inversion of deep network (NI) in three toy examples, as well as a dataset with two simulated neuron models (models from Allen Cell Type Database [16]). The details of the simulation setup are provided in Section V. We observe that, as predicted, PATHFINDER outperforms all the other methods for small training datasets. We provide a detailed discussion of the results obtained in Section VI in Section VII, and conclude in Section VIII.

\section*{II. \textbf{Problem Statement and Notation}}

Each stimulus is characterized by \(n\)-different parameters, with the \(j\)-th parameter denoted as \(\theta_j\), where \(\theta_j\) is a real number.

We collect all the parameters \(\{\theta_j\}_{j=1}^n\) into a parameter vector, denoted as \(\bar{\theta} = [\theta_1, \ldots, \theta_n]^T\). Let \(\Theta\) be the set/collection of all parameter vectors \(\bar{\theta}\) corresponding to all allowed stimuli. E.g., in [2], the stimuli of electrical currents is characterized by 3 parameters which are the amplitude, frequency and duration of the currents. In our notation, the amplitude, frequency and duration of the electrical currents would be denoted by \(\theta_1, \theta_2, \theta_3\), and the parameter vector would be \(\bar{\theta} = [\theta_1, \theta_2, \theta_3]^T\). Let the number of neural responses of interest be \(m\), and the \(k\)-th neural response be denoted as \(r_k\), where \(r_k\) is a real number. Define the response vector as \(\bar{r} = [r_1, \ldots, r_m]^T\). Then \(\mathcal{R}_\Theta\) be the collection of all distinct neural responses produced by all the stimuli present in \(\Theta\). For example, the relevant neural responses in [2] are the firing rates of the different neuron types, so \(\bar{r} = [r_1, r_2]^T\), where \(r_1\) and \(r_2\) are the firing rates of the two neuron types.

\textbf{Problem Statement:} Given a dataset \(\mathcal{D} = \{\bar{\theta}_i, \bar{r}_i\}_{i=1}^N\), formed by \(N\) parameter-vector-response vector pairs, where \(\{\bar{\theta}_i\}_{i=1}^N\) are independent and identically distributed (i.i.d.) samples\(^2\) from a distribution \(p(\bar{\theta})\) on \(\Theta\), and \(\bar{r}_i\) is the neural response generated by the stimulus characterized by \(\bar{\theta}_i\), the goal is to design/find a parameter \(\bar{\theta}_{des}\) from the set of allowed stimuli \(\Theta\) such that the stimulus corresponding to \(\bar{\theta}_{des}\) can elicit (close to) a desired (user-specified) neural response \(\bar{r}_{des}\) from the set of achievable responses \(\mathcal{R}_\Theta\).

We only allow access to the (fixed) dataset \(\mathcal{D}\). To restrict the scope of our work, we do not allow the acquisition of more data, i.e., actively sampling based on inferences from existing data. All the methods discussed in this work can be extended to their active-sampling versions, and the performance of these methods in that setting will be explored in future work. We allow restricted access to neurons for fine-tuning some hyper-parameters of each method (see Section VI), but to compare fairly, this data is not used to train any of the methods. An important design parameter in our problem is choosing the set of allowed stimuli i.e. \(\Theta\). This is decided a priori by the user, typically based on domain knowledge, e.g., [2] choose amplitude, frequency, and duration of the electrical waveforms as parameters of stimuli. For this work, we will assume that an appropriate \(\Theta\) has already been chosen.

We denote the forward mapping from the stimulus parameter space \(\Theta\) to the neural response space \(\mathcal{R}_\Theta\) as \(g : \Theta \rightarrow \mathcal{R}_\Theta\), a many-to-one function. A pseudoinverse of \(g\) is a mapping \(g^{-1} : \mathcal{R}_\Theta \rightarrow \Theta_{inv}\), where \(\Theta_{inv}\) (a subset of \(\Theta\)) is a restricted domain of \(g\), such that \(g(g^{-1}(\bar{r})) = \bar{r}\) for all \(\bar{r}\) present in neural response space. E.g., \(\cos : \mathbb{R} \rightarrow [-1, 1]\) is many-to-one and not invertible, but restricting the domain of \(\cos\) to \([0, \pi]\) provides the familiar pseudoinverse: \(\cos^{-1} : [-1, 1] \rightarrow [0, \pi]\). Note, for a pseudoinverse, equality in the other direction, i.e., \(g^{-1}(g(\bar{r})) = \bar{r}\) for all \(\bar{r}\) present in \(\Theta\), may not hold. A function can have multiple pseudoinverses (e.g., \(\cos(\cdot)\), with \(\Theta_{inv} = [0 + 2n\pi, \pi + 2n\pi]\), has infinitely many pseudoinverses, one for each integer \(n\)). For the goal of this work, estimating any one pseudoinverse suffices. We denote the estimated pseudoinverse of \(g\) as \(g_{est}\). Supp. Table 1 summarizes the symbols and abbreviations used in this work.

\(^1\)Traditional regression techniques do not work for pseudoinverse estimation, see [13] for more details.

\(^2\)Note that we require \(\bar{\theta}_{des}\), i.e. the parameter vector, collected for the dataset to be independent, which is different from the stimulus parameters, which are components of \(\bar{\theta}\), to be independent of each other in a parameter vector.
III. BACKGROUND AND EXISTING APPROACHES

We now discuss natural approaches based on existing literature for this problem.

Naive Inversion: Conceptually, the simplest approach to estimate a pseudoinverse is what we call naive inversion (NI): numerically invert an estimate \( \hat{g}(.\cdot) \) of the forward mapping \( g \). The general framework for NI can be summarized as follows:

\[
\hat{g} = \arg \min_{f \in \mathcal{F}_F} \sum_{(\hat{\theta}, \hat{r}) \in \mathcal{D}} l \left( f(\hat{\theta}_i), \hat{r}_i \right), \quad (1)
\]

\[
\hat{g}^{-1} = \arg \min_{f \in \mathcal{F}_I} \sum_{(\hat{\theta}, \hat{r}) \in \mathcal{D}} l \left( \hat{g}(f(\hat{\theta}_i)), \hat{r}_i \right), \quad (2)
\]

where \( \mathcal{F}_F \) and \( \mathcal{F}_I \) denote the family of functions being considered for estimating forward and inverse mapping (respectively), \( \hat{g}^{-1} \) is the pseudoinverse estimate and \( l(\cdot, \cdot) \) is an appropriate loss function, (e.g. the \( l_2 \) loss).

We now discuss a few recent works implementing the NI framework. Authors in [9] proposed XDream, a genetic algorithm that fine-tunes the input visual stimuli to the generator of a pre-trained generative adversarial network (GAN) [17] to maximize the neural response, but requires access to the neurons during training. Authors in [8] trained a convolutional neural network (CNN) using data collected from monkeys, and use a softmax loss to find a visual stimulus that selectively evokes response in ‘V1’ (the primary visual cortex), but this technique is limited to only finding selective firing rates. Authors in [10] developed a closed-loop experimental paradigm for optimizing visual stimulation in rats. They trained a CNN over multiple data-collection sessions. After each session, they used the trained CNN to find the images that maximally excited the target neurons, through gradient ascent. They then retrained the CNN with those new images to find the next set of images that maximally excited the neurons.

A major drawback of the NI approaches is that (2) is often hard to optimize (see Section VII), as \( \hat{g} \) is typically a complicated function (e.g. deep neural network), which makes the overall loss prone to getting stuck in local minima. This could explain why NI performs poorly in our numerical study in Section VI. All of the works discussed above were focused on finding one waveform/stimulus (e.g., selective stimulation in [8], maximal excitation in [10]) rather than a pseudoinverse with a large range, so they were able to circumvent (to a degree) the hardness of solving (2). [8] tackled the problem by using a softmax loss, but that is only applicable if we want to evoke selective activity. [10] and [9] took an active-sampling approach, where they retrained the forward mapping across multiple data-collection sessions based on the estimated stimulus. This allowed the forward mapping to be quite accurate, but only around the desired response. NI is also quite sensitive to the accuracy of the forward models, and accurate models and accurate models in many modalities are still lacking, e.g. electrical stimulation.

Conditional density estimation: The modes of the conditional distribution \( p(\hat{\theta}|\hat{r}) \) are, by definition, the most likely stimuli for producing the neural response \( \hat{r} \). Thus, a natural candidate for the pseudoinverse of \( g \) is:

\[
\hat{g}^{-1}(\hat{r}) := \arg\max_{\hat{\theta}} p(\hat{\theta}|\hat{r}) = \arg\max_{\hat{\theta}} p(\hat{\theta}, \hat{r}), \quad (3)
\]

where \( p(\hat{\theta}, \hat{r}) \) denotes the joint density. Two commonly-used CDE methods for estimating pseudoinverses are Masked Autoregressive Flow (MAF) [14] and Mixture Density Network (MDN) [15]. We now discuss works that employ these approaches. [7] used Sequential Neural Posterior Estimation (SNPE) [18] to estimate parameters of computational models of neurons from the visual cortex using MDNs. In [11], the authors proposed Sequential Neural Likelihood (SNL), which outperforms SNPE, and uses MAF. MA Inf is an instance of normalizing flows density estimator [19]. A particular drawback of normalizing flows (and hence MAF) is that we need to solve (3) using an optimization technique (e.g., gradient descent [20]), which may get stuck in a local minimum. Both, SNL and SNPE are active sampling methods that use MAF and MDN to actively sample the dataset, respectively. To test them in our non-adaptive setting, we use MAF and MDN to estimate \( p(\hat{\theta}|\hat{r}) \) directly from a randomly sampled dataset, without using active sampling. [21] and [22], though not focused on neural stimulation, also used MDNs to estimate pseudoinverses. MDNs with Gaussian component distributions are particularly attractive for learning pseudoinverses as the means predicted by the MDN provide an approximate estimate of the modes. Broadly, CDE-based approaches require a large amount of data and are hard to implement in high dimensions [11]. Regression methods, although still suffering from the curse of dimensionality, tend to perform better than their CDE counterparts at a lower number of data samples [12], which we also observed in our simulation results (see Section VI). Our regression-based approach, PATHFINDER, is discussed next. In Section VI, we compare MDN, MAF, NI, and PATHFINDER, in toy examples and a neuromodulation context.

IV. PATHFINDER

PATHFINDER estimates a pseudoinverse by harnessing the insight of Section II, namely, that many-to-one functions can be inverted over appropriately restricted domains. If such a restricted domain \( \Theta_{inv} \) is known a priori, then a restricted dataset can be created by excluding all data points \((\hat{\theta}_i, \hat{r}_i)\) where \( \hat{\theta}_i \) does not lie in the restricted domain \( \Theta_{inv} \) from the training dataset \( \mathcal{D} \). Since \( g \) is invertible over \( \Theta_{inv} \), any traditional regression technique applied to this restricted dataset will yield a pseudoinverse corresponding to \( \Theta_{inv} \). Formally, a pseudoinverse on \( \Theta_{inv} \) can be estimated as:

\[
\hat{g}^{-1} = \arg \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{I}[\hat{\theta}_i \in \Theta_{inv}] \| f(\hat{r}_i) - \hat{\theta}_i \|^2, \quad (4)
\]

where \( \mathcal{I}(\cdot) \) is the indicator function and \( \mathcal{F} \) is the family of functions being considered for regression. The challenge, as outlined above, is that we only have access to a dataset (and not the forward mapping), so \( \Theta_{inv} \) is not known a priori. To address this, PATHFINDER jointly estimates both a restricted
domain and the corresponding pseudoinverse as follows:

\[
g^{-1}, \{\hat{w}(\hat{\theta}_i)\}_{i=1}^N = \arg \min_{\{\hat{w}_i\}_{i=1}^N} \frac{1}{N} \sum_{i=1}^N w(\theta_i) \|f(\hat{r}_i) - \hat{\theta}_i\|_2^2 + \beta \sum_{i=1}^N w^2(\hat{\theta}_i), \text{ s.t. } \frac{1}{N} \sum_{i=1}^N w(\hat{\theta}_i) = 1,
\]

(5)

where \(\beta \in \mathbb{R}^+\) is a hyper-parameter. This optimization formulation follows the philosophy of (4), approximating \(\mathbb{I}[\hat{\theta}_i \in \Theta_{\text{inv}}]\) in (4) by \(\hat{w}(\hat{\theta}_i)\) which are learnt jointly with \(g^{-1}\).

How does the PATHFINDER optimization (5) incentivize learning of a restricted domain? If only parameters belonging to a restricted domain have non-zero weights \(\hat{w}(\hat{\theta}_i)\), the loss term would be low because the corresponding inverse mapping can be estimated accurately. Hence, the loss term encourages PATHFINDER to learn weights that are non-zero only for some restricted domain over which \(g\) is invertible. It is desirable that an estimate of the pseudoinverse \(g^{-1}\) (as discussed in Section II) has its domain as the entire \(\mathcal{R}_\Theta\), or at least as large a subset as possible, so it can provide stimuli for as many neural responses as possible (within the constraints discussed in Section II). This implies that in order for PATHFINDER to estimate a pseudoinverse, the image of its learned restricted domain should be as large as possible. This condition is not ensured by the loss term in (5), since it is small for any restricted domain (e.g., consider the two restricted domains \([0,1]\) and \([0, \pi]\) of \(\cos(\cdot)\); \([0, \pi]\) is more desirable here, as it corresponds to a larger range of response, i.e., \([-1,1]\). However, the loss term in (5) is small for both domains, and is unable to discriminate between them).

To encourage PATHFINDER to learn a restricted domain corresponding to a large response range/space, we use the following observation: If \(\Theta_{\text{max}}\) is the restricted domain having the largest size (measured by its total probability under \(p(\theta)\)), then its corresponding image/response space also has the largest size (theoretically justified in Supp. Sec. II). Hence, by learning the largest restricted domain we ensure we have the largest response space. This observation is incorporated in the regularizer and the constraint in (5) to encourage PATHFINDER to learn as large a restricted domain as possible. To see this, let us examine the following optimization that distills the effect of the regularizer to distinguish among restricted domains over which \(g\) is invertible (the first term in (5) is low for such domains):

\[
\{w_i^*\}_{i=1}^N = \arg \min_{\{w_i\}_{i=1}^N} \sum_{i=1}^N w_i^2, \text{ s.t. } \frac{1}{N} \sum_{i=1}^N w_i = 1,
\]

\[
\sum_{i=1}^N \mathbb{I}[w_i \neq 0] = K, \quad w_i \geq 0 \forall i.
\]

(6)

This optimization explores the behavior of the regularizer when only \(K\) out of \(N\) total weights are non-zero, and has the solution: \(w_i^* = N/K\) for any \(K\) out of \(N\) weights; the rest are 0. The regularizer term in (5) scales approximately as \(\sim 1/K^2\) for \(K\) non-zero weights, which incentivizes making a larger number of \(\hat{w}(\hat{\theta}_i)\) to be non-zero, encouraging PATHFINDER to consider as large a restricted domain as possible. Thus, there is a careful interplay between the loss, the regularizer and the constraints in (5). The loss encourages learning non-zero weights \(\hat{w}(\hat{\theta}_i)\) only over a restricted domain; the regularizer and the constraints try to make the restricted domain as large as possible. By carefully choosing the value of \(\beta\), a desirable pseudoinverse can be learnt.

A. PATHFINDER Ensemble

The PATHFINDER Ensemble algorithm is an extension of PATHFINDER that estimates an ensemble of pseudoinverses, instead of just one. Estimating an ensemble of pseudoinverses improves the performance, as we observe empirically in Section VI. The PATHFINDER Ensemble algorithm creates an ensemble of pseudoinverses in a greedy fashion as follows: (i) First, PATHFINDER is used to estimate the pseudoinverse and the corresponding weight mapping on the training dataset; (ii) The estimated weight mapping is used to identify and remove the datapoints in the training dataset that lie in the restricted domain (estimated by PATHFINDER in step (i)) to construct a new training dataset; and (iii) This new training dataset is then again used to perform step (i) and step (ii), and this process continues until the (remaining) training dataset becomes empty, resulting in an ensemble of pseudoinverses that are distinct due to step (ii). To predict the parameter vector \(\hat{\theta}\) for some response vector \(\hat{r}\), we choose the output of the pseudoinverse whose restricted domain contains the training datapoint with the response vector closest to the desired response vector. Intuitively, this pseudoinverse would have the most confidence in predicting the right parameter. Note that none of the other methods (discussed in Sec. III) can be used to create an ensemble of pseudoinverses in this fashion as only PATHFINDER explicitly estimates the restricted domain (in terms of the weight mapping) which is required for step (ii).

V. METHODS

In this section, we provide the implementation details of the simulations studies performed in Section VI. These studies were designed to characterize the ability of PATHFINDER, MDN, MAF and NI in estimating pseudoinverses – in toy examples and in an example of electrical stimulation of two neuron models – as a function of training dataset size. Additional details are provided in Supp. Section V.

A. Simulation Setup and Dataset Generation

We provide descriptions for the two scenarios, i.e. toy examples and bio-physical neuron models, in which we compare the performance of PATHFINDER, MAF, MDN and NI for estimating pseudoinverses.

Toy Example: We considered 3 different toy mappings for the purpose of estimating pseudoinverses. The following models were used to generate the data from each toy mapping:

\[
r = \cos(2\pi \theta) + \epsilon_1, \quad \theta \in [0, 3]; \quad r = e^{-\frac{\theta^2}{2}} + \epsilon_2, \quad \theta \in [-3, 3];
\]

\[
r = (\theta^2 - 4)^2 + \epsilon_3, \quad \theta \in [-3, 3],
\]

(7)
where $\epsilon_1, \epsilon_2,$ and $\epsilon_3$ are distributed according to the Gaussian distribution with mean 0 and variance 0.1$^2$, 0.2$^2$ and 2.5$^2$, respectively. They represent noise. To create a dataset of size $N$ for a toy mapping, i.e. $D_{\text{toy}} = \{ (\theta_i, r_i) \}_{i=1}^N$, $N$ samples of $\theta$ were uniformly randomly sampled from that toy forward mapping’s respective domain provided in (7). The corresponding $\{r_i\}_{i=1}^N$ for each toy mapping were then generated according to the respective models provided in (7).

**Bio-physical Neuron Models:** The cortex consists of excitatory and inhibitory neurons [5] and it is hypothesized that a careful interplay between the activation of excitatory and inhibitory neurons in the cortex drives its neural activity [23]. Imbalance in the activation of excitatory and inhibitory neurons is linked to neurological diseases such as seizures [24]; hence the ability to modulate the activity of excitatory and inhibitory neurons is of clinical relevance. In recent years, there has been a growing interest in using electrical currents to stimulate excitatory and inhibitory at desired firing rates [25], [26]. Controlling the firing rates of excitatory and inhibitory neurons (neural response) with electrical stimulation (stimulus) provides a good simulation setup to test the performance of PATHFINDER, MAF, MDN and NI in a more realistic neuromodulation context.

We adapt the simulations setup used in [25], but we replace its single neuron models with morphologically-realistic and bio-physically detailed multi-compartment neuron models taken from the Allen Cell Type Database. One neuron model is of a cortical pyramidal (Pyr; excitatory) neuron and other is of a cortical Parvalbumin-expressing (PV; inhibitory) neuron. Both neurons were simulated using the NEURON software [27] and the allensdk package [16] in python [28]. We construct a parametric waveform family using 5 parameters: $Q$, $A_n/A_p$, $T$, $T_{\text{zero}}/T$, and $T_{\text{neg}}/(T-T_{\text{zero}})$, where $A_p$, $A_n$, $T$, $T_{\text{zero}}$, and $T_{\text{neg}}$ are as depicted in Fig. 3(a). $Q$ refers to the total absolute charge of the waveform. The range of each parameter is as follows: $[0.15\text{nC}, 0.35\text{nC}]$ for $Q$, $[0.5, 2]$ for $A_p/A_n$, $[0.1\text{ms}, 500\text{ms}]$ for $T$, $[0, 0.5]$ for $T_{\text{zero}}/T$ and $[0, 0.6]$ for $T_{\text{neg}}/(T-T_{\text{zero}})$. Each parameter is uniformly sampled from its respective range. The duration of the waveform is 1000 ms. The corresponding waveform $u_{\theta_i}(t)$ (generated from $\theta_i$) is injected intracellularly into the soma of each neuron model. We also add synaptic noise to our models by adding additive white Gaussian noise with a mean of zero and variance of 2.25 ($\mu A$)$^2$ to the input waveform [29]. We run a peak detection algorithm, present in the scipy package [30] in python, on the time-trace of the resulting neuron’s membrane potentials to calculate the number of neural spikes. We use a simplistic definition of the firing rate for this simulation, i.e.

$$r = \frac{\text{Total Number of Spikes}}{1000 \text{ ms}},$$

where $r$ denotes the firing rate of the neuron model. We collect the firing rates for both neuron models to construct our neural response $\vec{r}_i = [r_{i1}, r_{i2}]^T$ ($r_{i1}$ for Pyr neuron and $r_{i2}$ for PV neuron). The range of firing rates in our dataset for the Pyr neuron is [0 Hz, 40 Hz] and [0 Hz, 100 Hz] for the PV neuron.

Typically, such a dataset would be collected using in-vivo patch-clamp electrophysiology [2], [31] (see Supp Section IV) and collecting data using such methodology is extremely expensive. Publications employing these techniques usually report dataset sizes of around 200-300 samples (waveforms). A recent work [2], which achieved selective stimulation between two pairs of neuron types using Gaussian Process Regression [32], uses a dataset of about 250 samples. This informs the range of dataset sizes we investigate in Section VI-C.

**B. Evaluation of Waveforms and Figure of Merit**

1) **Splitting the Dataset:** Since the input to our model is the neural response $\vec{r}_i$ and not the stimulus parameter $\vec{\theta}_i$, we split our data into training, test, and validation sets in the following manner: Split all possible neural responses present in $\mathcal{D}$ into training, validation and test firing rates. Let $\mathcal{R}_V$, $\mathcal{R}_{Te}$, and $\mathcal{R}_{Tc}$ be the sets containing the validation, training and test neural responses. Remove all the $\{\vec{\theta}_i, \vec{r}_i\}$ from the original dataset $\mathcal{D}$ where $\vec{r}_i$ is present in either the test or the validation set, to construct the training dataset $\mathcal{D}_{Tr}$. Note that for any $\vec{r}$ present in the test or the validation set, there may be multiple stimuli parameter vectors ($\vec{\theta}$) in the original dataset that produce $\vec{r}$ and we remove all of them. The validation dataset $\mathcal{D}_V$ can be similarly constructed by removing $\{\vec{\theta}_i, \vec{r}_i\}$ from the original dataset $\mathcal{D}$, where $\vec{r}_i$ is present in the training or the test set. For the test set, we only store the neural responses (i.e., $\mathcal{R}_{Te}$) as we want to generate stimuli that produce those neural responses. The exact test set chosen for the neural model study is described in Fig. 2 and consists of 10 Pyr-selective firing rate pairs (i.e., Pyr neuron fires at higher firing rate than PV neuron model), and 10 PV-selective firing rate pairs. **Selective stimulation between neuron types** has clinical relevance [2], [25], [33], and presents a challenging goal as intuitive strategies (e.g. increasing signal amplitude) often increase firing rates of both neurons. The test set for the toy examples consisted of 20 datapoints uniformly randomly sampled from their respective range of responses.

2) **Estimating the Waveform Parameters for Each Approach:** After training, each approach outputs a parameter $\vec{\theta}$ corresponding to every $\vec{r}$. The procedure for obtaining $\vec{\theta}$ for every approach is detailed as follows: PATHFINDER and **Naive Inversion** obtain $\vec{\theta}$ as the output of the estimated pseudoinverse, when provided with the input $\vec{r}$. For PATHFINDER-Ensemble, we use the prediction scheme described in Section IV-A. For MAF, we solve the optimization specified in (3) for every $\vec{r}$, using gradient ascent. We obtain $\vec{\theta}$ for MDN using an approximate solution of (3): the mean of the mixture with the highest probability corresponding to $\vec{r}$.

3) **Evaluating the validation/test Loss and Figure of Merit:** We will explain our figure of merit by using an example of its calculation over the validation set. For every neural response $\vec{r}$ present in $\mathcal{R}_V$, we obtain the corresponding $\vec{\theta}$ using the procedure detailed in the previous section. We feed the $\vec{\theta}$ to the neuron/model to obtain its actual firing rate $\vec{r}_{\text{act}}$, i.e. $g(\vec{g}(\vec{r})) = \vec{r}_{\text{act}}$. Since, we have $m$ different neural responses, we calculate the normalized mean absolute error (NMAE) for each individual neural response. We denote the maximum and
minimum values that can be achieved for the \( j \)-th neural response as \( r_j^{\max} \) and \( r_j^{\min} \), respectively. Then, we define the NMAE for the \( j \)-th neural response as follows:

\[
\text{NMAE}_j = \frac{100}{N_V \left( r_j^{\max} - r_j^{\min} \right)} \sum_{\vec{r} \in \mathbb{R}^V} |r_j - r_j^{\act}|, \tag{9}
\]

where \(|\cdot|\) is the absolute function, \( N_V \) is the number of response vectors present in the validation set, and \( r_j \) and \( r_j^{\act} \) are the \( j \)-th components of \( \vec{r} \) and \( \vec{r}^{\act} \), respectively. Note that for calculating NMAE, we do not explicitly need to know the restricted domain over which pseudoinverse has been estimated. The NMAE quantifies how close the neural response produced by the predicted stimulus is to the desired neural response on a scale of 0 to 100. The test NMAE can be calculated in a similar manner, where we replace the validation set by the test set. Notice that, for calculating the NMAE, we require access to the neuron (hence the need for accessing the neuron during hyper-parameter tuning, as discussed in Section II).

C. Implementation Details of Data-Driven Approaches

In this section, we briefly describe the details of implementing PATHFINDER, MAF, MDN, NI and a baseline approach (see Supp. Section V for a more detailed description). To choose the hyper-parameters for all the data-driven approaches, we use the 10-fold cross-validation NMAE. Specifically, we pick the hyper-parameter with the lowest cross-validation NMAE. To calculate the 10-fold cross-validation NMAE, we create a 90%-10% split of the dataset, where 90% of the datapoints are in the training data and 10% of the datapoints are in the validation data, and the NMAE is calculated for the validation data. This process is repeated 10 times, and the average validation NMAE is used as the test NMAE corresponding to the number of flows that produce the lowest validation NMAE. The exact values of the hyper-parameters are provided in Supp. Section V.

**MAF:** We used the implementation suggested in the original article [14]. MAF was trained to learn the joint density of data \( p(\vec{\theta}, \vec{r}) \), instead of \( p(\vec{\theta} | \vec{r}) \) as joint density provides more information, and both joint and conditional densities are equivalent for estimating the pseudoinverse (see (3)). For all the three toy-examples, we used a MAF with three flows; and for the bio-physical neuron model setting, we considered a range for the number of flows from 2 flows to 8 flows, and we present the test NMAE corresponding to the number of flows that produce the lowest validation NMAE. The exact values are provided in Supp. Section V. After each flow, a batch normalization flow (see [14]) is applied. The order of the inputs is reversed after each flow. The initial order of the inputs was assigned randomly. The corresponding Masked Autoencoder for density estimation (MADE) [35] for each MAF flow, consists of 2 hidden layers, each with 10 hidden units for the toy example and 100 hidden units for the bio-physical neuron example. For base-density, a standard Gaussian was considered.

**MDN:** We chose to implement a Gaussian MDN. For each toy example, the number of mixtures was 15; for the bio-physio-logical neuron model a ranging number of mixtures was considered, starting from 10 to 100 in intervals of 10. For the toy examples, a Multilayer Perceptron (MLP) with 5 hidden layers, each with 10 hidden units, is used. For the neuron model case, MLPs with 5 to 8 hidden layers, each layer with 100 hidden units, were considered. Again the results are presented for the hyper-parameter settings that produced the lowest validation NMAE. The exact values of the hyper-parameters are provided in Supp. Section V.

**Naïve Inversion:** The architecture can be visualized as an autoencoder, where the decoder serves as the forward model and the encoder as the inverse. For the toy examples, both the forward and inverse models are MLPs with 5 hidden layers, each with 10 hidden units. For the neuron models, MLP having 5 to 8 hidden layers, each with 100 hidden units, are considered for both the forward and inverse mapping. The exact values of hyper-parameters are provided in Supp. Section V.

**Baseline:** In addition to the data-driven methods, a brute-force approach, akin to the 1-Nearest Neighbour Method [20], is used...
as a baseline method for the simulation study performed in Section VI. In the baseline method, we use the training data as a look-up table, where we output the parameter vector present in the training dataset whose response vector is closest to the desired response vector under the $l_1$ distance metric.

Rectified Linear Unit (ReLU) activation was used for hidden layers of all approaches. MAF and MDN were trained by minimizing the negative log-likelihood (NLL). For naive inversion, the $l_2(\|\cdot\|_2)$ loss. First, the forward mapping is trained using (1) and then the weights of the forward model were frozen. Next, the inverse model was trained by minimizing the loss in (2). In PATHFINDER, both weight and regressor were trained simultaneously by minimizing the loss in (5). While training PATHFINDER, a simplex projection [38] was performed on the batch output of the weight network to ensure the constraints specified in (5) are met (see Supp Section V for more detail). Adam optimizer [39] with a learning rate of 0.001 is used for minimizing the objectives for all techniques. Batch normalization [40] was applied wherever applicable. Since all the methods were trained for multiple different sizes of the training datasets, batch size for Adam was chosen according to the training dataset size. A detailed list is provided in Supp. Section V. We trained each model until the validation loss converged (note loss instead of MAE), defined as a relative change of less than 0.1% in the validation loss between successive steps. Validation loss here was calculated using the more traditional method. For example, to calculate the validation loss in MDN and MAF, we simply calculate the negative log-likelihood of the validation set. In our simulations, we found that both validation loss or validation NMAE suffice for deciding Adam’s convergence, but validation loss does not require access to neurons and hence is more desirable. A detailed description is provided in Supp. Section V. All of the implementation mentioned in this section was performed in tensorflow [41] and tensorflow probability [42].

VI. RESULTS

In this section, we detail our theoretical as well as simulation results regarding PATHFINDER. The organization of this section is organized as follows. In Section VI-A, we prove a theorem showing that given enough data, the PATHFINDER loss estimates the pseudoinverse with the largest restricted domain (as defined in Section IV) with arbitrary precision. Section VI-B and Section VI-C detail the results of the simulation studies we performed, showing that PATHFINDER outperforms MAF, MDN, and NI in the small data regime, thereby making PATHFINDER a more desirable alternative to MDN, MAF and NI in the neuromodulation context (training datasets are typically small in neuromodulation applications; see Section I).

A. Theoretical Results

This section provides a formal justification of the intuition behind PATHFINDER (Section IV), albeit under idealized assumptions of sufficiently rich $\mathcal{F}$ and noiseless data. The optimization problem of PATHFINDER defined in (5) can be viewed as approximating the following problem:

$$g^{-1}, w^* = \arg \min_{f, w \in \mathcal{W}} \mathbb{E}_{p(\theta)} [w(\bar{\theta})\|f(\bar{\theta}) - \bar{\theta}\|_2^2] + \beta \mathbb{E}_{p(\theta)} [w(\bar{\theta})], \text{ s.t. } \mathbb{E}_{p(\theta)} [w(\bar{\theta})] = 1, \quad (10)$$

where $\mathcal{W} = \{ w : \Theta \rightarrow \mathbb{R}^+ | w \text{ is a measurable function} \}$, $\mathcal{F}$ is the family of the functions being considered for regression, and $\mathbb{E}_{p(\theta)} [\cdot]$ is the expectation with respect to $p(\theta)$ (defined in Section II). In the following theorem, $g_{\max}^{-1}$ is the pseudoinverse corresponding to $\Theta_{\max}$, the largest restricted domain over which $g$ is invertible (see Section IV).

**Theorem 1:** Assume that $g : \Theta \rightarrow \mathcal{R}_{\Theta}$ is a Lipschitz $l_2$-integrable function such that $g_{\max}^{-1}$ as defined above exists, $g_{\max}^{-1} \in \mathcal{F}$ where $\mathcal{F}$ is the family of functions being considered for estimation, and the dataset is noiseless. Then, for any $\epsilon > 0$, $\exists \alpha > 0$ such that for $0 < \beta \leq \alpha$, the following applies:

$$\mathbb{E}_{p(\theta)} \left[ ||g^{-1}(\bar{r}) - g_{\max}^{-1}(\bar{r})||_2^2 \right] \leq c_1 \epsilon + c_2 \sqrt{\epsilon}. \quad (11)$$

where $g^{-1}$ is the solution of (10), and $c_1, c_2 \in \mathbb{R}^+$. 

**Proof:** The proof is provided in Supp. Section III.

Theorem 1 implies that if the global optimum of (10) is attainable, then given enough data, PATHFINDER can estimate the pseudoinverse $g_{\max}^{-1}$ with arbitrary precision by tuning $\beta$. Note that because the optimization problem defined in (10) is non-convex, guaranteeing convergence to this optimum is non-trivial, but in practice, stochastic gradient descent methods [20] performed reasonably well in solving the PATHFINDER loss. In our theoretical result, we only analyze the simpler case of no noise, but empirically we observe PATHFINDER also performs well in the presence of noise (see Section VI-C). The change in the theoretical bounds on $\beta$ with the noise and the finite size of the dataset is left for future work, but empirically $\beta$ can be found by cross-validation error, like any other hyper-parameter (see Section V-B3).

B. Simulation Results: Toy Examples

The main result presented in this section is that, in the small data regime, PATHFINDER outperforms MAF, MDN and NI in estimating the pseudoinverse for the 3 toy forward mappings presented in Section V-A. To characterize the data-efficiency of all the data-driven techniques in estimating pseudoinverses, we calculate the NMAE of each data-driven technique at 4 different training dataset sizes for all the 3 toy examples. The resulting NMAEs are shown in Fig. 1(a)–(c).

We observe that at small dataset size, PATHFINDER GP-Ensemble has a $>7\%$ decrease in NMAE compared to the next data-driven technique, which is not a variant of PATHFINDER. This provides evidence regarding the data-efficiency of PATHFINDER. Furthermore, GP-variants of PATHFINDER outperform their neural network counterparts. This is also expected since GP tends to be more data-efficient than neural networks in regression tasks. We also observe that the ensemble variants of PATHFINDER have less error compared...
to their standard versions, justifying the use of PATHFINDER-Ensemble. Fig. 1(d)–(i) illustrate\(^3\) the pseudoinverse estimated by each technique for the Gaussian toy example \((r = e^{-\theta^2/2})\) at 10 training datapoints. The corresponding visualization of the pseudoinverse estimated by all the data-driven methods for the other two toy examples are provided in Supp. Section V-C. Results remain qualitatively the same in the noiseless case (see Supp. Section VII).

C. Simulation Results: Electrical Stimulation in Neurons

In this section, we present the results obtained by applying PATHFINDER, MDN, MAF and NI to estimate the stimuli parameters (electrical waveform in this particular case) for desired neural responses (firing rate of the two neuron models). The main goal of this simulation study is to show that PATHFINDER performs better at “small” dataset sizes and is comparable to other methodologies at “large” dataset sizes. That is, qualitatively, the toy example results obtained in Section VI-B continue to hold in a more realistic scenario. The setup of the performed simulation is detailed in Section V.

We calculated the NMAE for all the data-driven approaches (PATHFINDER, MDN, MAF, and NI), as well as the baseline approach at different training dataset sizes, namely, 25, 50, 100, 250, and 500. The results are shown in Fig. 2. We only show the results for PATHFINDER-GP-E (the best-performing across all the toy examples). The NMAE for each technique at each training dataset size was calculated across 50 independent trials, and the average NMAE is reported. Fig. 2(a) shows the mean of the NMAE across the two neuron models, for all the methods. Fig. 2(b) is a zoomed-in version of Fig. 2(a) showing the average NMAE for the three top-performing methods, namely PATHFINDER, Baseline, and MAF. The color bars show the 95% confidence interval. Fig. 2(c) shows the relative decrease in the average NMAE of PATHFINDER compared to the next-best alternative method. To quantify this relative decrease, we plot (NMAE of next best alternative-NMAE of PATHFINDER-Ensemble) at 10 training datapoints. The orange line is the estimated inverse, and the blue line is the original toy forward mapping.

\(^3\)The datapoints in the figure were not used for training PATHFINDER; rather, they were independently sampled from the forward mapping to illustrate the estimated pseudoinverse more clearly.
Fig. 2. (a) shows the average of the NMAE across both neurons with different training dataset sizes constructed from the bio-physical computational neuron models (see Section VI-C). The target firing rate vectors over which NMAE is calculated is as follows: [11 0], [10 0], [11 1], [10 2], [9 1], [8 0], [9 2], [12 5], [8 1], [25 108], [22 101], [23 101], [30 107], [26 103], [30 105], [24 97], [19 92], [26 99], and [26 98], where the first number in the vector is the firing rate of Pyr neuron and the second is the firing rate of PV neuron in Hz. (b) is a zoomed-in version of (a), only showing the average NMAE for PF, Baseline, and MAF. The color bars show the 95% confidence interval. (c) shows the relative decrease in PATHFINDER's NMAE compared to the best alternative at different training dataset sizes. (d) and (g) represent two waveforms predicted by the PATHFINDER algorithm for producing a Pyr-selective response (Pyr target firing rate 10 Hz and PV target firing rate 0 Hz) and a PV-selective response (Pyr target firing rate 23 Hz and PV target firing rate 101 Hz). Note that the target firing rate here refers to the desired firing rate we want to achieve and not the actual firing rate produced by the predicted waveform, which is close but not equal to the target firing rate. (f) and (e) show the corresponding membrane potential in the soma of the PV and Pyr neuron models resulting from the application of the waveform in (d), respectively. Similarly, (i) and (h) show the corresponding soma membrane potential of the PV and Pyr neuron models, respectively, after applying the waveform shown in (g).

PATHFINDER)/NMAE of PATHFINDER × 100 for every training dataset size. The particular dataset sizes investigated for the neuron model case are informed by experimental constraints in collecting datasets (see Section V-A).

Results: We observe that PATHFINDER significantly out-performs the other techniques and has > 20% less NMAE compared to the best alternative for all training dataset sizes (Fig. 2(a)–(c)). Notably, PATHFINDER requires only 50 datapoints to achieve the Baseline method’s performance at 250 datapoints (the best alternative at 250 dataset size). We also observe that only PATHFINDER consistently outperforms the Baseline method. MAF performs better than MDN for the neuron model case. This result aligns with the results from the original work on MAF [11]. This aspect is different in our toy examples (Fig. 1), where MDN performs better than MAF. NI seems to perform the worst out of all the methods tested for the neuron model case. This contrasts to its good performance in the toy examples. We discuss these observations in Section VII.

VII. DISCUSSION

In this section, we first provide discussions regarding the results observed in Section VI, on the relevance of data-driven techniques and the limitations of our study.

1) Data Efficiency of PATHFINDER: A potential reason for PATHFINDER’s data efficiency (seen in Section VI) is what we call the maximization bias. The maximization bias refers to the phenomenon that PATHFINDER tends to estimate the pseudoinverse with the largest number of datapoints. E.g., in \( \cos(\cdot) \) mapping, assume that PATHFINDER only estimates 1 of the 6 pseudoinverses corresponding to the restricted domains \([0,0.5],[0.5,1],[1,1.5],[1.5,2],[2,2.5]\) and \([2.5,3]\). Let \( n_i \) be the
number of datapoints that lie in the restricted domain of the \(i\)-th pseudoinverse. Recall that the regularizer in the PATHFINDER loss tries to give non-zero weights to as many datapoints as possible (Section IV). Consequently, the regularizer encourages PATHFINDER to choose the restricted domain with the largest number of datapoints, i.e., \(\max_i n_i\). For a size-\(n\) dataset sampled using a uniform distribution, we show (in Supp. Section VI) that \(\mathbb{E}[\max_i n_i] = \frac{n}{6} + c\sqrt{n}\) (\(\mathbb{E}[\cdot]\) is the expectation, and \(c\) is a constant). Note that, while the expected number of datapoints in any one restricted domain is \(n/6\), the largest restricted domain has extra \(c\sqrt{n}\) datapoints. PATHFINDER loss encourages inversion in just this restricted domain, and thus, is able to harness these extra datapoints, lowering the required overall sample size. On the other hand, MDN, MAF, and NI, in one way or another, try to estimate the entire forward mapping, and are not designed to harness maximization bias. A more detailed explanation is provided in Supp. Section VI. We aim to precisely characterize the maximization bias in future works.

2) Explanation of How MDN Estimates Conditional density/pseudoinverses: The number of modes of \(p(\theta | r^i)\), in most cases of interest, is determined by the number of parameters \(\theta\) that produces the response \(r^i\), e.g. \(p(\theta | r^i)\) for \(\cos(\cdot)\) forward mapping has 6 modes. For all our toy examples, \(p(\theta | r^i)\) has less than 6 modes but in neuron models, \(p(\theta | r^i)\) has an extremely large number of modes (technically, infinite), due to the dimension of \(\theta\) being higher than \(r^i\). At small sample sizes, MDNs, and in general mixture models, are known to overfit \(p(\theta | r^i)\) when the number of modes is large [43]. This explains why MDNs work well in our toy examples, but not in our neuron models. However, MAF, as a normalizing flow estimator, avoids overfitting due to a large number of modes and performs better than MDN in neuron models. For a \(p(\theta | r^i)\) with a large number of modes, MDN can still estimate pseudoinverses reasonably well, as evidenced in our neuron model study, possibly due to mode collapse [44]. To estimate pseudoinverses, we do not need MDNs to model every mode, but rather just one. E.g., Fig. 3(d) illustrates that MDNs with fewer mixtures than the total number of modes of \(p(\theta | r^i)\), can still estimate \(g^i\)’s pseudoinverse. Therefore, mode collapse actually helps, not hinders, in estimating a pseudoinverse. However, in MDNs, mode collapse is not “controlled” explicitly, and so on average, performance of MDN suffers. PATHFINDER can be viewed as “controlled” mode collapse to learn the pseudoinverse corresponding to the largest restricted domain.

3) NI’s in Neuron Simulation Study: To understand the degradation in NI’s performance from our toy examples to the neuron model study, we performed the following experiment: For the \(\cos(\cdot)\) mapping, we compared the performance in estimation of \(g^{-1}\) in i) traditional NI (where we use the neural network estimate \(\hat{g}\) of \(g\) in (2)), with ii) using the actual mapping \(g(\cdot)\) instead of \(\hat{g}\) in (2) (\(\cos(\cdot)\) in this case). The plot of NMAE vs training dataset sizes is shown in Fig. 3(c). NI using the actual mapping \(g(\cdot)\) achieves a small error for even a small number of data points, whereas NI with estimated forward mapping \(\hat{g}\) does not. The latter observation is surprising because, from the test loss for the estimated forward mapping (shown in Fig. 3(b)), it might seem that the estimated forward mapping is a good fit (especially at 40 and 50 samples). Together, Fig. 3(b) and (c) illustrate the challenge of numerically inverting a deep network: even small inaccuracies in the forward model lead to substantial error upon inversion. Worse, in our neuron models example, the estimated forward mapping has a larger error than toy examples (10.5% even at 1000 data points), due to it being a higher dimensional problem (data requirements for estimating forward mapping grow exponentially in the dimension of \(\theta\) [13]). Therefore, the error in the forward mapping compounded with the errors introduced due to the challenge of numerically inverting a deep network cause a significant drop in the performance of NI for our neuron-model examples.

4) Importance of Data-Driven Techniques: In this work, we discuss data-driven techniques for designing stimuli that produces desired neural responses. Traditionally, such stimuli are designed by choosing 1 or 2 “important” parameters (based on the intuition from the bio-physics of the system) and then brute-force searching across the 1 or 2 parameters to characterize the neural response. For example, [25] considers the frequency of the sinusoidal stimulation as a parameter and brute-force searches across the frequency to characterize the neural response. A limitation of such brute-force search approaches is that they do not scale well as the number of parameter increases.

The most important advantage of the data-driven techniques is that they let us explore a higher number of parameters. To illustrate, from Fig. 2, we observe that brute-force techniques (denoted as baseline in Fig. 2) require 250 datapoints to achieve the same performance as PATHFINDER at 50 datapoints. Hence, PATHFINDER allows an exploration of higher number of parameters due to its data-efficiency. In practice, to an end-user, this
implies that instead of choosing 1 or 2 most relevant parameters, they can now choose 5 or 10 most relevant parameters to search across. The advantage of exploring more number of parameters is that we can discover novel stimuli that can produce neural responses that were impossible to produce by just exploring 1 parameter. Intuitively, data-driven techniques are able to scale well to a higher number of parameters because they exploit the inherent smoothness of the neural response, in one form or other, and provide a more principled way of searching the parameter space.

The data-requirement for these data-driven techniques can be even further reduced by using adaptive sampling techniques. For example, the weight mapping of the PATHFINDER can be used for sampling only from the restricted domain thereby reducing the data requirements or the estimated $p(\theta|\tilde{v})$ in the conditional density methods can be used to adaptively sample around a desired response $\tilde{r}_d,des$, as done in [11]. We leave the characterization of the data-efficiency of the adaptive-sampling techniques for future work.

5) Computational Cost of Data-Driven Approaches: All of the data-driven techniques used in this work were run on a standard laptop (a 2021 Macbook Pro with 8 GB RAM) in around 15-20 minutes. These can be made much faster with hardware parallelization or techniques like network distillation [45]. However, the main cost in optimizing neuromodulation comes in collecting data. For example, collecting the dataset in [2], that used patch-clamp techniques, took about 3-5 months, whereas running MAF (the most computationally expensive algorithm) took $\sim$15-20 minutes on our laptop. The computational cost of the data-driven methods discussed here is negligible compared to the cost of collecting data.

6) Limitations of Our Study: Validation against real world data is not performed. This is due to the lack of publicly available datasets for this problem. We aim to collect such datasets for future works (see Supp. Section VIII). For simulations, we used single cell multi-compartment models of neurons. While existing studies [33] do use single cell models to test their techniques, more complicated simulation study using biological network can be performed, although the rewards from doing more complicated simulation studies might be diminishing. Active sampling techniques, which can offer substantial improvements, are not considered here and are a logical next step. Density estimation is a big field of research, and many alternative techniques to MAF and MDN exist, e.g., [46], and should be explored. Another limitation of data-driven approaches is that the stimulus/parameter space ($\Theta$) needs to be decided a priori. As briefly mentioned in Section II, the choice of stimulus/parameter space decides the range of responses for which the data-driven techniques can design parameters. A poor choice of stimulus/parameter space can adversely affect the performance of the data-driven approaches. Typically, a good parameter space can be chosen based on domain knowledge. Our results are indicative of trends regarding the amount of data required between different techniques, and the absolute number of samples will depend upon many factors such as noise, the dimensionality/size of the waveform family being considered etc. While, our simulation results show that PATHFINDER outperforms existing methods when the dataset size is small in the scenarios investigated, the actual choice of technique will be influenced by the application of interest, and needs to be treated on a case-by-case basis.

VIII. CONCLUSION

In this work, we investigate design of data-driven techniques for designing stimuli to produce desired neural responses. Data-driven techniques allow end-users, such as clinicians and neuroscientists, to explore a larger number of parameters as compared to brute-force search methods, which allow only 1 or 2 parameters. Exploring a larger number of parameters helps design novel stimuli that produce neural responses which are clinically relevant, e.g. in [2], data-driven techniques helped in finding novel electrical waveforms for helping in treatment of Parkinsonian symptoms. It is desirable to make data-driven techniques as data-efficient as possible in order to increase their applicability in different neuromodulation contexts, as collecting data in neuromodulation domain is typically expensive. Towards increasing the data-efficiency of data-driven techniques, we propose PATHFINDER an algorithm that designs stimuli for desired neural responses by estimating a pseudoinverse. We observe that in toy examples and neuron models, compared to existing data-driven techniques, namely MDN, NI, and MAF, PATHFINDER requires the least amount of data for designing stimuli. This study, to our knowledge, is the first study to compare different data-driven techniques for stimuli design and provide a characterization of the data-efficiency of each technique. Data-driven techniques provide a promising alternative to traditional brute-force search of parameters and can help design novel stimuli.

ACKNOWLEDGMENT

The authors would like to thank Dr. Alireza Chamanzar, Dr. Praveen Venkatesh, Dr. Mats Forssell and Sara C. Martinez. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NIWC Atlantic and DARPA.

REFERENCES

[1] E. S. Krames et al., “What is neuromodulation?” in Neuromodulation. Amsterdam, The Netherlands: Elsevier, 2009, pp. 3–8.
[2] T. A. Spix et al., “Population-specific neuromodulation prolongs therapeutic benefits of deep brain stimulation,” Science, vol. 374, no. 6564, pp. 201–206, 2021.
[3] M. Y. Cheng et al., “Optogenetic neuronal stimulation promotes functional recovery after stroke,” Proc. Nat. Acad. Sci., vol. 111, no. 35, pp. 12913–12918, 2014.
[4] D. Chaudhury et al., “Rapid regulation of depression-related behaviours by control of midbrain dopamine neurons,” Nature, vol. 493, no. 7433, pp. 532–536, 2013.
[5] E. R. Kandel et al., Principles of Neural Science, vol. 4. New York, NY, USA: McGraw-hill, 2000.
[6] E. M. Izhikevich, Dynamical Systems in Neuroscience. Cambridge, MA, USA: MIT press, 2007.
[7] P. J. Gonçalves et al., “Training deep neural density estimators to identify mechanistic models of neural dynamics,” Elife, vol. 9, 2020, Art. no. e56261.
[8] P. Bashivan, K. Kar, and J. J. DiCarlo, “Neural population control via deep image synthesis,” Science, vol. 364, no. 6439, 2019, Art. no. eaav9436.
[9] C. R. Ponce et al., “Evolving images for visual neurons using a deep generative network reveals coding principles and neuronal preferences,” *Cell*, vol. 177, no. 4, pp. 999–1009, 2019.

[10] E. Y. Walker et al., “Inception loops discover which excitatory neurons most using deep predictive models,” *Nature Neurosci.*, vol. 22, no. 12, pp. 2060–2065, 2019.

[11] G. Papamakarios, D. Sterratt, and I. Murray, “Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows,” in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 837–848.

[12] M. P. Holmes, A. G. Gray, and C. L. Isbell, “Fast nonparametric conditional density estimation,” in *Proc. 23th Conf. Uncertainty Artif. Intell.*, UAI07, Arlington, Virginia, USA: AUAI Press, 2007, pp. 175–182.

[13] Y.-C. Chen et al., “Nonparametric modal regression,” *Ann. Statist.*, vol. 44, no. 2, pp. 489–514, 2016.

[14] G. Papamakarios, T. Pavlakou, and I. Murray, “Masked autoregressive flow for density estimation,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2338–2347.

[15] C. M. Bishop, “Mixture density networks,” Tech. Rep. NCRG/4288, Aston University, Birmingham, U.K., 1994.

[16] S. E. de Vries et al., “A large-scale standardized physiological survey reveals functional organization of the mouse visual cortex,” *Nature Neurosci.*, vol. 23, no. 1, pp. 138–151, 2020.

[17] A. Dosovitskiy and T. Brox, “Generating images with perceptual similarity metrics based on deep networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, pp. 658–666, 2016.

[18] G. Papamakarios and I. Murray, “Fast $\varepsilon$-free inference of simulation models with Bayesian conditional density estimation,” in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 1036–1044.

[19] G. Papamakarios et al., “Normalizing flows for probabilistic modeling and inference,” *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 2617–2680, 2021.

[20] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.

[21] R. Unni, K. Yao, and Y. Zheng, “Deep convolutional mixture density network for inverse design of layered photonic structures,” *ACS Photon.*, vol. 7, no. 10, pp. 2703–2712, 2020.

[22] H. Zen and A. Senior, “Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2014, pp. 3844–3848.

[23] H.-Y. He and H. T. Cline, “What is excitation/inhibition and how is it regulated? a case of the elephant and the wise men,” *J. Exp. Neurosci.*, vol. 13, 2019, Art. no. 1179069519059371.

[24] M. Avoli et al., “Specific imbalance of excitatory/inhibitory signaling establishes seizure onset pattern in temporal lobe epilepsy,” *J. Neurophysiol.*, vol. 115, no. 6, pp. 3229–3237, 2016.

[25] M. Mahmud and S. Vassanelli, “Differential modulation of excitatory and inhibitory neurons during periodic stimulation,” *Front. Neurosci.*, vol. 10, 2016, Art. no. 62.

[26] M. Komarov et al., “Selective recruitment of cortical neurons by electrical stimulation,” *PLoS Comput. Biol.*, vol. 15, no. 8, 2019, Art. no. e1007277.

[27] N. T. Carnevale and M. L. Hines, *The NEURON Book*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[28] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA, USA: CreateSpace, 2009.

[29] P. Lánský, “On approximations of stein’s neuronal model,” *J. Theor. Biol.*, vol. 107, no. 4, pp. 631–647, 1984.

[30] P. Virtanen et al., “SciPy 1.0: Fundamental algorithms for scientific computing in python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.

[31] K. L. Perkins, “Cell-attached voltage-clamp and current-clamp recording and stimulation techniques in brain slices,” *J. Neurosci. Methods*, vol. 154, no. 1/2, pp. 1–18, 2006.

[32] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*, vol. 2. Cambridge, MA, USA: MIT Press, 2006.

[33] M. Gopakumar et al., “Cell-type selective stimulation of neurons based on single neuron models,” in *Proc. 9th Int. IEEE/EMBS Conf. Neural Eng.*, 2019, pp. 411–414.

[34] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *J. Mach. Learn. Res.*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html

[35] M. Germain et al., “MADE: Masked autoencoder for distribution estimation,” in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 881–889.

[36] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elsu),” in *Proc. Int. Conf. Learn. Representation (ICLR)*, 2016.

[37] F. Pedregosa et al., “Scikit-learn: Machine learning in python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[38] Y. Chen and X. Ye, “Projection onto a simplex,” 2011, arXiv:1101.6081.

[39] D. P. Kingma and J. Ba, “ADAM: A method for stochastic optimization,” in *Proc. Int. Conf. Learn. Representation*, 2015.

[40] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[41] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, [Online]. Available: http://tensorflow.org/

[42] J. V. Dillon et al., “Tensorflow distributions,” 2017, arXiv:1610.09003.

[43] C. N. Davis et al., “The use of mixture density networks in the emulation of complex epidemiological individual-based models,” *PLoS Comput. Biol.*, vol. 16, no. 3, 2020, Art. no. e1006869.

[44] L. Theis, A. van den Oord, and M. Bethge, “A note on the evaluation of complex generative models,” in *Proc. Int. Conf. Learn. Representation*, 2015.

[45] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” in *Proc. NeurIPS Deep Learn. Workshop*, 2014.

[46] W. Grathwohl et al., “FFJORD: Free-form continuous dynamics for scalable reversible generative models,” 2018, arXiv:1810.01367.