Neuronal Deep Fakes

Data Driven Optimization of Reduced Neuronal Models

by

Russell J Jarvis

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2020 by the
Graduate Supervisory Committee:

Sharon M Crook, Co-Chair
Rick C Gerkin, Co-Chair
Yi Zhou
James J Abbas

ARIZONA STATE UNIVERSITY

December 2020

ABSTRACT

Neuron models that behave like their biological counterparts are essential for computational neuroscience. Reduced neuron models, which abstract away biological mechanisms in the interest of speed and interpretability, have received much attention due to their utility in large scale simulations of the brain, but little care has been taken to ensure that these models exhibit behaviors that closely resemble real neurons. In order to improve the verisimilitude of these reduced neuron models, I developed an optimizer that uses genetic algorithms to align model behaviors with those observed in experiments. I verified that this optimizer was able to recover model parameters given only observed physiological data; however, I also found that reduced models nonetheless had limited ability to reproduce all observed behaviors, and that this varied by cell type and desired behavior. These challenges can partly be surmounted by carefully designing the set of physiological features that guide the optimization. In summary, we found evidence that reduced neuron model optimization had the potential to produce reduced neuron models for only a limited range of neuron types.

ACKNOWLEDGMENTS

This body of work was a large international team effort that was only possible because of continuous attention from diverse faculty at ASU.

First and foremost I would like to thank my committee: Professor Sharon Crook, Professor Rick Gerkin, Professor Jimmy Abbass, and Professor Yi Zhou. Additionally Sharon Crook and Jimmy Abbas offered significant unexpected personal support. Lastly, Rick poured many hours of his life into consulting about this project.

Other sources of general support came Professor John Alcock who allowed me to lodge for free in his house and, SOLS Graduate Program Associate Directory Professor Emilia Martins, Professor Julie Liss, Professor Susanne Neuer, and Professor Bradley Gregor, were called on. I would also like to thank Amy Pate who helped with VR and Augmented reality technology.

I would also like to thank the late Jane Hurley, a recent PhD graduate from ASU Department of Exercise and Nutritional Sciences, who put significant work into making Arizona desert accessible to outsiders, and offered significant personal support. Also from Arizona, I would like to thank Maria Grimshaw Clark, and Kendra Hunt. I would also like to thank the Anthony Nicholas for inspiration and Imogen Hamel-Green, and the Hamel-Green family.

I would also like to thank my parents, as well as the late Jane Hurley, a recent PhD graduate from ASU who offered significant personal support.

“When applying digital methods, you may have to try very many different things." – Emilia Martins

I would also like to thank Research Professional Renate Mittelmann for supporting an early and difficult transition to Docker-driven development.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Why Model the Brain?

Diseases of the brain are very widespread and cost world governments a significant amount of money (Di Luca *et al.*, 2018). The 2015 Global Burden of Disease study estimates that about a third of the population worldwide is affected by mental or neurological disorders across their lifespans. Disorders of the brain rank among the leading causes of ill-health and disability and account for 35% of Europe's total disease burden with a yearly cost of 800 billion euros, of which 60% are related to direct health care and non-medical costs (Di Luca *et al.*, 2018).

1.1.1 Computational Models

In addition to producing actions and goal directed behaviors, the mammal nervous system filters and processes signals in steps that are increasingly understood at mechanical and algorithmric levels (Marr and Poggio, 1976). The recent success of “deep-neural networks" and “recurrent neural networks" is due in part to the simple application of rudimentary neural design principles to networks of computational units. Innovations in artificial intelligence (AI) have been achieved by recapitulating just a few such algorithms observed in the brain. For example, artificial neural networks trained to navigate a landscape spontaneously exhibit the emergence of nodes (cell analogues) exhibiting the same behavior as so-called “grid" and “place" cells that encode spatial location in mammalian brains (Banino *et al.*, 2018).

However, further progress in both human health and artificial intelligence (AI) is likely to require a greater understanding of how the brain actually works. Biophysically realistic computational models of neurons and neural circuits have emerged as useful tools to elucidate brain function in health and disease. Developing and testing these models may be required to give programmers and engineers insight into the neural principles that underlie human learning and reasoning, so that these principles can be implemented in an electronic substrate. Because improvements in AI directly contribute to improvements in robotics, and because computers and robots can perform some unsafe and repetitive tasks with high precision, substantial gains in global economic productivity are at stake. AI may also help to push the frontier of scientific research itself. Our medical understanding of the brain and digital implementations of brain algorithms would likely improve if existing models for neuron and neural circuits were more tractable, both conceptually and computationally.

In this thesis, I describe and document tools I developed that improve the speed and accuracy of some classes of neural models. The speed improvements come from the implementation of a low-level model simulation, largely relying on existing research. I also identify and exploit some opportunities for faster model dispatch and simulation. The accuracy improvements, which constitute most of the novelty here, derive from application of optimization techniques to improve the agreement between models and corresponding “wet lab" experiments in neurons. By improving single cell models, these tools will also serve as a basis for improved neural circuit models, of which many neuron types are among the components. Since network models many consist of many thousands (or more) of cell models, both speed and accuracy are essential.

1.1.2 Simulation as an Experimental Platform

The human brain is inaccessible to many experiments for technical and ethical reasons. Neuroscience relies instead on “model systems" for understanding the human brain. One such model system is the animal model (of which there are many), chosen due to biological similarities to the organization of the human brain, and the relatively smaller ethical issues associated with experimentation. However, animal models have limitations which can make them scientifically unfavorable Animal models often involve years of investment, genetic engineering, complex surgery, or complicated behavioral learning paradigms. Many things can go wrong while developing an assay, potentially ruining a costly experiment, and diminishing years of work. Furthermore, experiments are still limited by technology; arbitrary control of neural activity is generally not tractable. Therefore causality and identifiability are still major issues.

For example, in a rodent model of Alzheimer’s disease, it is possible to investigate rodent behavior for Alzheimer’s symptoms, and to attempt to identify associated changes in physiology, such as changes in cell-specific firing rates or circuit dynamics. However, the density of placement of recording electrode arrays is very limited, and so the read out of brain activity from electrode arrays is sparse. Immune reactivity and glial scarring may undermine the fidelity of the underlying system, and make such recordings more difficult. In *in vitro* recordings, recorded cells are susceptible to “run-down" (Colquhoun, 1994), where the membrane of the recorded neuron membrane its ionic complement changes too much from its *in vivo* conditions. Cell run-down creates a small time window for learning about cells in animal experiments, fortunately, cell-run down is obviously not a problem for *in-silico* models.

In-silico model systems, e.g. computational models represent an alternative approach. Rather than simply thinking of computational models as systems of differential equations used to describe and represent the biological processes under investigation, we can think of them as experimental platforms. Much detail will inevitably lost, and a main assumption is that the important features of the biological processes are reflected in electrical activity, e.g. changes in membrane potential, and that other details abstracted out of the model, for example long-term changes to genes and protein expression, are not essential to understand the problem at hand. This assumption may be reasonable for investigations of short-time scale phenomenon (e.g. instantaneous responses to stimuli) but not longer-time scales (e.g: glial cell re-organisation associated with learning new skills (Draganski and May, 2008)).

However, all model systems are imperfect; the *in vitro* brain slice is not an *in vivo* brain, and lacks many putatively important features of a real brain, such as long-range connectivity or even complete axonal or dendritic trees. The limitations expressed in the paragraph above may limit measurement quality. Ultimately, the question is whether results from the model system will translate to the biological system, and how and whether a good simulation platform can outperform an *in vitro* animal model system is an open question. To achieve this, it is likely that the simulation platform will need to utilize model components as faithful to the biological system as possible, and thus it is important that reduced neural models are the most faithful to the cell behavior they represent.

1.2 Why Model Neurons?

The idea that neurons are the fundamental computational units of the brain dates to the 19th century (Shepherd, 2015; Jones, 1999). A large fraction of computational models have thus included the neuron, at some level of abstraction. Conventionally, each neuron has been regarded as an independent generator of events (e.g. synaptic conductance's or currents in postsynaptic neurons) and selective receiver of the same events, these inputs summed in some fashion across time and space.

Although cortical neurons have many interconnections (Ball *et al.*, 2014), these connections themselves are plastic, changing with development and learning. In contrast, the dynamics of the neurons themselves are typically considered less variable over most relevant timescales, i.e. the parameters of the underlying equations may be fixed. Consequently the rules governing these neurons are assumed to be static over short time-scales. whicat the center of modeling efforts, and letting only synapses change over time, one can still model a large number of relevant neuroscience problems. But just because the modeled neurons can be approximated with static rules does not mean they should be considered monolithic: there are many thousands of different classes of neurons, each with its own idiosyncratic dynamics and connectivity. And many such classes might be essential for the generation of even simple behaviors. Therefore understanding the brain may require the ability to accurately model not just neurons per se, but many kinds of neurons.

Alternatively, one could model subcellular processes to get an even more accurate picture of biology than is obtained from considering the neuron as the basic unit of the brain. However, this approach greatly increases computational complexity and is even more limited by the availability of experimental data (or theory). It is an open question whether such detail improves the predictive or explanatory accuracy of models anyway. Simplifying assumptions are often needed in modeling, and the simplifying assumption that I choose here is to abstract away these subcellular processes into a small number of state equations that describe observable physiology at the whole cell level.

One could also model above the level of neurons, by creating mean-field and population models of brain tissue. However, this makes it difficult to understand intra-circuit computations occurring in local circuits, or to understand how a change to e.g. a single conductance might affect a behavior. Some also argue that spiking is necessary to reproduce some kinds of population dynamics, especially those that evade intuitive understanding. And model-as-simulation-platform is less compelling if the outcomes of the simulations are obvious in advance.

In AI, spiking neural networks are useful for distinguishing sequential features in temporal signals. Spiking and temporal feature discrimination are necessary for "online-learning". In online-learning, neural networks are constantly being remodeled in response to constant and changing signals from the environment

1.2.1 Conductance-based Models

Conductance-based models are mathematical representations of the electrical and chemical behavior of semi-permeable neuronal membranes. The phospho-lipid membrane of neurons, otherwise a insulative capacitor, contains a variety of conductive pores (ion channels), each permitting the passage of specific ions at specific rates. These rates are determined by free energy gradients, jointly determined by transmembrane ionic concentration gradients (entropy) and on the transmembrane voltage changes adjacent to the channel (potential energy). Conductance-based models use ordinary differential equations to represent the balance of currents and the kinetics for the gating of the channels, where these models differ in their representation of cell geometry.

Some conductance-based models represent an isoelectric patch of membrane (in this case the model is abstract and dispenses with the notion of space in nervous tissue). When the spatial aspects of the electrical properties of a neuron are of interest, conductance based models can be combined with a multi-compartment framework where the geometry of the neuron is "modelled", but it is approximated using coupled compartments that create systems of branching patterns (Rall, 1962; Hodgkin and Huxley, 1952).

Rall (1962) took the numerical methods used for modelling heat and electrical conductance in cables and showed that similar equations could be used to accurately predict current flow in a cable representing a neuron dendrite or axon. This cable equation forms the basis for the electrical properties in the multi-compartment approach.

In the context of modelling realistic morphologies, conductance based, multicompartment models are typically slow to solve because they contain many (tens or more) coupled differential equations which must be solved at every discrete location across the neuron. These locations (segments or compartments) may have unique parameters, which are as simple as segment diameter (corresponding to e.g. tapering of dendrites) or as complex as distance-dependent changes in the distribution of specific ionic conductances. Many of these fine structural variations are incorporated into biophysically-detailed brain models. Because the geometric structure of the morphology affects the neural firing dynamics, even a generic set of principles may result in distinct dynamics across different realizations of the same class of neuron. Clearly, there multicompartment modeling brings a large cost in simulation performance and in generalizability.

1.2.2 Reduced Models

Conductance-based models can be complex–both conceptually through implementation details and in computational demands–so it is common to employ a variety of simplifications that retain the general input/output behavior of the full model. These "reduced models" ignore the precise neuron biophysics, usually retaining only the medium-time-scale (milliseconds to tens of milliseconds) dynamics, by assuming that anything faster occurs instantaneously and anything slower can be ignored. The 3D structure of biological cells is reduced to a point onto which currents impinge and voltages evolve over time. There is a great diversity of biological neurons, all of which differ substantially in their electrical behavior (Quadrato *et al.*, 2017). There are a few different classes of general purpose neuronal models that can reproduce these different types of electrical behaviours, given appropriate parameterizations of the models. A selection of key reduced models is described below.

1.2.3 Adaptive Exponential Integrate and Fire Model

The Adaptive Exponential (AdExp) IF model (Brette and Gerstner, 2005), is another type of reduced neural model The AdExp model is a special instance of leaky integrate and Fire model. The AdEx model most often appears with an exponential spike shape. For a full derivation of the adaptive exponential model see the publication: Brette and Gerstner (2005) This model's instantaneous spiking rate is calculated by consulting the recent spike history in a time "window". Adaptation is achieved by looking back into the window and counting the number of spikes that occurred in the previous $10ms$, for example, the instantaneous value of firing frequency is able to reflect adaptation or slowing of firing frequency as observed in-vitro.

1.2.3.1 Izhikevich Model

One existing family of neuron models, often referred to as the Izhikevich model (Izhikevich, 2003), was published with parameter sets believed to produce model outputs that accurately align with a variety of outputs from experimentally derived neuron data. Is it possible to choose parameters for the Izhikevich model that allow the electrophysiological behavior of any neuron to be capture?. The original formulation of that model involves the integration of a quadratic function. For the derivation of the Izhikevich equation see the publication: Izhikevich (2003)

An alternative, more versatile implementation (which is used here) includes additional parameters. The additional parameters allows the coefficients in the quadratic to vary.

The Izhikevich model and the AdEx model are both related to simple Integrate and Fire model. The Izhikevich model accounts for the change in membrane potential, $\frac{dV_M}{dt}$, in response to and externally supplied current (e.g. from a patch pipette or a synaptic current), the membrane capacitance, and internal cellular mechanisms that are abstracted away. $\frac{dV_M}{dt}$ in the Izhikevich model is quadratic in V, in contrast to the GLIF and the AdExp models which are exponential in V.

Typically, a network simulation is only as fast as the slowest neuron model that is part of the network, since the next updated state of the network may depend on that neuron. Consequently, replacing biophysically detailed models with the reduced models described above may be required to complete simulations in a reasonable amount of time. However, the accuracy of these models in representing neuron behavior is also paramount. Yet there are no reports of large numbers of reduced models with demonstrated agreement with many characteristics of experimental data.

Because multi-compartment conductance models represent more details about the biophysical properties of the membrane, it is tempting to think that this modelling paradigm is likely to produce models that maximally agree with data, however, newer generations of simpler models such as the GLIF model, may actually be better at recapitulating some experimental measurements (Meunier and Segev, 2002).

1.2.4 Simulation of Reduced Models

Although this class of model is in principle potentially fast due to the limited number of equations and state variables, python implementations of it are typically slow due to code that was written to make running populations of many neurons fast, at the expense of running single neurons.

1.3 Parameter Fitting and Model Optimization

Model parameters dictate model behavior, but many parameters either (a) cannot be easily measured in experiments for technical reasons or (b) don't correspond to any measurable quantity. As an example of the latter, consider any "emergent behavior" with observables that do not map directly onto any specific model parameter. In the transition from conductance-based models to reduced models, the number of unknown parameters of form (b) increase relative to those of form (a). In either case, these unknown parameter values make specification of realistic models difficult.

One strategy for inferring model parameters, sometimes called “parameter-fitting", is to identify those parameter values which produce model simulation data which agree (according to some criteria) with corresponding observable experimental measurements. For example, a common approach for approximating unknown ion channel densities is to ‘optimize’ or iteratively simulate the model, altering the parameters systematically until the simulated time series for V matches some membrane potential trace recorded under similar conditions in a neuron.

What does it mean for the simulated and experimental time series of V to match? Recall that both neuron models and neurons can be injected with specific amounts of current. Because neurons are electrically excitable, the somatic membrane potential departs from the resting membrane potential in response to current injections. For many classes of cells, abundant experimental data exist concerning the responses (membrane potential time series) to each magnitude or shape of somatic current injection. A crude measure of error would be to simply directly compare the model and biological time series, using the mean-squared-difference, for example. However, because excitability reflects a non-linear transformation between input and output, this means that very small differences in input (or parameters) can have massive differences on output, especially its timing. A single action potential shifted by a millisecond (approximately the width of one action potential) might be insignificant from a practical perspective, but produce a massive “error" signal using mean-squared difference. Consequently, I use an approach based on more robust statistical summaries of the membrane potential, including those believed to be meaningful in neural coding. These are described in more detail below.

The process of optimization involves what is known as an “inverse" problem. A naive approach to solving this inverse optimization problem works on the principle of elimination. A large number of unsatisfactory parameter sets must be sampled, to rule out the possibility that each candidate parameter set is not worse than some currently favored parameter set. Doing this exhaustively, using a grid search across all combinations of parameter values with sufficient resolution, is often computationally intractable. Consider the Izhikevich model above. In its most general form, it contains 10 tunable parameters. Evaluating even 25 values of each parameter implies $25^1 0$ parameter sets, or nearly 100 trillion parameters sets. Even if one million parameter sets could be checked every second, evaluating all parameter sets would require three years, where this cost grows exponentially with increases in parameter resolution or number.

Fortunately, more efficient approaches exist, but the utility depends on the shape of the so-called “error surface". For any specific instance of a model, with a given set of parameter values, its suitability is evaluated according to some "objective function" that compares the simulated output under one or more conditions to some corresponding experimental data. By convention, the objective function may report this suitability as an error metric, minimal when the simulated and biological results disagree to the smallest extent possible (which may not be zero, see below). Model “optimization" then corresponds to finding the parameter set which minimizes this objective function. Because the objective function returns a measure of error, the “error surface" corresponds to the output of the objective function for each possible set of parameter values. Ideally, this surface is perfectly “convex", meaning that there is a single minimum, and from any starting point one can follow some variant of a gradient descent strategy to find that minimum.

1.3.1 Genetic Algorithms

A variety of numerical optimization techniques exist, where algorithm performance depends on the demands of the problem. For functions with many local minima, it is often best to broadly sample the error surface, rather than attempt a “greedy" search for the global minimum. Several notable algorithms including genetic algorithms (GA, Figure 1) and stochastic gradient descent (SGD) have been developed to address this use case. Both of these algorithms are able to avoid putting all of their "eggs" in one local minimum “basket", allowing them to discover over many iterations that a lower minimum is available. For example, given enough samples, a genetic algorithm will learn the global minimum of the Rastrigin’s function, although it is very likely to sample wells of the error function at many depths on the way to the solution.

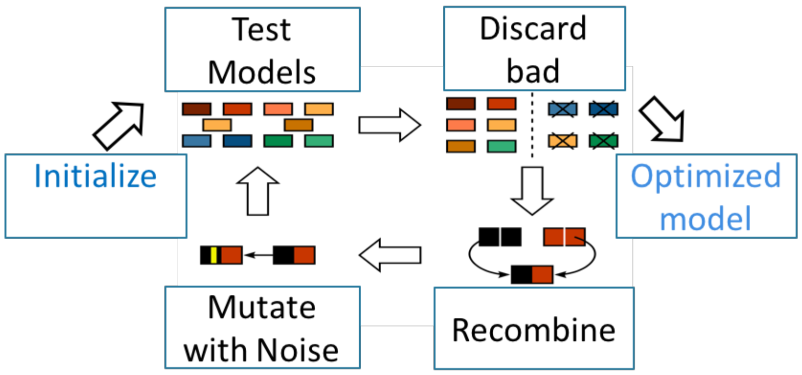


Figure 1: **Schematic of Genetic Algorithm Flow.** Genetic algorithms (GA) find satisfactory solutions by incompletely sampling the solution space. The evolution of the algorithm is guided by the combined application of stochastic sampling and selective pressure. Stochastic contributions that guide GA sampling provide incentives for exploring regions beyond the local minima. Stochasticity is applied in the combined actions of cross-over and mutation. Model parameterizations are encoded as binary strings called genes. When genes breed, eligible pairs of genes are aligned and the status of a bit is exchanged at random locations.

I use genetic algorithms in the work presented here, and while there are many variations on this approach, they all involve some combination of the following features:

1.3.1.1 Chromosomes

In the context of genetic algorithms, a chromosome is the complete set of model parameters that are necessary to fully define the model being optimized, in this case a neuron model. These parameter sets are collections of floating point numbers, for example -65.7 for a reversal potential in mV.

1.3.1.2 Genes

In the context of genetic algorithms, a gene is a single model parameter. Consequently, the number of genes in a chromosome is equal to the number of parameters in the model. There are at least two stochastic operations that over time provide a gentle “exploratory drive" for the algorithm, that cause genes to either double down on exploration of a local minimum, or possibly “jump" out of a local minimum to realize lower error. These operations are applied, with some user-determined frequency, with each “generation" of the population of chromosomes, to determined the members of the subsequent generation.

1.3.1.3 Simulated Genetic Recombination: Cross-over

Cross-over is the exchange of genes between chromosomes. In a population of many chromosomes, one can “mate" two of them, producing off-spring which represent a fraction of genes from parent 1 and the remainder from parent 2. If each chromosome represents two opposing vertices of an n-dimensional hypercube (for n model parameters), then cross-over is capable of producing offspring representing any of the other vertices in that hypercube.

1.3.1.4 Simulated Genetic Mutation

A mutation reflects a (random) change to a single gene, i.e. a change to a single model parameter. In some implementation this is achieved by flipping one bit in the binary representation of the floating-point value of the parameter, with the magnitude of the resulting change in the parameter value depending on which binary digit was flipped. Often mutations are limited to a particular range for a given parameter.

Later in this work I will talk about a property denoted η this property controls the magnitude or step-size of gene changes made via either mutation or cross over. In the cross-over context, η describes how similar child chromosomes are to their parents. In the context of mutation, η changes the location of bits (from Least Significant position to Most Significant Bit). At specified bit locations the bits that encode genes are randomly toggled on or off in mutation.

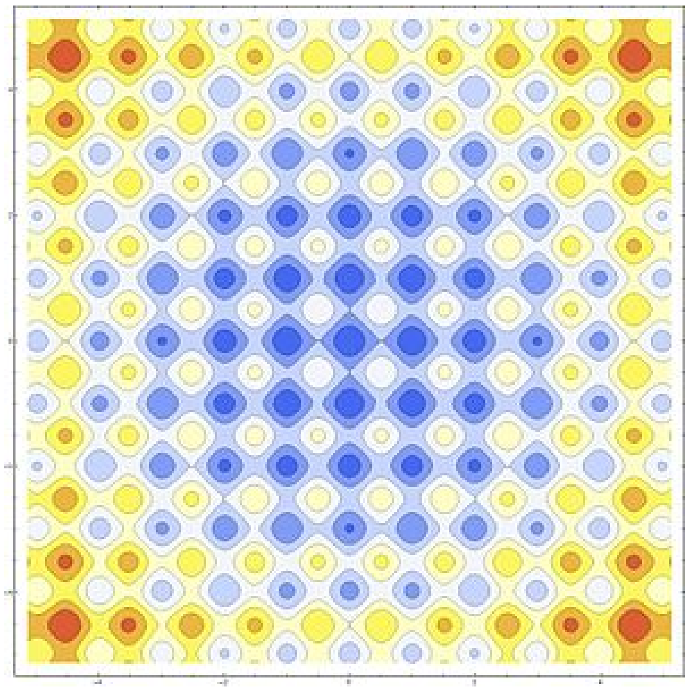


Figure 2: **Rastrigin's Function** Rastrigin's function is challenging to optimize, as it contains hundreds of local optima which would trap a gradient descent algorithm (Rastrigin, 1974). This function offers a paradigm for testing and examining the capabilities of an optimization algorithm.

1.3.2 Local Minima

Many practical optimization problems have error surfaces that are not perfectly convex but instead are rippled, with multiple troughs of similarly low error value (local minima). These ripples make navigating the error surface non-trivial. Rastrigin's function (Rastrigin, 1974) is a function for creating error surfaces to test the robustness of algorithms that attempt to overcome moderately difficult error surfaces that are densely populated with local minima, but with a global convex shape. The surface described by the function is densely populated by local minima, each of which is placed on a globally convex envelope, such that the true minimum of Rastrigin's function is at its centre. As a certain number of genes are retained between generations, genes made on previous generations act as memories of previously experienced error surface. Due to this memory effect, the genetic algorithm is able to exploit the global shape of Rastrigrin's function 2. It should be noted that there are error functions that contain no useful or learn-able information in the error surface. A satisfactory solution to such problems might still be obtained due to the exploratory and stochastic nature of the GA, however, much of the advantage of the genetic algorithm approach is lost, as the approach presupposes learn-able features about error surfaces.

1.3.2.1 Elitism

The number of chromosomes (model instantiations) in each generation is typically fixed, with some chromosomes in generation $n+1$ coming from crossover of chromosomes in generation n and others from mutation. The remainder are often chosen from the fittest, i.e. lowest error, members of generation n, a strategy known as elitism. The degree of elitism essentially controls the amount of evolution that is allowed to occur in each generation.

1.3.3 Multi-objective Optimization

Multi-objective optimization problems are a subset of optimization problems. In a multi-objective optimization, model fitness is evaluated using multiple "objective functions" of model fitness, each contributing either independently or in conjunction to the error surface. Each function may ask a different question about the model; for example, one may ask how well the model reproduces the shapes of action potentials, and another function may ask how well the model reproduces the counts of action potentials. The output of each such function is a scalar, with a lower values representing greater model/data agreement, i.e. lower error. These functions are sometimes called constraints.

1.3.3.1 Weighted Optimization

A simple strategy is to turn a multi-objective optimization problem into a single objective optimization problem, for example by summing the outputs of objective functions together. This is an common strategy, but this approach comes with a significant price. Optimization using a sum over multiple error measurements leads to a situation where a single constraint has an especially deep error surface, such that the gradient of the overall summed error surface is dominated by that single constraint. In other words, minimizing overall error mostly corresponds to satisfying only one constraint. The optimizer invests most of its search effort achieving a low error on one constraint, leaving the others unacceptably high. This problem can in principle be ameliorated by re-scaling the objective functions to have similarly deep error surfaces, but in general the shapes of the error surfaces cannot be known in advance, so this sort of approach cannot generalize.

1.3.3.2 Non-dominated Sort

A less appreciated but perhaps more important drawback to collapsing multiple objectives to a single one is that it imposes an opinion about the relative importance of each objective. However, while there may be underlying “true values" for measured biological variables, there is no underlying truth about which variables are more important than others. In problems where not all measurements can be perfectly replicated by models (in part because the models are simplifications of biology), this problem becomes even more pressing. The experimental data may also come from multiple data sources, leading to solutions for a single objective that reflect the diversity of the underlying system (e.g. physiological diversity of neurons of a given type), which is lost with the collapse to a single objective.

A multi-objective optimization paradigm that admits diverse sets of solutions to a list of constraints might better reflect biology, while also imposing less opinion about which measurements are most important. The so-called “non-dominated sort" is one such paradigm. This approach considers each objective function separately, and only considers one chromosome to be superior to another if it “dominates" that other, meaning that the objective function output is lower (i.e. smaller error) for each objective function. If even a single objective function has a larger value for the latter chromosome, then it is not dominated. Any chromosome that is not dominated by any other chromosome is considered “non-dominated" and is retained (i.e. for mutation, cross-over, or elitist selection) into the next generation.

The non-dominated set of chromosomes has a simple interpretation: each of its members would minimize a single, weighted objective function for some set of weights. If no other chromosome has strictly equal-or-lower objective function evaluation for all constraints, then there exists a linear combination of weights which would give that chromosome the minimal evaluation (minimal error, maximal fitness) across the entire population. Thus, retaining this population is equivalent to retaining all possible “opinions" about appropriate weights for the entire optimization process.

The NSGA2 algorithm (Deb *et al.*, 2000) is a common implementation of this strategy, and is used throughout this work. Its performance can be compared to, for example, an exhaustive "grid search" across sets of model parameters, at least in low dimensions.

1.3.4 The Curse of Dimensionality

Brute-force exploration of parameter space requires computation time that scales exponentially with the number of parameters. In contrast, more efficient searches can be polynomial (e.g. linear or quadratic) in dimension number. However, if constraints are weak, or conflicting, or the error surface is especially non-convex, many of these gains are lost, and exponential search times may return. Thus, there is no guarantee in advance that reduced model optimization, with existing experimental data, is a problem that can be solved in reasonable time for sufficiently high-dimensional reduced models.

1.4 Data Fitting and Optimization with NeuronUnit

The ASU ICON laboratory previously developed a Python package called "SciUnit" for scoring scientific models according to their agreement with various experimentally-derived data (encoded as "validation tests") (Omar *et al.*, 2014). The objective functions that I use here are implemented using this scoring process. In particular, I used a SciUnit-based Python library called "NeuronUnit" which provides a number of such validation tests for neuron models, and I developed additional tests in the course of my research.

1.4.1 Constraint Scoring

The scoring mechanism within these tests determines the output of each objective function to be minimized. There are many candidate scoring mechanisms, depending on what one means by “the model agrees with the data". Each first computes some summary of model output, for example the resting membrane potential, called a “feature". Here, the two most frequently used scoring mechanisms are: (1) compute the ratio between the model output feature and the experimental data feature mean, and (2) compute a Z-score reflecting the normalized distance between the model output feature and the distribution of experimental data features. In both cases, the relevant experimental data distribution might be within trials, across trials, across recorded cells, or even across labs, producing many sources of hetereogeneity. For example, resting membrane potential (measured in (mV)), may vary in some dataset according to a distribution with a mean and standard deviation of $(\mu, \sigma) = (-65mV, 15mV)$. The optimizer may sample a model (with some parameters) with a simulated resting potential of $-68mV$. NeuronUnit would then use these values to compute a Z-Score of $(-68 - (-65))/15 = 0.2$. A model with a Z-Score of 0 would reflect perfect agreement with experimental data, and very low or very high values would represent poor agreement. Our lab previously showed models can be scored on very large numbers of such tests, corresponding to experimental data for particular neuron types, and then used to build biophysically detailed neuron models for the mammalian olfactory bulb (Birgiolas, 2019).

1.4.2 Data Aggregation

NeuronUnit thus converts a quantitative measure of model/data agreement into a useful error signal that can guide model optimization. But a second function of NeuronUnit is to aggregate the relevant experimental data, in order to make construction of the tests described above possible. And a third function is to apply some signal processing and feature extraction steps to these data in order to produce something that can be scored. These features were present at the beginning of this project, but I have extended them to cover more datasets and more features, as described in the Methods.

1.5 Model Simulation and Exchange

The NEURON (Carnevale and Hines, 2006) simulator is a software suite that wraps powerful and fast ordinary differential equation solvers based in the C programming language inside a mixed compiled/interpreted environment targeted at research scientists. NEURON is somewhat analogous to older, analog circuit simulators; however, rather than describing complex resistor-capacitor circuits, NEURON instead solves equations for the time varying membrane potential of multi-compartment models.

In contrast to reduced models, multi-compartmental models digitally encode the form of membrane tissue: cables of varying diameters and lengths that represent the morphology of neurons, and these cables support smaller scale representations of ion channels, and ion currents in the membranes. These neuronal models can be coupled together into a network, where the electrical state of one neuron has an impact on the state of coupled neurons through synaptic currents. Specifying the system of differential equations representing these neuronal morphologies, ion channels, and synaptic connections is complicated, but *NEURON* makes multi-compartment neuron simulation efficient, convenient, and achievable. Models expressed in *NEURON* code are procedural in nature, and the code consists of low-level implementation details. Procedural descriptions of models are difficult to extend and re-use, leading to a need for a declarative model description language.

NeuroML (Gleeson *et al.*, 2010) is an extensible markup language (XML) tasked with describing these complex network models. In principle, any simulator (not just NEURON) can execute a simulation based on a NeuroML file, so NeuroML models are readily executed across different types of simulators (ideally with identical results). This facility makes NeuroML an excellent format for model sharing, and permits cross-examination of results if they vary across simulators, but also allows different modeling communities to retain the tools that they know best. Because NeuroML is extensible and component based, it also incentivizes a “plug-in" environment for re-using existing model components in large-scale models for different contexts. These existing components may include reduced models, for example embedded into larger neural circuits.

Open Source Brain (Gleeson *et al.*, 2019) is a collaborative, model development environment based on the NeuroML format, which contains collections of model-descriptions and model implementations, along with tools for simulation and visualization. These collections are well tested and understood, and the Open Source Brain platform was a practical tool for learning the Izhikevich models correct parameter ranges (which can vary depending on preferred units).

NEURON focuses on efficient simulation of multi-compartmental models, while NeuroML provides for model exchange. However, neither tool was designed with the optimization of reduced models in mind. In optimization, the same model may be re-executed hundreds of thousands of times (with different parameter sets), so any computational overhead (for example, associated with reading/writing a file) is quite costly. There are other tools that may result in faster optimizations, but as I show here, even these are not optimally fast. Consequently, it is best to think of tools like NEURON and NeuroML as useful for running and sharing optimized models, but not necessarily useful for the optimization process.

1.6 Analysis of Model and Experimental Variance

It is common to observe large variations in measurements of a single electrophysiological entity from neurons of the same classification (Tripathy *et al.*, 2014). For example, measured input resistance may be different when recorded from different cerebellar Purkinje cells. Such differences may be due to intrinsic physical differences in cells (for example different surface area), due to differences in experimental methods across organisms or investigators, or simply due to the finite and limited number of samples available in the experimental literature. Aggregated data have been collected from mice, rats, marmosets, macaques, or even humans, where human data may have been collected under any number of disease states.

Understanding and modeling within-neuron-type variation should be considered an essential component of the scientific merit of computational modeling of a neuron-type. However, the impact of this experimental variation on the production of neuron-type-specific models has not previously been explored. Consequently, I will present here a large-scale analysis of the consequences of this variability–across many electrophysiological features–on the effort to produce canonical reduced models of several common neuron-types. We would like to understand whether the space of existing or potential neuron models accurately represents the variability in the experimental data, and to help researchers map data collected from a number of experimental preparations onto a standardized rodent electrophysiological phenotype space.

1.6.1 Relationship between Missing Data Imputation and Optimization

Existing data sets are incomplete, consisting of a sparse sampling of cells in the rodent brain. By necessity, models are constrained using these incomplete data sets. These missing data occur at multiple levels including exact neuron to neuron wiring patterns, un-sampled morphologies, unknown synapse activation times, and unknown axon and dendrite synapse locations. More relevant to reduced models, even very basic information about spiking patterns in response to simple current injections may be unavailable. However, since most fast numerical methods are incompatible with missing data, an additional component of model development–especially in the context of optimization–remains, which is the synthesis of plausible missing values.

1.6.2 Closed Loop Development

The speed of simulation is critical to successful optimization. However, even when using High Performance Computing (HPC) resources, errors may arise and require special handling or visualizations to diagnose. These errors may even reveal errors in the model specification, but this may not be revealed until thousands of simulations have already run. The ability to address and correct such mistakes requires rapid and verbose feedback during optimization, as well as simulations of sufficient speed to allow error “breakpoints" to be reached in a reasonable amount of time.

1.6.3 Successful Optimization

Integrating the above ideas, I aimed to develop a framework for rapid optimization of reduced neuron models subject to multiple constraints forged by experimental data, and to use this framework to optimize and share such models. Here I will describe a number of methods and simulation experiments which demonstrate the successful implementation of many of these ideas, focusing on genetic algorithms that retain a diverse set of model parameterizations, potentially reflecting within-type neuronal diversity.

The interactions between model constraints cannot always be known in advance, and in fact some constraints are mutually exclusive, either causing optimization to fail, or leading to a compromise. I also aimed to catalog such cases and use this knowledge to produce a more robust set of constraints, implemented via *NeuronUnit* tests for future optimization work.

No model–including those that are published– should be regarded as final. Instead, I aimed to develop tools to improve these models by making them agree with experimental data, in the hopes that such models may be of general use to the neuroscience community in future work. At best, replacing complex cell models with simplified models is a way to reduce the complexity of electrical brain activity to fundamental mechanisms (Teeter *et al.*, 2018). At worst, simplified cell models may capture only a subset or a single aspect of important traits such as spike times (Hertäg *et al.*, 2012).

There are currently many optimization packages for optimizing multi-compartment conductance based models (Friedrich *et al.*, 2014) (Van Geit *et al.*, 2016b) (Vella and Gleeson, 2012), these optimization packages are exclusively intended to work with the *NEURON* simulator. There are at least two packages for optimizing reduced neuronal models, but one the these packages is used to fit models to spike times (Rossant *et al.*, 2010), not spike shapes. This optimization framework uses specialized hardware and software GPU and CUDA, so this solution is not accessible to the average neuronal modeller.

Additionally, the Brian2 simulator supports some model fitting (Stimberg *et al.*, 2019b),(Stimberg *et al.*, 2019a). Although Brian2 can fit spike shapes by comparing traces, the type of optimization performed is not multiobjective, as it computes the the $RMSE$ of whole waveforms, and uses it as a single objective error function. Brian2 can also optimizes on spike times, but its not clear if it does this in a multiobjective context. Additionally, performance of brian2 was not reliable.

Some core components of the optimization procedure detailed here do not deviate from those described elsewhere (Druckmann *et al.*, 2008; Van Geit *et al.*, 2016b). These established optimization approaches use a multi-objective feature-based framework; however, there are several key differences. Critically, in this work we fit reduced models instead of conductance-based models, and this work seeks to formalize the quality of fits using Z-score, χ^2 distributions, allowing us to compare the overall quality of data driven fits within and between model types. This allows us to answer questions such as: "Does one model type lead to better fits?" and "Are some *in vivo* neuron measurements outside of the reduced model parameter's intended range?" Additionally, in this body of research we fit models using a novel range of electrical properties. One such property is the firing rate versus current (FI) slope, which is chosen to guide optimization because it produces models that are believed to be pre-tuned for use in networks. Models that fit the same FI-slope have the ability to fire at prescribed rates for a given amount of excitatory current. It is important that network neurons are calibrated in this respect, such that the behavior of the modelled network has balanced firing rates and realistic interspike intervals.

Chapter 2

METHODS

In addition to the deployment of existing methods to achieve my research goals, this dissertation contains a number of innovations which are best described as methodological. I include some of these innovations here, especially those of an extremely technical nature. Some other methodological innovations, especially those of more interest to the computational neuroscience community, are reported later in Results.

2.1 Approach to Optimization Using NeuronUnit

Model optimization follows the following basic approach:

1. Identify a model class whose parameters are to be optimized, e.g. the Izhikevich model.
2. Identify experimental data associated with a neuron that will be used to guide optimization.
3. Identify a suite of tests that can use the experimental data to guide the optimizer.
4. Execute optimization of that model class against that suite of tests to return an optimized model.

Within these steps are a number of smaller decisions, including where the experimental data will be obtained and what kind of simulator will be used to run the model. Using NeuronUnit, the steps above take the follow pseudo-code form:

```
# Import code from NeuronUnit
from sciunit import TestSuite
from neuronunit.models import MyModelClass
from neuronunit.tests import MyTestClass1, MyTestClass2, MyTestClass3
from neuronunit.data import get_data_from_database_x

# Get data about some neuron
neuron_type = "CA1 Pyramidal Cell"
neural_data = get_data_from_database_x(neuron_type)
test1 = MyTestClass1(neural_data[1]) # Test based on data feature 1
test2 = MyTestClass2(neural_data[2]) # Test based on data feature 2
test_suite = TestSuite([test1, test2])

# Optimize a model against this test suite
optimized_model = test_suite.optimize(MyModelClass)
```

This approach was used to obtain the majority of results in this work; however, some other important results were obtained using a second equally import code pattern, which is stated in the Appendix (Section B).

2.2 Pre-Existing Model Class Implementations

First I accessed an implementation of the Izhikevich model that was translated from jNEUROML into a NEURON simulator implementation. However, this implementation fragmented the family of Izhikevich models into chattering and non-chattering subtypes. The model implementation also seemed to have an internal conflict between two capacitive terms and could not reproduce all of the original publication figures. Although the NEURON simulator is designed to be fast, there is a cost associated with reloading the NEURON environment many times in fast succession, and implementation odel execution times were not brief enough to be useful for optimization.

2.3 Other Model Class Implementations

For various reasons described in detail below, existing implementations of some models were not adequate for this research. These reasons included speed, generality, and consistency.

2.3.1 Model Execution: The Need for Speed

Because optimization may involve an extremely large number of independent simulations of the same class of model, each varying only in model parameter values, it is critical that both time costs–both the overhead for model instantiation and the duration of simulation itself–be as low as possible. Existing modeling tools contain overhead associated with model initialization and shuttling results in memory. These costs are trivial for single simulations, but begin to add up in optimization runs of thousands or even millions of simulations. Even the marginal cost of simulation–expressed as seconds on a wall clock per seconds of model output simulated–is often slower than expected using existing tools, due to some of these tools being written to accommodate more complex, biophysical models, rather than engineered for raw speed. During optimization, many parameter sets are explored, and the speed of simulation can be determined by the parameter values. For example, those parameter sets which produce many spikes in a given simulation run more slowly, because $\frac{dV_M}{dt}$ changes rapidly throughout the simulation, necessitating reductions in simulation step size to avoid numerical instability.

2.3.2 Model Design: Lack of Generality

Significant time was spent in the early years of this project shoe-horning pre-existing tools into the desired optimization framework, with limited success. These tools included, among others, model designers and neural simulators such as PyNN, Brian2, NEURON, and jNeuroML. However, several unexpected road blocks were encountered on the way.

2.3.2.1 NEURON

The NEURON simulator is a very mature and respected neuron and neural network modelling framework (Carnevale and Hines, 2006). This simulator specializes in multi-compartment conductance based models of neurons and neuronal networks, but the simulator has also been made to accommodate many reduced model implementations. The NEURON implementation of the Izhikevich model is fractured in that there are different NEURON NMODL implementations of the equations for different parameter regimes.

When running only a single model simulation this is not much of a problem. However, switching between Izhikevich model regimes during optimization (as would occur when a parameter value crossed a regime boundary) is a non-trivial exercise.

Even if successful, any multi-language source code successfully implementing this would be complicated, unreadable, and lack generality. Specifically, NEURON requires NMODL files to be compiled for each different regime, and it may be difficult to know in advance which regimes the optimizer is likely to sample from. Additionally, the promise of fast performance due to the C-based NEURON library is not actualized with this model. Because NEURON is well-understood within the OSB and NeuroML community, I used it only to produce reference simulations to verify that the output of my model implementations were in fact accurate according to community standards.

2.3.2.2 PyNN

PyNN provides the convenience of working in Python, and with a convenient procedural interface for model design and execution (Davison *et al.*, 2009). However, its implementations of most reduced models (e.g. Izhikevich) are simply “wrapped" versions of NEURON models; consequently PyNN has the same disadvantages as NEURON. PyNN is also designed with network simulations in mind, which means its designers have chosen performance trade-offs that favor network simulations over single neuron simulations. For example, a data-type called the “lazy-array" is the most elemental container for neuron models in PyNN, but it is meant to store populations of neurons as opposed to single neurons; as such the lazy-array adds overhead to accessing single model results.

Additionally, the NEURON implementations that underlie the PyNN-provided AdEx and Izhikevich model classes suffer from some fidelity issues under certain regimes and parameter sets, compromising optimization quality (see Davison (2020a,b) for details).

2.3.2.3 Brian2

Brian2 (Stimberg *et al.*, 2019a) is, in principle, an excellent simulator for working with reduced neuron models, as it allows for differential equations to be expressed in an intuitive form, while also keeping track of dimensions and units. However, it may not be mature enough for complex applications, as it produced errors in optimization contexts that did not occur during routine simulation of single-parameter sets.

Even when these errors did not occur, (e.g. using the Brian2 AdExp model), certain optimization steps (such as identifying the rheobase current for a given set of model parameters) took 2-3 times more simulation time then a reference approach (described in the next section). This slowness was not caused by the simulation mechanics themselves (Brian2 is relatively fast and efficient, as described in Stimberg *et al.* (2019a).) Instead, these delays are caused by the way the model is internally defined, specifically using a so-called “neurodynamics" layer (Gerstner *et al.*, 2014).

While it is very likely that this implementation is useful and correct in many contexts (Gerstner is an author of one of the original AdEx model publications (Brette and Gerstner, 2005), and the Brian2 implementation is derived directly from his work), it is problematic for the feature extraction step required in optimization. Specifically, these implementations do not formally contain any notion of “overshooting" spikes, since when the spike threshold is reached, the membrane potential is simply set to some reset value; the presence and timing of a spike is recorded only when a separate process is explicitly set to watch for such an event. This is not technically wrong, but it violates a key assumption in the *NeuronUnit* feature extraction protocol (the existence of an action potential waveform to extract), and the extra layer for detection of spiking adds computational overhead. Imputing a spike-like waveform near threshold can help solve the problem, but then optimization results and performance becomes contingent on the design of this waveform imputation, and not on the model itself.

Lastly, over the course of evaluating the Brian neural dynamics model (Gerstner *et al.*, 2014), I encountered some problems specific to genetic algorithm optimization. This context is not identical to simply running a series of simulations in series, because optimization operates in parallel, must fit into computer memory, and thus requires that simulation objects be created, simulated, and then destroyed rapidly and en masse. Since Brian2 was designed for stability, is was not designed to make model disposal computationally efficient (the problem of clearing objects from memory efficiently is, from a computer science perspective, trickier than it might initially sound). Therefore, performance of Brian2 suffered when I re-purposed its code to work in an optimization context. Brian2 does support its own internal scheme for model fitting (Stimberg *et al.*, 2019b), however this scheme was only published late in this thesis work, and it is unknown what technical tricks they employed to enforce model garbage collection. Additionally this scheme is highly divergent from the multi-objective DEAP framework described in Section 2.6, so it is not readily interoperable with the model fitting workflow described here.

2.3.2.4 My Approach

In summary, despite several choices for existing, free, open-source software (FOSS) reduced model implementations, these implementations were not useful, or significant intervention would have been required to apply them within an optimization framework. To overcome this and accelerate optimization, I built faster “direct" implementations of two neuronal models (the Izhikevich model and the AdEx model). One of the these was inspired from the existing MATLAB forward Euler implementation of the Izhikevich model, while the other was adapted from an existing Python implementation of the AdEx model using vectorized code. While neither of these was especially fast, they provided the basic recipe upon which a faster Python implementation could be built. Do note that the purpose of these new implementations was not model exploration, analysis, or sharing; existing tools are adequate for these purposes. The purpose of the new implementations was simply to make large optimization runs computationally tractable.

Although typically much faster than R, Python does not have a reputation for speed; implementation details have a large impact on performance. Therefore, I used a tool called Numba (Lam *et al.*, 2015) that enables Just-In-Time compilation (JIT) of Python code, making it comparable in speed to compiled C code. This tool cannot be applied to any arbitrary Python code, so functions to which it is applied must be designed with only a fairly plain subset of the usual syntax and library of Python. In other words, it cannot be used to simply speed up any pre-existing Python code. I crafted the two model types above to be JIT-compliant, with the result that both became significantly faster than analogous models using NEURON or Brian2 simulators. Importantly, simulation outputs retained a binary near-match in all cases, confirming that nothing was lost in the course of gaining this performance improvement. I used these new implementations extensively throughout the project, and they are available to others at Jarvis (2020b). The code that implements them is fairly easy to understand, share, and execute, and I hope they may be useful to others who have similar performance needs, either for optimization contexts or in large network models on generic commodity computer hardware where small performance gains are worth chasing.

Some models were executed in their native implementations using the NEURON simulator with an adaptive time step. After each model run, the variable time step vectors were resampled into fixed time step vectors using interpolation. I accelerated this inherently slow process by applying the JIT framework to prior code contributed by a colleague (Birgiolas, 2019).

Below, I profile my implementations and compare them to the existing FOSS implementations. My implementations led to faster per-simulation evaluations of simulations involving somatic current injection. Furthermore, my implementation exhibited over-shooting spikes (spikes crossing 0 mV, as occurs in real neurons), making them more compatible with NeuronUnit feature extraction.

2.3.3 Profiling the Models

Obtaining the rheobase of a model for a single parameter set requires simulating it many times at different values of somatic current injection until the minimum action-potential inducing current is obtained (to within some tolerance; here I used 0.1 pA, near the standard deviation of thermal noise). This takes 10-15 simulations, on average.

```
My AdExp implementation:
Single model simulation: 0.00126 s
Rheobase computation: 0.183 s


My Izhikevich implementation:
Single model simulation: 0.002 s
Rheobase computation: 0.462 s
```

2.3.3.1 Comparison of Speed and Accuracy Versus Brian2

In-order to implement NeuronUnit-compatible Brian2 AdEx model simulations, I imputed spike waveforms (at recorded spike locations) immediately following each simulation. The simulation time of this model is determined by multiple factors, as discussed elsewhere. Execution time is also not uniform across model parameterizations; in particular, parameter sets exhibiting more spikes take longer to solve numerically. I compared this with the same parameter sets in my implementation, with the results shown below:

A large fraction of the time spent simulating models under a single set of parameters is spent obtaining the rheobase current, upon which several subsequent tests and extracted features depend.

The JIT implementation of the AdExp model was approximately 1000$\times$ faster than the Brian2 model. Another benefit of the JIT implementation was that it did not require imputation of action potential waveforms (as was required for the Brian2 implementation); without additional work, the JIT implementation produced much more realistic-looking action potential waveforms under most model parameterizations. To the extent that action potential shape is an optimization target, this is a decisive advantage for the JIT implementation.

2.3.3.2 Comparison of Speed and Accuracy vs NEURON

I also compared the performance of my implementation of the Izhikevich model to the one generated by NEURON from the OpenSourceBrain Izhikevich model NeuroML2

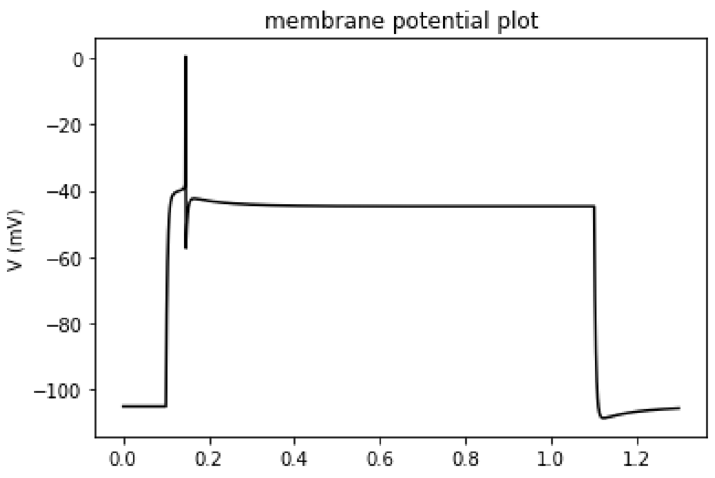


Figure 3: **Brian2 Simulation of the AdEx Model.** Simulated membrane potential trace from the AdEx model at rheobase using the Brian2 simulator. The action potential waveform has been interpolated at the time when the simulator reported a spike. The horizontal axis shows time in seconds.

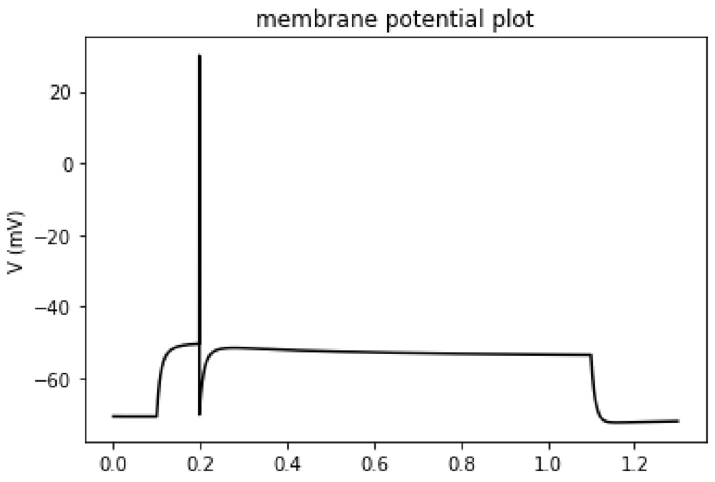


Figure 4: **JIT Simulation of the AdEx Model.** Simulated membrane potential trace from the AdEx model at rheobase using my JIT implementation. In contrast to Fig. 3, the dynamics of the action potential arise naturally from the integrated equations and do not require interpolation.

files. This NEURON implementation has several drawbacks, including: 1) It depends on an external file which must be recompiled each time this project is recreated; 2) The build environment of NEURON is non-trivial; 3) The model implementation code is less generalizable than than the published Izhikevich model itself. For example, the standard NEURON-NeuroML2 code only covers the Regular-Spiking flavors of this model, and does not support the full range of model parameterizations; 4) Name space conflicts between built-in NEURON parameters and Izhikevich model parameters.

The NEURON implementation of the Izhikevich model took 78 seconds to identify the rheobase current. In contrast, my implementation identified the rheobase in only $\tilde{0}.5s$. This represents a very dramatic speed up, which can largely be attributed to overhead associated with initializing successive simulations. Furthermore, my implementation generalizes to all possible parameter values for the Izhikevich model parameters, thus allowing for all of the many diverse spiking behaviors exhibited in the original publication by Izhikevich (2003).

2.3.3.3 The GLIF Model: a Limited Model

Although GLIF models are intentionally limited in behavior to below threshold firing dynamics, these models are still relevant to the neuronal modelling community. I developed a Generalized Leaky Integrate-and-Fire (GLIF) model by manipulating some pre-existing code until it was interoperable with the NeuronUnit framework. Because GLIF models do not include spike waveforms (like the AdEx implementations discussed above), imputation of these waveforms is required for broad spike shape optimization. GLIF models are not particularly fast, nor have they historically been good at predicting spike timing in neurons Teeter *et al.* (2018). However, GLIF models are widely-used within the Allen Institute, with that organization providing cell-specific GLIF models for each neuron that they record. So I included them here for completeness and for comparison to previous work.

2.4 Parallel Rheobase Determination

In the preceding section, I discussed determination of the rheobase of a model as one of the most computationally expensive steps in evaluating a given set of model parameters.

2.4.1 Why is the Rheobase Important?

The rheobase is defined as the minimum current required to elicit at least one action potential. In slice physiology experiments, this usually means a square pulse of somatically-injected current lasting for a fixed amount of time, for example $500ms$. The rheobase not only characterizes the excitability of a cell, but it also serves as a landmark or anchor for computing many other features of a cell's suprathreshold behavior. For example, once the rheobase is known, one can compute a so-called "FI curve" – the number or frequency of action potentials in response to a given amount of injected current – at fixed multiples of the rheobase, providing a compact summary of excitability. Both the Allen Institute and the Blue Brain Project use such rheobase-linked excitability measures. The rheobase current can also be used to compute features of spike waveforms. These features may vary with the amount of injected current, because the rising phase of an action potential may include both sodium current and patch pipette current. By restricting the analysis of single action potential waveforms to those evoked using the rheobase current, this confound is minimized because at rheobase the patch pipette current is roughly offset by outward currents (were the pipette current any less, the outward currents would have prevented a spike, by the definition of rheobase). Consequently, action potential waveform features like threshold, width, and height are often performed at the rheobase current.

2.4.2 How is the Rheobase Determined?

Determining the rheobase involves repeated application of a more general algorithm that runs one simulation to determining the number of action potentials evoked by a particular magnitude and duration of somatic current injection. Because the rheobase value partitions suprathreshold stimulus amplitudes from subthreshold ones, its determination can be accelerated by treating its determination as a search tree problem. In a search tree, the search space is adaptively narrowed between two endpoints until a target is identified. For the rheobase, this means asking (1) “What is maximum current injected so far that resulted in zero spikes?" and (2) “What is the minimum current so far that resulted in one or more spikes?" and then running a simulation at some current amplitude in between those two values (e.g. halfway between in the case of a binary search tree).

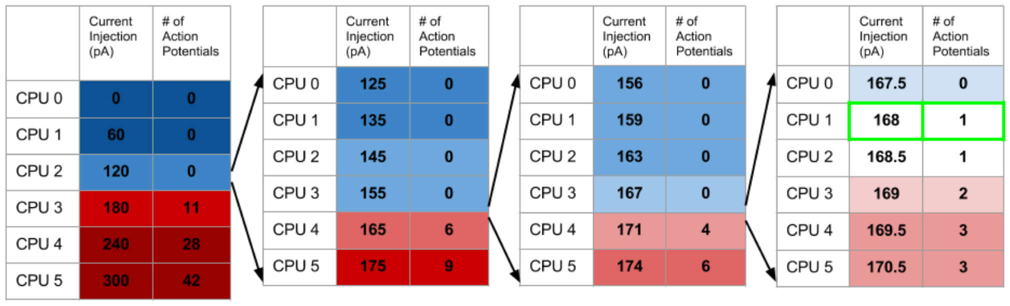
2.4.2.1 Serial Rheobase Determination

The procedure above can be run in serial (i.e one simulation after another) until the rheobase is narrowed down to an acceptably narrow range, e.g. +/- 1 pA. The initial search begins with no knowledge of any minimum suprathreshold or maximum subthreshold current amplitudes, so I use the starting range 300 pA to -100 pA, respectively. A binary search is applied within this space, with additional code to handle edge cases outside this range. Ignoring those edge cases, such a binary search requires $log_2(I/i)$ simulations, where I is the range being searched (here, 400 pA) and i is the resolution of the solution (here, 1 pA). Thus, ~ 9 simulations are required to obtain the rheobase using this binary search strategy.

2.4.2.2 Parallel Rheobase Determination

This process can be accelerated by running simulations in parallel. Each step of the search requires knowledge of the outcomes of the previous simulations, so there can be no parallelism across steps (other than brute force parallel search of the entire range of currents, which is extremely inefficient). However, it is possible to parallelize within each step, so I exploited this approach. A binary search partitions the search space in two by simulating a current injection at $(sub + super)/2$ pA, where sub is the previous maximum subthreshold current, and $super$ is the previous minimum suprathreshold current. The value $(sub+super)/2$ pA either does or does not produce a spike, leading to its value being used to update either $super$ or sub, respectively. The search space is thusly cut in half, so this simulation effectively generates one additional bit of information about the amplitude of the rheobase. This repeats $\sim$ 9 times until all 9 bits of uncertainty (from the initial 400 pA) range have been eliminated. I accelerated this with parallelism by applying an N-ary search (rather than a binary search), where N is the number of parallel processes, and N+1 the number of regions of current amplitude to search. This is described in Figure 5.

Consider the initial 400 pA range. With only one thread, this range is bisected and a simulation run at it midpoint $(-100pA + 300pA)/2 = 100pA$. With seven threads, this range can be octo-sected, with concurrent simulations run at each of seven values, i.e. $-50, 0, 50, ..., 300, 350$ pA. The highest of these seven values that produces a spike is assigned to *sub* and the lowest that does not to *super*, resulting in the search space now being restricted to only one of these 50 pA wide regions. This is 1/8 as a wide as the initial space, so 3 bits of information about the rheobase have been obtained. This parallel process is then repeated serially (i.e. octo-section of the new 50 pA region, octo-section of the ensuing 6.25 pA region, etc.), until the rheobase has been determined. Because 3 bits of information are obtained in every step instead of 1 bit, the search is 3x faster. In general, the parallel N-ary approach is $log_2(N+1)$ faster than the plain serial binary search approach, with speedup gain therefore growing logarithmically in the number of concurrent threads (usually, proportional to the number of CPU cores) being used. As architectures with hundreds of cores are now common, speedups of 7-10 fold are achievable.



	Current Injection (pA)	# of Action Potentials
CPU 0	0	0
CPU 1	60	0
CPU 2	120	0
CPU 3	180	11
CPU 4	240	28
CPU 5	300	42

	Current Injection (pA)	# of Action Potentials
CPU 0	125	0
CPU 1	135	0
CPU 2	145	0
CPU 3	155	0
CPU 4	165	6
CPU 5	175	9

	Current Injection (pA)	# of Action Potentials
CPU 0	156	0
CPU 1	159	0
CPU 2	163	0
CPU 3	167	0
CPU 4	171	4
CPU 5	174	6

	Current Injection (pA)	# of Action Potentials
CPU 0	167.5	0
CPU 1	168	1
CPU 2	168.5	1
CPU 3	169	2
CPU 4	169.5	3
CPU 5	170.5	3

Figure 5: **Parallel Rheobase Algorithm.** I developed a generic algorithm which finds the minimal current injection value needed for one spike (rheobase). Rather than a brute force or binary search, I implemented an N-ary search that uses multiple parallel simulations to rapidly identify the rheobase. This leads to significant speed-ups for testing and optimization in all models.

2.4.3 Practical Application

I benchmarked this approach using simulations of multi-compartment neuron models. The parallel rheobase determination algorithm resulted in a significant speed up relative to the serial algorithm. To amplify the effect, I also considered a scenario where one wants to learn the value of the rheobase with much more precision (down to small fractions of a pA), for example in studies of dynamics in the neighborhood of a bifurcation where all other state variables can be considered nearly unchanged as one crosses incrementally between sub- and suprathreshold scenarios. To achieve such precision, i.e. for a small value of i such as $i = 0.0001 * pA$, the number of simulations $log_2(I/i)$ may be ~ 20. Since additional model features may require only a few additional simulations to extract, it is clear that in this scenario the rheobase completely dominates the bulk of the total simulation budget.

In this scenario, using only 16 threads (with a theoretical speedup of $log_2(16+1) \sim 4.09$), I achieved the following results:

```
NEURON simulation of multicompartmental model
Serial Rheobase determination: 18.7 s
Parallel Rheobase determination: 4.8 s
Speed up = 3.9x


Brian2 simulation of AdEx model
Serial Rheobase determination: 0.791 s
Parallel Rheobase determination: 0.259 s
Speed up = 3.0x
```

The total speedup approached but fell a bit short of the theoretical speedup due to overhead in the parallel search algorithm itself. As the complexity of each simulation increases, and as the number of CPU cores brought to bear increases, this overhead should become a vanishingly small fraction of the total rheobase determination time.

2.4.4 Generalization to Target Spike Counts

This approach for determining rheobase was also generalized into a more fundamental algorithm for determining the amplitude of current required to generate any target number of action potentials. In other words, it can be used to invert a model's "F-I" function, finding the "I" associated with a given "F". For more information, see the test at (https://github.com/russelljjarvis/neuronunit/blob/master/neuronunit).

2.5 Electrophysiological Measurement Distributions From the Experimental Literature

Organized, publicly available electrophysiological measurements from single, biological neurons can form an optimization target. Together, they can be used to parameterize a suite of tests against which a model is optimized. Optimization aims to find model parameters so that electrophysiological measurements based on model simulations are as similar as possible to those observed in neurons.

2.5.1 NeuroElectro

One general source of such experimental measurements is The NeuroElectro Project (Tripathy *et al.*, 2014), which contains experimental values for 47 distinct electrophysiological measurements across 235 different neuron types. As with most of the data discussed here, most (but not all) of these measurements were obtained from slice physiology experiments in rodents. These measurements were programatically extracted from peer-reviewed journal articles over a ~ 20 year period from $\sim 1990 - 2012$, and are made easy to access by an application programming interface (API) that NeuronUnit provides bindings to. Importantly, the measured values–even for a single neuron type–reflect experiments done in many labs using (in some cases) variable methods. Therefore, the mean of these values (e.g. the mean input resistance across reported Purkinje cells) averages over heterogeneity across cells within a slice, slices within an animal, animals within a lab, and labs within the field. The sample size for one measure (e.g. input resistance) may be larger than for another (e.g. resting potential) meaning that these measures may reflect different subsets of experiments. With those caveats in mind, NeuroElectro remains the most direct way to get a large number of optimization-constraining data values for most neuron types.

In order to verify that the data from NeuroElectro was plausible and was being captured correctly for the purposes of the work in this thesis, I used the API along with a batch visualization pipeline to visualize the distributions of electrophysiological measurements and inspect them for a) quality control and b) evidence of multimodality. Multimodality, meaning multiple peaks in the histogram of a single measurement type for a single cell, could be evidence of a physiological heterogeneity not easily explained by random measurement error. Two peaks in the histogram, for example, could result from two distinct subclasses of a single nominal neuron type, each with its own (narrower) distribution of the same measurement. In some instances, the mean and standard deviation alone described the measurement distributions well, as would be expected for random, normally-distributed measurements of a single cell type under reasonably consistent conditions. These values were then “approved" for use in model-fitting. In other cases, these conditions were not met, as exemplified in the figures below.

For the majority of cell types and electrophysiological features, the distributions obtained from NeuroElectro were well-described by a normal (or log-normal) distribution. However, I manually identified and labeled those cases where the data were not well-behaved, as these cases are likely to produce optimized models that do not reflect anything of biological relevance.

Methods for verifying that a distribution is unimodal exist (Maechler, 2013); however, rather than entrust this job to top-down automation, I decided to apply my own human knowledge of statistics in order to assess each distribution individually.

I estimated that across all NeuroElectro data sampled here, about 2/3 of distributions are well represented by a unimodal and normal distributions (e.g. Figure 6). In the remaining 1/3, where this did not hold, I observed a small but still significant number of odd cases: highly skewed distributions (Figure 7), bimodal distributions (Figure 8), uniform-like distributions, and distributions with insufficient samples to make any judgement.

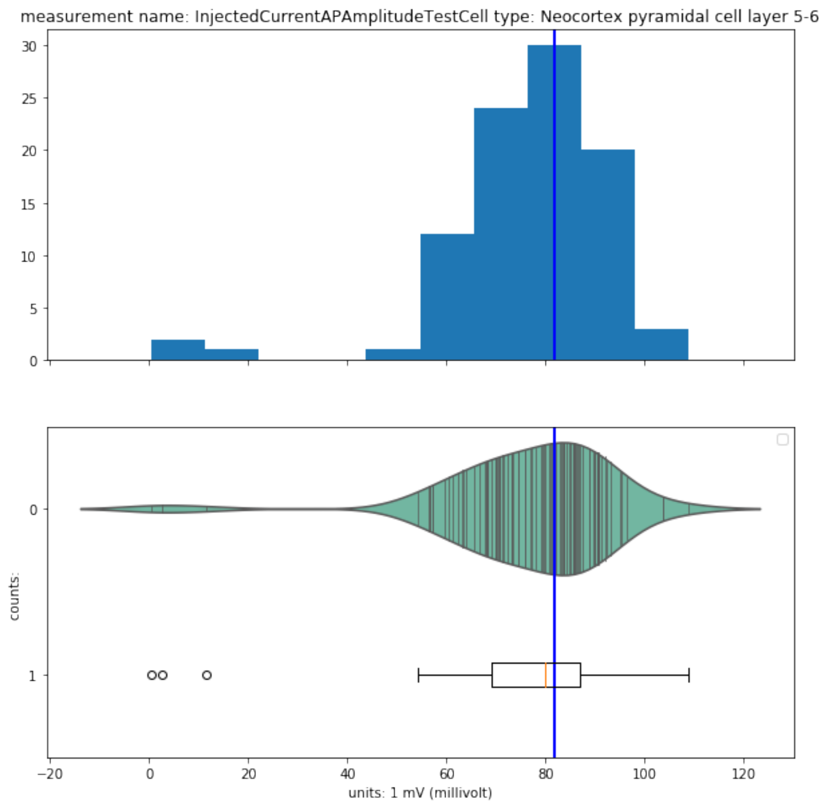


Figure 6: **AP Threshold Data Distribution for Layer 5 Pyramidal Cell.** The majority of NeuroElectro data sets followed a normal distribution, where the mean is surrounded by a very high density of samples, which slowly thin out with increasing distance from the mean. The distribution is approximately symmetrical. Top Panel: A histogram of AP Threshold measurements from Layer 5 Pyramidal Cells; Bottom Panel: A violin plot and a box plot summarizing the same distribution. In this plot, the mean and mode are close together (a necessary condition for near-normality). Analagous plots were generated and inspected for all electrophysiological features computed here.

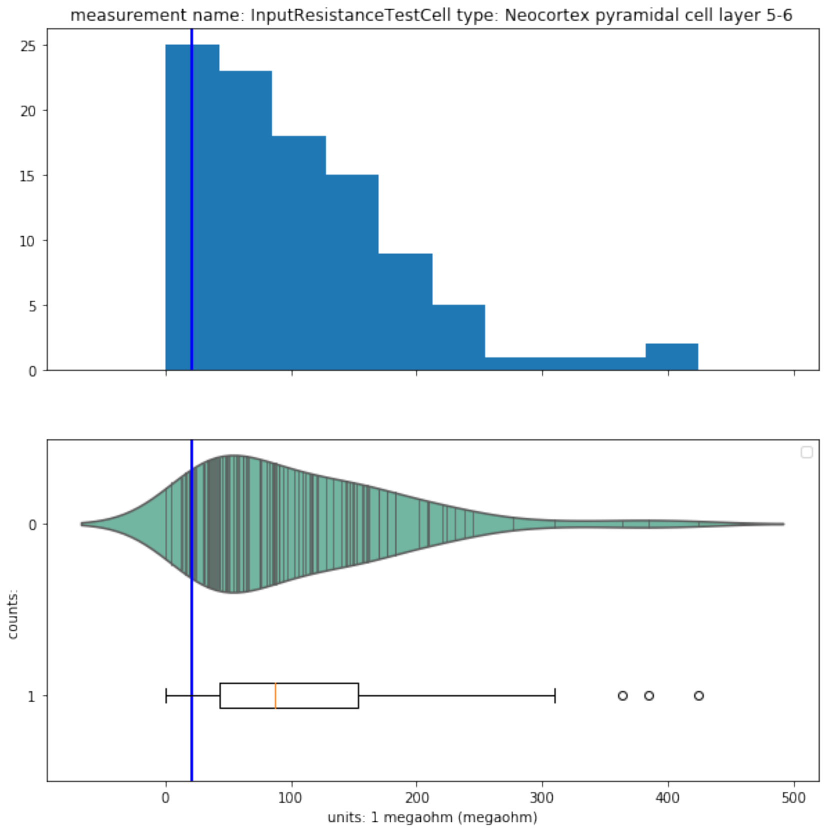


Figure 7: **Input Resistance Data Distribution for Layer 5 Pyramidal Cell.** Similar to Figure 6, except for Input Resistance. Unlike in that figure, the distribution shows a strong skew towards higher values. The mean and the mode are no longer well-aligned, and therefore the mean is no longer representative of the most typical value of this feature for this neuron type.

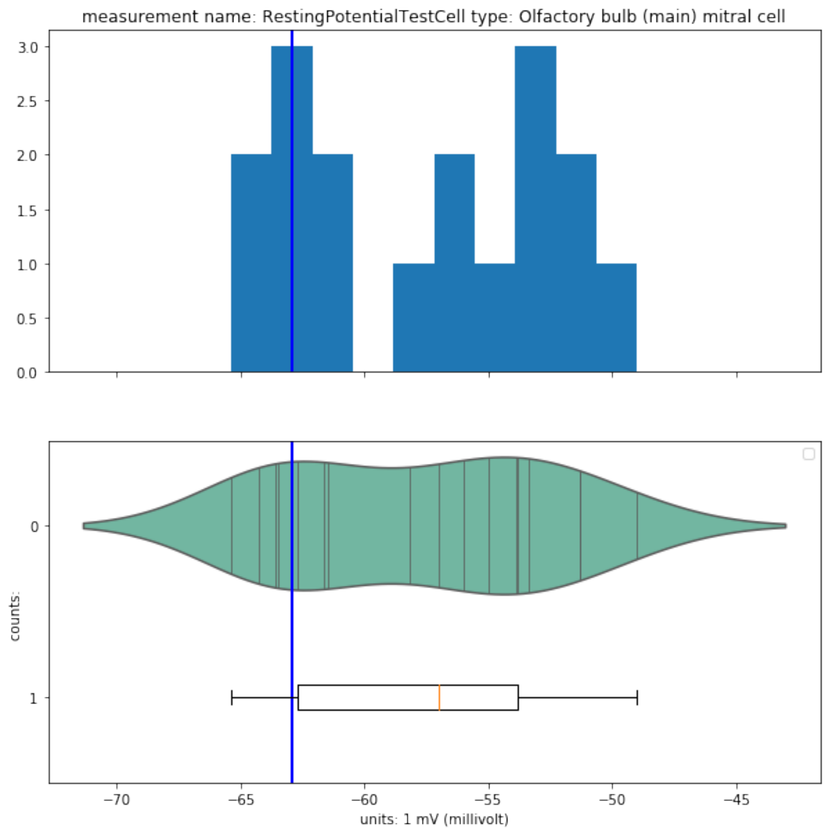


Figure 8: **Resting Potential Data Distribution for the Olfactory Bulb Mitral Cell.** Similar to the previous figures, but for a different cell type and electrophysiological feature. In this case, the distribution is clearly bimodal, with each mode containing a similar density of the data. Now both the mean and the median (small red line in box plot) are especially unrepresentative, lying in a region of low probabilty density.

2.5.2 EFEL and The Allen Institute Cell Types Database

The Electrophysiology Feature Extraction Library (EFEL) (Van Geit, 2015b) was developed as part of the Blue Brain Project. Although EFEL computes common spike train statistics related to spike timing, approximately 2/3rds of features extracted by EFEL pertain to spike shape, where some of these features are shown below. The Allen SDK comes with a very comprehensive Python-based feature extraction suite. Like

EFEL, the Allen suite well-represents a large number of spike shape measurements as well as spike train statistics. Unfortunately, the Allen SDK feature extractor is significantly slower than EFEL, as EFEL was implemented using the very fast language $C++$. The performance cost may not be felt when dispatching single runs, but slow performance is a significant impediment to optimization. In optimisation, feature extraction is directly coupled to chromosome fitness calculations, and it is executed very often across the evolution of the genetic algorithm. Additionally by default, the Allen SDK feature extractor assumes that the user will apply very high sample frequency and noisy traces encoded in the NeuronData Without Borders (Teeters *et al.*, 2015) standardized format. These traces require filtering before computing the Allen features, where significant intervention is required to turn off filtering. Inappropriately applying filtering to model traces causes problems, because the the lower sampling frequency intrinsic to simulated model traces is not predicted by the digital filter. Overall, the EFEL was fast enough to be useful, and its default settings were appropriate to my use case (Garcia *et al.*, 2014b).

Data available through NeuroElectro cover a large number of cell types; however, recording conditions and measurement algorithms are heterogeneous. It is unclear whether the distribution of measurements across such an ensemble is actually a good summary of any one individual neuron. In order to ensure that reduced models could be optimized against data recorded exclusively from single neurons, I also used data from the Allen Institute Cell Types Database (Institute, 2015), a project of the Allen Institute for Brain Science. This Cell Types database consists of summary physiological, morphological, and histological data for thousands of individual neurons (across a few dozen subtypes) from mouse visual cortex, obtained using patch clamp recordings in slices. Each experiment is done using exactly the same methods and with the same sequence of stimuli (Institute, 2015), ensuring not only that models generated using this data are directly comparable, but that each such model is reflective of an individual neuron.

The Cell Types Database provides some limited pre-computed measures of action potential waveform characteristics. However, the data are not organized in a way that makes it is useful for the types of optimization and data analysis performed here. Specifically, I require features that are computed on cell responses to current injection values that are fixed multiples of rheobase. Additionally, the pre-computed features are thin relative to those that used for the optimizations described in the Results section. Because raw data are available through the Cell Types API, I re-computed all necessary features from this raw data, according to the consistent standards reflected in the NeuronUnit code.

In contrast to NeuroElectro, the Cell Types database also has a great deal more information relevant to the above threshold dynamics of neurons, such as the number and pattern of action potentials they discharge in response to somatically-injected currents much larger than rheobase, or in response to non-square injected currents. In order to exploit these, I developed several additional NeuronUnit tests using EFEL (describe later in sections 2.6.4.2), such as: “time to first spike test", “mean AP amplitude test", “time to last spike test" and “adaption index". In principal any feature measured in the Cell Types Data could be upgraded to a NeuronUnit test, and I created a code-generation template to accelerate this task. Effectively, code generation meant, that any EFEL feature, could be turned into a NeuronUnit test. In principle Allen tests can be generated from templates in much the same way. The final set of operational tests were EFEL Van Geit (2015b) tests that were adapted from descriptions of feature extraction in the literature and shown in Table 1. Features are shown in Figures 9 and 10. I also crafted additional NeuronUnit tests to supplement these including one that measures the slope of the FI curve ($FISlopeTest$) and one that measures the coefficient of variation of the ISI distribution for suprathresholds stimuli ($ISICVTest$), a measure of burstiness.

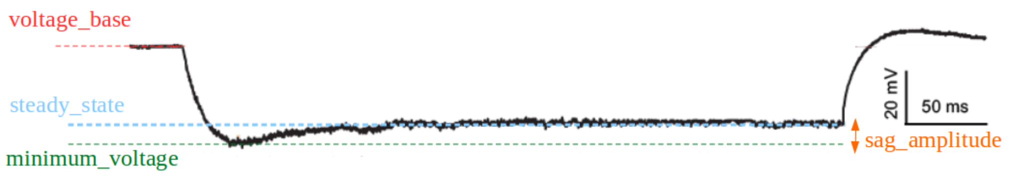


Figure 9: **Passive Membrane Properties from a Hyperpolarizing Stimulus.** Applying a negative (outward, hyperpolarizing) current stimulus minimially activates voltage-dependent ion channels, making it a good method for measuring “passive" membrane properties such as the input resistance (measured as the difference between the resting potential (red) and steady-state hyperpolarization (green). Nonetheless, some intrinsic conductances are activated, allowing for measurment of additional features such as the sag amplitude or ratio (green trough vs blue steady state). Figure from EFEL documentation (EFEL-Developers, 2018).

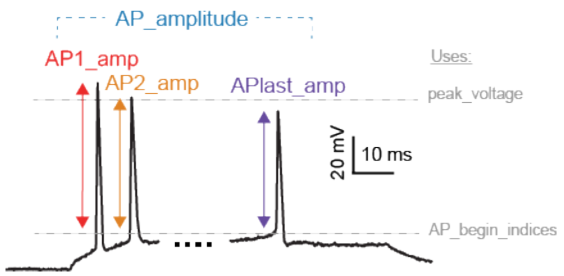


Figure 10: **Action Potential Features Measured from a Depolarizing Current Stimulus.** A positive (depolarizing, inward) current of sufficiently amplitude produces one or more action potentials. Many features can be computed from these, including their absolute amplitudes (in color), relative amplitudes, widths, thresholds, and afterhyperpolarizations (AHPs). Additional features about their number and relative timing can also be computed (not shown). Figure from EFEL documentation (Van Geit, 2015a).

These tests can be used to assess the agreement between neuron models and biological neurons on supra-threshold dynamics, largely reflected in patterns of spiking such as bursting and adaptation, but also mean spike height and mean spike width, resting membrane potential, mean trough depth, and upstroke times.

2.5.3 The Blue Brain Project Neocortical Microcircuit Portal

I also made use of an additional data source, the The Blue Brain Project Neocortical Microcircuit Portal, similar in some ways to the Allen Institute Cell Types Database but reflecting measurements taken from mouse somatosensory cortex (again in patch-clamp recordings from slices). From this dataset I exclusively used a collection of experiments from animal $B95$, which for reasons unknown yielded a tremendous amount of data (Ramaswamy *et al.*, 2015). Conceptually, this dataset did not add anything new, but it did allow for high-quality optimized models to be produced from another brain region (somatosensory, rather than visual cortex). These data are also linked to–and constrain–the on-going Human Brain Project effort to simulate biophysically-detailed multi-compartmental models of the same neurons (and whole neural circuits). This means that the reduced models produced here can be compared directly to those more detailed models, or that the general NeuronUnit-driven genetic optimization framework developed here could be used to optimize detailed models which should, in principle, be similar to those produced through the larger Human Brain Project effort. Indeed, the Human Brain Project is already a user of the SciUnit framework developed in my lab, on which NeuronUnit is based.

The tests which lead to the best fits in the above threshold experiments were the tests made from application of EFEL features to Allen data sources (Figure 9). The measurement type and the test type did not change between Allen Cell Types and

Test Name	**Test Description**
adaptation-index	Measures spiking fatigue in response to constant current
adaptation-index2	The same as Adaption index1, except it is used as an alternative when spikes below $0mV$ occur.
time-to-first-spike	amount of time elapsed until first spike
mean-AP-amplitude	The average spike height in a spike train
spike-half-width	The width of a spike is obtained at point when spike height is half its total amplitude
AHP-depth	The after hyperpolarisation depth
minimum-voltage	The minimum voltage
peak-voltage	the maximum voltage, usually a spike peak.
time-to-last-spike	The time of last spike onset
AHP-depth-abs	After Hyperpolarisation depth (absolute value).
all-ISI-values	All interspike interval times
voltage-base	minimum voltage while undergoing stimulus, often below the threshold of APs.
min-voltage-between-spikes	Needed because during high frequency firing AHPs may be skipped.
Spikecount	Just the number of spikes that occured in the provided stimulus window

Table 1: 14 key features identified in the EFEL (Van Geit, 2015b) library that I impemented and encoded into NeuronUnit tests for optimization.

Blue Brain Data. Only the reference data which informed comparison measurements changed.

Table 1 constitutes a summary of both NeuroElectro and Allen experimental data reports.

Test Name / Cell Type	CA1 Pyramidal	Purkinje	NCP Layer 5-6	Mitral Cell	623960880	623893177	471819401	482493761
RheobaseTest	189.24 pA	680.79 pA	213.85 pA	NaN	70.0 pA	190.0 pA	190.0 pA	70.0 pA
InputResistanceTest	107.08 $M\Omega$	142.06 $M\Omega$	120.67 $M\Omega$	130.08 $M\Omega$	241.0 $M\Omega$	136.0 $M\Omega$	132.0 $M\Omega$	132.0 $M\Omega$
TimeConstantTest	24.5 ms	NaN	15.73 ms	24.48 ms	23.8 ms	27.8 ms	13.8 ms	24.4 ms
CapacitanceTest	89.8 pF	620.27 pF	150.58 pF	235.75 pF	NaN	NaN	NaN	NaN
RestingPotentialTest	-65.23 mV	-61.59 mV	-68.25 mV	-58.14 mV	-65.1 mV	-77.0 mV	-77.5 mV	-71.6 mV
InjectedCurrentAPWidthTest	1.32 ms	0.41 ms	1.21 ms	1.61 ms	NaN	NaN	NaN	NaN
InjectedCurrentAPAmplitudeTest	86.36 mV	71.23 mV	80.44 mV	68.4 mV	NaN	NaN	NaN	NaN
InjectedCurrentAPThresholdTest	-47.6 mV	-46.89 mV	-42.74 mV	-38.9 mV	NaN	NaN	NaN	NaN
FISlopeTest	NaN	NaN	0.05 Hz/pA	NaN	0.18 Hz/pA	0.12 Hz/pA	0.18 Hz/pA	0.09 Hz/pA

Table 2: Data for 9 tests (features) across 8 cells. The first 4 cells are specific cell types spanning several brain regions, and the corresponding data comes from neuroelectro.org. The remaining 4 are single (cortical) cells comes from the Allen Cell Types database, and the features were directly computed using NeuronUnit tests. “NCP" indicates neocortical pyramidal.

2.5.4 Novel Data-Driven Tests for Model Optimization

At the onset of this project, NeuronUnit contained a range of model validation tests, but these were inadequate for for optimization. Most such tests were restricted to measurements of passive membrane properties or action potential waveforms. However, the rich diversity of neuronal physiology is also reflected in the rate, timing, and sensitivity of patterns of action potentials.

As noted in Section 2.5 there were two experimental data types, each of which was used to constrain models differently: raw data from which features were (re-)calculated, and pre-computed features as reported in the literature. The distinction is important here because in the former case new feature extraction routines are required. In order to developed tests based on these newly calculated features, I created an interface to the Allen Cell Types API; this allowed me to automatically create NeuronUnit tests parameterized by features extracted from the membrane potential traces available from the Allen Institute through that API. In order to create the new NeuronUnit tests, one must query the Cell Types Database, impose a new organization on the data, extract relevant features, and convert model features for use with NeuronUnit tests. A similar API was created in order to work with the Blue Brain data described in section 2.5.3, so that both of these data sources could then guide model fitting. These APIs and their use in generating NeuronUnit tests are documented in Jarvis (2020a).

At this stage the range of tests available for optimization in NeuronUnit was still incomplete; the data was adequate, but several key features that distinguish one pattern of spiking from another were still missing. Therefore, I implemented another series of tests in NeuronUnit using the EFEL features (Van Geit, 2015b) (section 2.6.4.2). I developed the ability for all of the EFEL features (Table 1) to be calculated on NeuronUnit models, as well as a new test to compute the slope of the F-I curve. I organized these tests into new "at threshold" and suprathreshold test suites, which included features such as interspike-interval (ISI) statistics, after-hyperpolarization (AHP) depths, spike frequency adaption ratios, etc.

Adaptation in spike trains is commonly occurring. When fitting models to spike trains it is best to measure adaptation directly, and to use it as one of many equally weighted objective functions. Fitting models against adaptation directly is important for taking full advantage of the Adaptive Exponential model type.

In order to optimize reduced models, it was necessary to develop "optimizer-friendly" implementations of these models. As described in section 2.3.2.4, I developed three different optimizer-friendly reduced models: the Izhikevich model (spanning seven dynamical regimes), the Adaptive Exponential Integrate and Fire (AdEx) model, and the GLIF model. Additionally I created one slower single-compartment conductance-based model and I retrofitted a pre-existing multi-compartment conductance based model to make a broader array of comparisons across model types.

2.5.5 Conversion Between Types and Model Exchange Across Threads

Multi-core and or multi-threaded optimization requires that information about model properties be shareable across various processor threads. However, some very common data types used in programming are not easily shareable between CPUs. The release of Python 3.8 solved a subset of these problems, but this did not occur until late 2019 so I implemented an alternative scheme for cross-thread sharing of models. I created a new NeuronUnit class "Data Transport Container" (DTC), which can circulate essential data about model parameters and state variables between threads. Models on one thread were coalesced into this container, shared between threads, and then reconstituted on a new thread. This added a small amount of overhead, but this computational cost was negligible when measured against the gains of using multi-threaded processing.

Beyond the multi-threading context described above the DTC class constitutes the necessary expansion of Neuronunit model class needed to make optimization work. The DTC class enables the creation chromosomes from models to evaluate the fitness of any single chromosomes, and to convert chromosomes back to models, these types of conversions are essential to visualising Genetic Algorithm jobs.

2.5.6 Web Application for Optimization

In order to both control optimization parameters and visualize optimization results, I also developed a web application that requires no programming skill to use, and allows a user to select among multiple cell-specific constraints, and multiple model types. Once the user has specified enough parameters to define an optimization job, the job can launch, and return interactive results to the user after the optimization job completes (typically in minutes). This is described in section 3.6.

2.6 Technical Details of the Optimizer

The sections above describe my innovations in model construction and simulation, as well as the experimental data brought to bear on optimization. These data are used to parameterize NeuronUnit tests, one per measurement type. For example, input resistance data for one neuron type in NeuroElectro, or one specific neuron in the Cell Types database, is passed to an *InputResistanceTest* defined in NeuronUnit. This test “asks" the model to generate a corresponding simulation, measures the input resistance in this simulation output, and then assesses model/data agreement, resulting in a score. These mechanics have been described at length previously in Omar *et al.* (2014), Gerkin *et al.* (2019), and Birgiolas (2019).

2.6.1 Generating and Using Scores

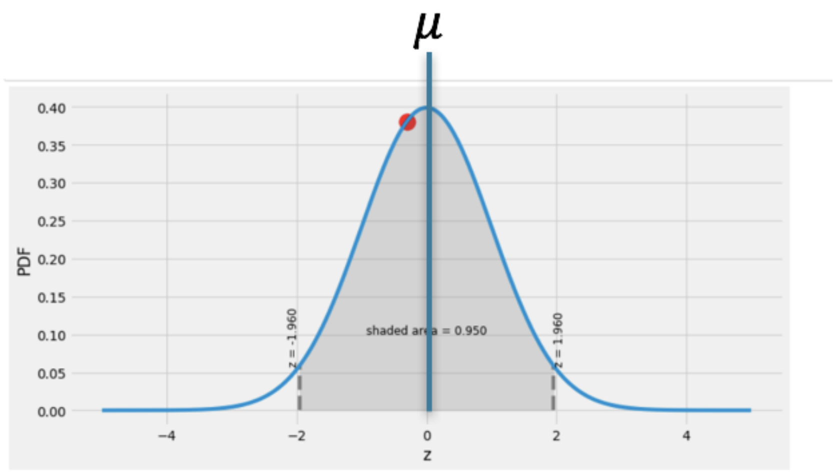


Figure 11: **Z-scores for NeuronUnit Tests**. As discussed in the section 1.4, error functions were evaluated with the assistance of the *NeuronUnit* library. This involves obtaining an experimental distribution over electrophysiology feature measurements for a cell type, measuring corresponding model output features, and then locating those features in that experimental distribution. Scores that are closer to the experimental mean are identified as low error. The Z-score encodes this information; a Z-score of 0 is the lowest possible error.

One way to ask whether the simulated feature is a good match to the biological data distribution is to use a Z-score. The Z-score is defined as:

$$Z - Score = \frac{s - b_{\mu}}{b_{\sigma}} \tag{2.1}$$

where s is the value of the feature in the model simulation, and b_{μ} and b_{σ} are the mean and standard deviation of that feature in the biological data distribution. The

Z-score does not specifically assume that the biological data are normally distributed, although this generates the most natural interpretations. In cases when the biological data from one neuron type comes from a single experiment on a single neuron (as with some data from the Allen Cell Types database or the Blue Brain Project), there is no mean or standard deviation, so I compute a *RatioScore*:

$$Ratio - Score = \frac{s}{b} \tag{2.2}$$

where b is the observed biological feature value. Both types of scores were then normalized to produce an error signal in the range $(0, \inf)$ for use by the optimizer. For example, suppose a feature(e.g. the rheobase current) had value $\mu \pm \sigma = (100pA \pm 40pA)$ in the biological data, and $110pA$ in the simulated model output. Then the following steps were taken to transform it into an error signal:

1. A Z-score is computed: $\frac{110pA - 100pA}{40pA} = 0.25$
2. This is converted to the range (0, 1) using the error function: $abs(erf(Z)) = 0.27$
3. The logarithm is computed: $\log_{10}(0.27) = 0.56$

The value 0.56 above represents a larger model/data disagreement than the “best" possible value of 0 (corresponding to a Z-score of 0), but less disagreement then the “worst" possible value of inf (truncated in practice at 100) representing a Z-score of $+\infty$ or $-\infty$. The summed error signal over all n NeuronUnit tests (e.g. rheobase, input resistance, spike rate adaptation, etc.) is:

$$TotalError = \sum_{i}^{n} error_i \tag{2.3}$$

i.e. the sum of all of the errors. Again, 0 would represent perfect model/data agreement across all tests.

While the optimizer attempts to minimize the total error of the model according to the equation above, evaluation of the quality of the optimized model is evaluated using a hypothesis testing framework. Specifically, I ask whether there is sufficient evidence that the optimized model is representative of the distribution of feature values observed in the biological data. The null hypothesis can be states as “the observed features of the optimized model were drawn from the distribution of features of biological neurons". In order to generate a test statistic for hypothesis testing, I compute χ^2, defined as (in the case of Z-scores):

$$\chi^2 = \sum_{i=1}^{n}(Z_i^2) \tag{2.4}$$

This equation arises from the fact that a Z-score is simply a standardized normal variable, and a chi-squared distribution with n degrees of freedom is simply the distribution of n independent, squared normal variables. I compute a p-value by comparing the observed chi^2 statistic to the cumulative distribution function of the chi^2 distribution with degrees of freedom equal to the number of NeuronUnit tests used (which is equal to the number of Z-scores produced). If this p-value is small (e.g. < 0.05), that can serve as evidence to reject the null hypothesis, indicating that the optimized model did not “fit in" well with the observed biological data. In contrast, failure to reject the null hypothesis would suggests that the optimized model was somewhat convincing in its mimicry of a biological neuron, for the features in question.

2.6.2 Mechanics of Optimization Using NeuronUnit

Here I will describe how I generate these scores concurrently for many parameterizations of the same model and how they guide the optimization path. I created two different optimization code bases based on the DEAP Python package for genetic optimization (Fortin *et al.*, 2012), one that relied on DEAP directly, and one that relied on the BluePyOpt package produced by The Human Brain Project (Van Geit *et al.*, 2016b) (These have since been merged together), in order to achieve optimization using NeuronUnit. A few key modules are essential to both approaches. I wrote the file *optimization-management.py* to contain the logic of and methods for managing complex optimization jobs. It helps the optimizer handle both fixed and varying model parameters, contains methods for random sampling of model parameter spaces, can plot models output for visualization of this space, and assists in computing the F-I curve. I also add several methods for inter-converting between representations of the models themselves and the chromosomes that represent only parameter values.

A created a *NUFeature_standard_suite* class to convert NeuronUnit features to BluePyOpt objective functions, as outlined in simpler terms in the enumerated list above. These classes contain a complicated nesting of fault handling statements, as there are many reasons why a candidate model could return unusable simulation output (typically non-biological parameter values), resulting in values like NaN and inf; such values must be recast as poor but finite errors so that the optimizer can see a smooth error surface. There are two flavors of *NUFeature_standard_suite*, one for supra-threshold simulation experiments and another for at threshold or sub-threshold experiments, since each experiment type produces different feature requiring different feature extractors, and producing different sets of edge cases to be handled independently. For example, there are more ways for a model to fail to elicit multiple action potentials (causing all ISI-based feature extraction functions to return NaN values), than there are to fail to exhibit a hyperpolarizing response to a small outward current injection for the measurement of input resistance.

I created a *model-parameters.py* file, a collection of ordered dictionaries, that informed the optimizer which parameters should be modifiable (in the highest-dimensional cases, all of them) and what are reasonable (biologically plausible) search boundaries. This file also contains example parameter sets representing notable dynamical regimes, such as those shown in Izhikevich (2003). I also made this file and its methods inter-operable with BluePyOpt model parameter management scheme.

2.6.2.1 Optimization Parameters

Optimization requires searching for better and better solution across multiple generations of chromosomes (parameter sets), as noted in section 1.3.1. Robust optimization for the models used here required $NGEN \sim 150$ generations with a population size (number of parameter sets explored in each generation) of $\mu = 35$. In other words, it took about 150 generations of mutation, crossover, and selection to achieve convergence, and in each generation about 35 models had each of their feautures computed and scored. These parameter values achieved an acceptable balance between exploration of the parameter space and exploitation of favorable regions. In some cases, values as small as $NGEN = 10$ and $\mu = 10$ were tolerable, for example when optimizing only low-dimensional cross-sections of parameter space. In other cases, such as when the number of optimization objectives (i.e. the number of electrophysiological features being tested) was $NOBJ > 25$, values as high as $NGEN = 300$ and $\mu = 100$ were required to obtain adequate results.

2.6.2.2 Multiobjective Scoring and Selection

One potential scientific goal is to maintain a diverse set of solutions (i.e. very different parameter sets that nonetheless each produce simulations that adequately match observed experimental measurements). The optimization literature has developed many competing approaches for doing this (Deb *et al.*, 2000), but it usually involves two popular algorithms, named IBEA and NSGA2, which I investigated here. NSGA2 uses some additional ranking mechanisms, to re-weight the perceived fitness of each chromosome and influence the probability that it survives (or is bred into) the next generation. For example, it tries to minimize “crowding distance", penalizing chromosomes that aggregate in clusters, as persistent cluster formation means that the GA becomes preoccupied with more limited regions of the solution space, harming solution diversity. Another meta-constraint called “non-dominated sorting" ranks most highly each chromosomes that is not unanimously defeated by any other chromosome on any feature score. For example, though one parameter set P might produce a model which score poorly on all features except Input Resistance, if no other parameter set has a higher-scoring Input Resistance feature then P is retained. Consistent with personal communication with Van Geit *et al.* (2007), adding in crowding distance and non-dominated sorting typically harms optimizer performance, in the context of neuronal model optimization, though the reason for this is not argued conclusively, I speculate on a likely cause in the discussion of this work. A simpler “select best" algorithm (labelled IBEA) dispenses with these tricks, performs no meta-constraint scoring, and simply retains the fittest chromosomes for mutation, crossover, and selection. This simpler selection algorithm was found to work well when optimizing reduced neuronal models.

2.6.3 Comparison to Previous Approaches to Optimization

2.6.3.1 Time-dependent Mutation

Other labs have previously developed schemes to optimize neuron models, e.g. Druckmann *et al.* (2007). I retained the conceptual insights of these approaches where they were useful for the problems at hand. For example, I utilized a time-dependent mutation magnitude (η). The idea is that big mutations are more helpful in the early stage of optimization, when it is important to explore the vast hyper-volume of parameter sets and get a general picture of the error surface, and that these mutations should be smaller during the later “refinement" stage of optimization, as the best solutions are approached. Time diminishing η did improve genetic algorithm on reduced model optimization problems.

2.6.3.2 Variants on Somatic Current Injection

Nearly all neuron optimization work (including this one) relies on the responses to somatically-injected current as the domain for optimization. This is largely motivated by the existence of a common and simple experimental analogue using patch clamp (which drives experimental design for both the Allen Cell Types Database and The Blue Brain Project). But there are three different strategies for choosing the subset of these experiments that are recapitulated in optimization.

In the optimization framework I tested many different strategies for constraining models with current injections, there were only two important differences between all four strategies: one type of strategy constrains cell behavior at multiple values of current injection, and the other strategy constrains cell behavior at a single current injection value (rheobase or otherwise). Each strategy seeks to resolve a “bias variance trade-off" differently, and so knowledge of bias variance trade-off is important for understanding the dramatically different quantities of fit found.

When only one current injection value is used seemingly great fits of spike shape, and spike time can be achieved, because the reduced models are more able be over-fitted with respect to a limited range of data. When a model is constrained relative to multiple current injection values, over-fitting of the model is less possible. The optimizer produces a model that is compromised on almost all constraints, with few exceptions.

Deliberately over-fitting models was an important optimization strategy in the earlier development of the optimizer, as at the time the highest priority was to verify that the optimizer was functional in a multi-spiking context, however, now with functionality well established producing less good but more generalized fits will become a higher imperative.

2.6.4 Feature Extraction

Each NeuronUnit test used in the optimizer represents the evaluation of a single feature of simulated output, for example the Input Resistance. I greatly extended the number of such features/tests covered by NeuronUnit in order to produce a rich, multi-objective optimizer that could capture important spiking dynamics and to obtain insights into what would be the most compact subset for subsequent use in unique identification of models.

2.6.4.1 Elephant Features Test Suite

Elephant (Denker *et al.*, 2018) provides feature extraction capabilities for membrane potential time series expressed using the Neo library in Python (Garcia *et al.*, 2014a). Eight NeuronUnit tests (five used here) are derived from Elephant feature extraction, particularly those associated with passive membrane properties assessed with subthreshold stimuli, or action potential waveform properties assessed at rheobase. Fundamental quantities such as capacitance or input resistance are among these, though they are not “emergent properties" of the model since they are roughly predictable from the parameters of the model equations.

2.6.4.2 Electrophysiology Feature Extraction Library (EFEL)

The Blue Brain Project developed the Electrophysiology Feature Extraction Library (EFEL) to compute many such statistics from spike trains, and I used these to generate tests of suprathreshold dynamics for optimization (Van Geit, 2015b).

2.6.4.3 Allen Institute Software Development Kit (Allen SDK)

The Allen Institute offers yet another set of tools for feature extraction, applying to both sub- and suprathreshold features of neuron responses to current injection. The reason to use Allen features in addition to the above is that some of these features are predicated on particular current stimuli (e.g. a stimulus that is exactly 20 pA stronger than the rheobase current).

Such stimuli either were or were not delivered to the various experimentally recorded neurons, and for the purposes of this thesis there is no going back and delivering additional ones. Consequently, for model testing and optimization it makes sense to use features–and the stimuli that generated them–that can be directly compared to the recorded neurons. For the biological neurons in the Allen Cell Types database, these features are available in the Allen SDK.

Chapter 3

RESULTS

In the Results section I hope to help the reader understand what I discovered about optimization of reduce neuron models. How well did it work? What do these optimized models look like? And is it possible to improve existing published models via optimization?

In section 3.1, I show how well and under what circumstances the optimizer can work on recovering ground-truth model parameters using simulated data as input. This is an essential step, since if an model optimized on simulated data does not match the model that simulated it, the whole enterprise can be called into question.

In section 3.2, I use various suites of biological-data-driven tests to optimize reduced models corresponding to real neuron types. I show how some of the assumptions and methodological approaches underlying the a subset of the tests might be problematic, and determine their impact on optimization quality.

In section 3.3 and 3.4, I assess the quality of these optimizations. While I focus on representative cases in this section, an exhaustive account of all optimized models is also available in the Appendix. I show which models lead to the best fits overall, and for which neuron types.

In section 3.5, I show that most published models actually deviate significantly from biological experimental data. I locate specific features underlying this disagreement, and explain how this creates an opportunity for optimization to close the gap. I then show that optimization can bring the behavior of the model and biological neuron back into agreement.

Finally, in section 3.6, I demonstrate an novel web application for optimization that I created to showcase the work described in the other sections. This web application can be used to setup, execute, and visualize optimization in real applications or to teach the relevant technical concepts to trainees.

3.1 Verification of the Optimizer

The reduced models I used are known to be too simple to precisely match all electrophysiological features exhibited by all cell types; this can be an advantage, as this also means that they are too inflexible to fit the “noise" in the data, whether it be random thermal fluctuations or systematic recording errors. These latter may be quite common; sites like NeuroElectro control only whether the extracted data is faithful to that reported in the publication, not that it reflects what was seen in the actual experiments, or that those experiments were carefully performed. Therefore, if the first step were applying the optimizer to fit models to real biological data, we would have no idea whether any optimization failures (poorly fitting models) reflected the limitations of the biological data, the limitations of the underlying model, or the limitations of the optimizer algorithms.

It is therefore necessary to create data-source independent “ground truths", by simulating data using the models, and using the optimizer to see if those model parameters can be recovered using that simulated data. Because genetic algorithms (such as the DEAP (Fortin *et al.*, 2012) implementation used here) have notably good performance in handling complicated error surfaces (such as Rastrigrin’s function), I expect that my optimization frame work, which derives from it, should also be able to handle potentially complicated features spaces such as one would expect from the output of a complex dynamical system like a neuron.

I wrote an algorithm that senses the edges of defined model parameter boundaries, and defines uniformly distributed random numbers within those boundaries. Alternatively I could have chosen to draw numbers from uniformly distributed random number generators, but doing so would risk testing only typical optimization cases and excluding edge cases.

	Observations	Predictions	Z-Scores
RheobaseTest	1.62 pA	1.62 pA	0
TimeConstantTest	13.18 ms	13.18 ms	0
RestingPotentialTest	-77.43 mV	-77.43 mV	0
InputResistanceTest	270.84 megaohm	270.84 megaohm	0
CapacitanceTest	48.65 pF	48.65 pF	0
FITest	7.51 Hz/pA	7.51 Hz/pA	0

Table 3: “Observed" and “Predicted" electrophysiological features match to within a few decimal places. Here, “observed" refers to values extracted from traces simulated from a ground-truth model. The optimizer has no knowledge of the parameters of that model, but uses these “observed features" to attempt to discover the parameters using a genetic algorithm. The algorithm modified these parameters in order to bring the “predicted" features values, extracted from new simulations, in line with the “observed" ones. Each row refers to a distinct NeuronUnit test used to simulate the model, extract a single electrophysiological feature, and compute a Z-score. The outcome of this algorithm is shown in this table, demonstrating that the optimizer has identified a parameter set that perfectly reproduces these features.

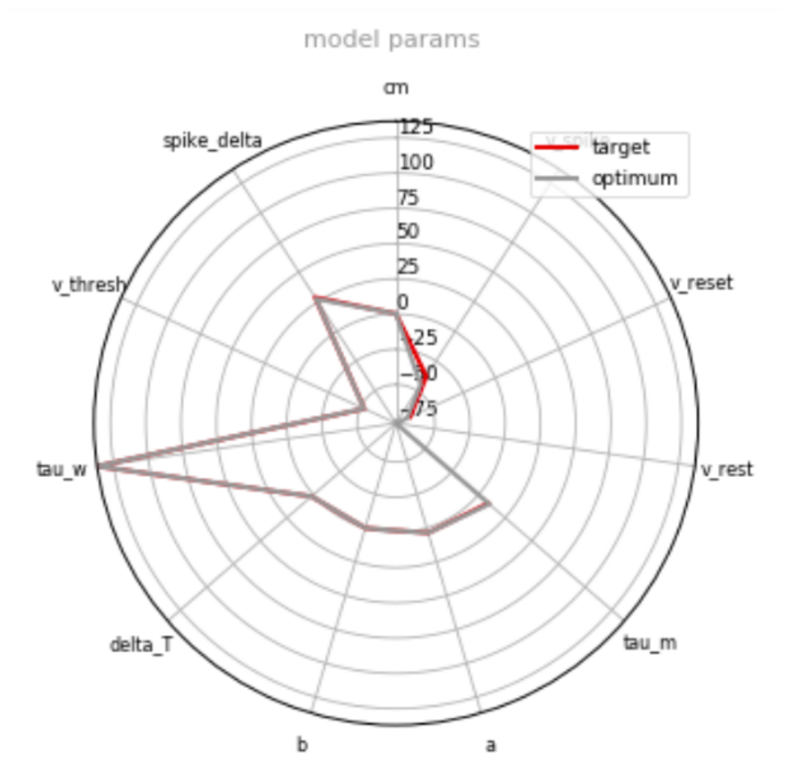


Figure 12: **Radar Plot of Optimizer Verification against Ground Truth.** “Target" refers to parameters from the ground-truth model. “Optimum" refers to parameters obtained by the optimizer when trying to match the features shown in Table 3. Despite having no knowledge of the ground truth parameters, the optimizer can discover them by feature-matching, producing a near perfect match. Nearly all of the 11 Izhikevich model parameters are accurately recovered, with only a trivial discrepancy in v_spike.

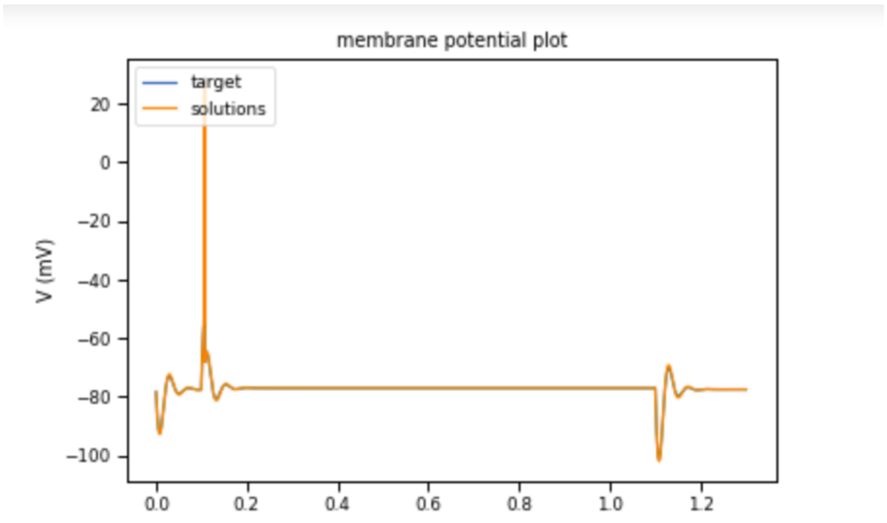


Figure 13: **Optimizer Verification Example 1.** The orange trace shows the model waveform recovered during the process of optimization. Blue trace (not visible due to occlusion by orange trace) depicts the simulated trace (in response to the rheobase current) from the ground truth Izhikevich model that was used to guide optimization. The horizontal axis shows simulation time in seconds. Clear agreement is shown in between simulated responses of the ground truth model and the optimized model from Figure 12.

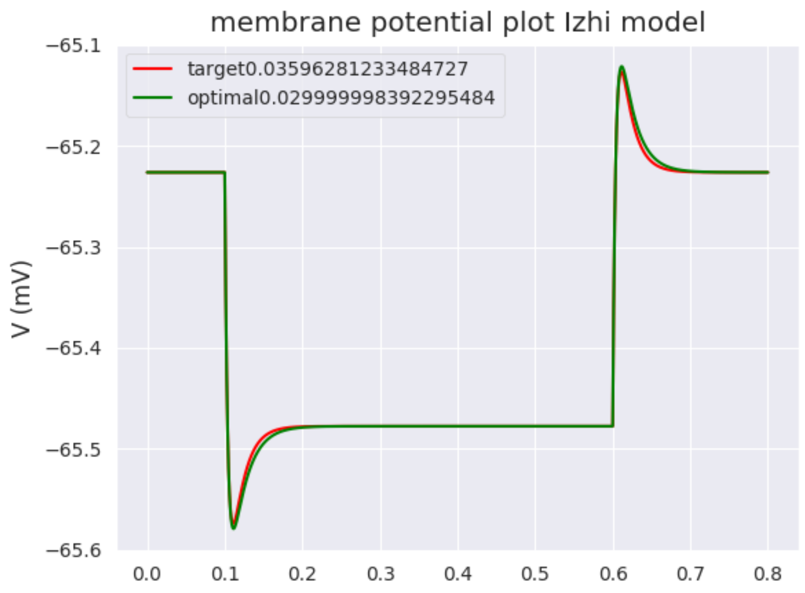


Figure 14: **Optimizer Verification Example 2.** Similar to Figure 13, but with a hyperpolarizing current injection value of $-10pA$ applied between $100ms - 600ms$ to the Izhikevich model. Here the red trace represents the response in the ground truth model, and the green trace represents the response in the optimized model, demonstrating that responses to subthreshold stimuli can be adequately reproduced by the optimizer.

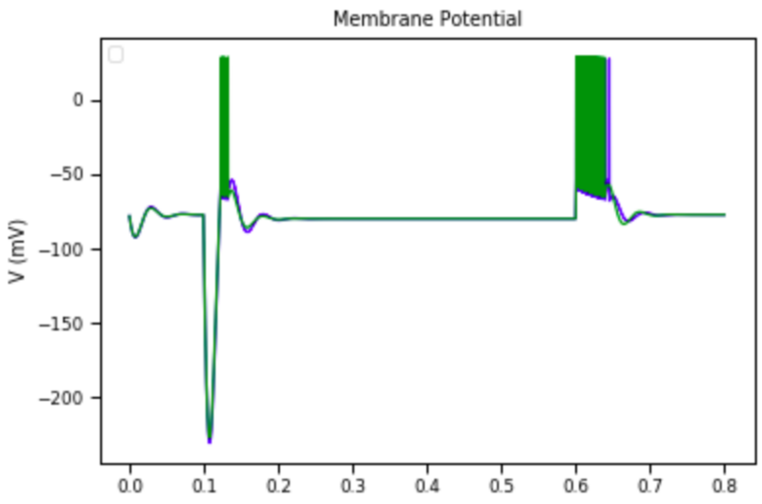


Figure 15: **Optimizer Verification Example 3.** Similar to Figure 14, but for the AdEx model. Because model parameters are assigned to the ground truth model randomly, an unusually small value of C in the AdEx model produces rebound spikes and bursts. This demonstrates that the match between the ground truth model and the optimized model is not limited to a single stimulus condition or exclusively to only typical model parameters.

3.1.1 Verification Endpoints

My optimizer was capable of identifying model parameterizations that nearly perfectly matched the parameters of the “ground truth" models that generated the constraining simulated data (Figure 12). Simulated output of the two models matched closely as well (as expected from similar model parameters) (Figures 13, 15, 14). NeuronUnit-based electrophysiological features extracted from the traces shown in these figures, and from other traces simulated in the course of optimization, also showed a near-perfect match (i.e. Z-score of 0, indicating perfect optimization) (Table 3).

In these results, only a small number of features were used (Time Constant, Capacitance, Rheobase, Resting Potential, Input Resistance, and FISlope); interestingly only two of these involved a measurement of action potentials. Other NeuronUnit tests that measure and judge models according to action potential *Width, Amplitude, or Threshold* did not participate in guiding optimization, for reasons that will be described in Section 4.2.4.

Results for additional models and parameterizations are given in the Appendix.

Failure of optimization verification, when it occurs, could be indicative of insufficient constraints. By analogy, when solving a system of linear equations, finding a unique solution requires that the number of constraining equations is greater than the number of free variables you are solving for. Similarly, in a genetic optimization algorithm we solve for unknown variables using stochastic principles, but the number of variables (i.e. model parameters) we can identify is still limited by the number of independent measurements of model output that we use. In other words, assuming that the number of objectives in the multiobjective optimization problem, $NOBJ$ is greater than $NDIM$, the number of model parameters, optimization should be achievable. Even when $NOBJ < NDIM$, such as in the examples above, optimization can still work due to correlations or redundancies in the model that lead the actual manifold on which models live to be of lower dimension than $NDIM$ itself.

3.1.2 Verification Efficiency

My optimizer can recover ground truth models from simulated data in several cases. Does it do so efficiently? How long does this optimization take, and does it get stuck exploring irrelevant regions of parameter space?

In Figure 16, I show how the optimized model converges towards the ground truth model over time. In can require up to 200 generations of parameter set evolution for tight convergence to be realized. This takes approximately $10 - 20$ minutes on a my personal laptop.

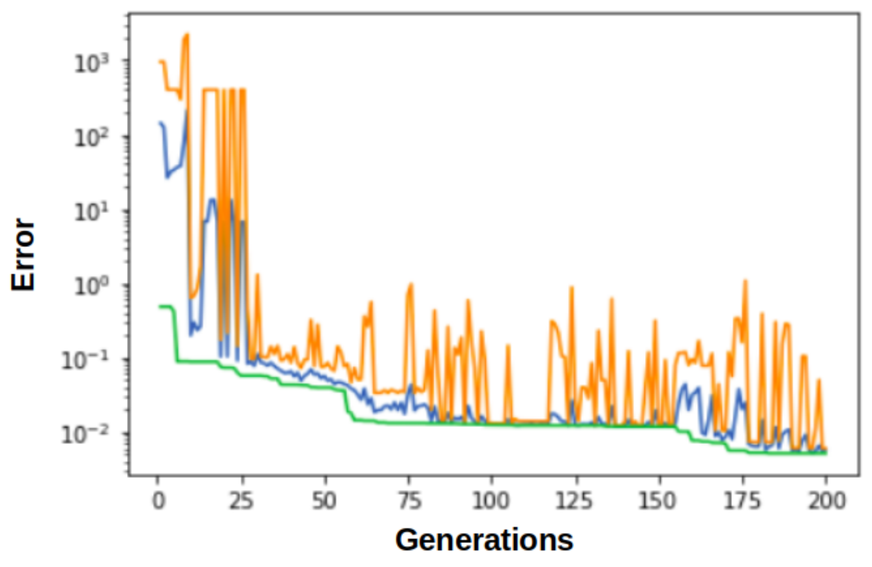


Figure 16: **Evolution of Optimized Model Quality over Generations.** The green line tracks the lowest error in each generation of candidate AdEx models, the blue line shows the average error, and the orange line shows the highest error. Optimization hyperparameters were: Number of Generations=200, Population Size=50, Crossover Probability=0.3, Mutation Probability=0.2. The periodic jumps in the orange error represent mutation or crossover events in which poor-performing models were generated. These were not typically selected into the subsequent generation. Because the crossover and mutation probabilities only sum to 0.5, the remaining half of all parameter sets in each generation either filtered out by selection or are carried over to the subsequent generation. Extensive hyperparameter tuning (not shown) demonstrated that this level of chromosome conservation was a good balance between exploration and exploitation.

3.1.3 Alternatives for Verification

The ground truth is known independently of the optimizer, so one can test alternative strategies to see if there is a solution that represent a better match to ground truth than the one obtained through optimization. For example, one can exhaustively search the solution space to look for the best model. An exhaustive search might be a reasonable (but lazy) approach when there are only a small number of free parameters, however using a sampling grid of 25 distinct parameter values for each of N parameters mean that 25^N total models must be examined. Even for a simple model ($N = 5$) this represents nearly 10,000,000 model evaluations, and of course the problem gets much worse with each additional parameter. 25 values for each parameter may also be insufficient resolution when the parameter regime exhibiting the desired behavior is narrow.

3.1.4 Implications of a Verified Optimizer

This success suggests that for reduced models, there is unlikely to be much degeneracy in model parameters where distinct combinations of parameters produce identical simulated responses across a range of stimuli. This might also be expected from the motivation of reduced models, which is to identify and consolidate redundant or unimportant biophysical equations/parameters into a handful of key reduced model equations/parameters. Importantly, I showed that the electrophysiological features used here, which correspond to measurements that can be made in real neurons, were sufficient for this task.

Confident that the optimizer can identify model parameterizations that can generate observed electrophysiological features *in principle*, the focus of the remaining sections becomes the localization of other potential sources of model/data disagreement.

3.2 Limitations of Optimization using Experimental Data

3.2.1 When is Optimization Possible Using Real Biological Data?

Not all of the available data-sets are conducive to optimizing reduced models. For example, consider the cerebellar Purkinje cell. The Purkinje cells has a very large surface area (much of it the dendrites that support $100,000$ synaptic inputs). It consequently has a large capacitance and a low input resistance, and as such it demands a very large current stimulus to elicit a rheobase spike: $680pA$. However, most reduced cells typically cannot exhibit such a large rheobase under any parameterization that otherwise looks like a neuron model. The fact that an expansive dendritic tree is able to absorb so much somatically injected current may be difficult for a reduced model to capture. Specifically, the upper limit for rheobase found in results was typically as $350 - 400pA$, but even this comes at the expense of sacrificing fit quality for all other electrophysiological features. Consequently, optimized models of the Purkinje cell always failed to be biologically plausible. The p-value of the χ^2 statistic was always sufficiently low to reject the null hypothesis that such optimized models were representative of the biological data distribution. Like the Purkinje cell, the Mitral cells of the main olfactory bulb also escaped successful model fitting. These mitral cells also have high membrane capacitance $235pF$, and reduced models could not reproduce their features well. The Izhikevich model was achieved the lowest overall χ^2 statistic for Purkinje cells and Mitral cells, being slightly more flexible than the

Test name	observations	predictions	Z-Scores
RheobaseTest	190.0 pA	199.52 pA	0.04
TimeConstantTest	13.8 ms	6.21 ms	0.32
RestingPotentialTest	-77.5 mV	-39.29 mV	0.26
InputResistanceTest	132.0 $M\Omega$	44.94 $M\Omega$	0.45

Table 4: Predicted and observed features for neuron 471819401 from the Allen Cell Types database, following optimization of the AdEx model against data from this neuron. Other neurons showed similar optimization performance, and are shown in the Appendix.

AdEx or conductance based models (see Tables 4 and 5). In general, however, these reduced models may have been developed with smaller, more electronically compact cortical and hippocampal cells in mind.

3.2.2 Conflicts between Experimental Features Constraining Optimization

Feature values extracted from multiple data sets appeared to be in conflict for some cell types. For example in the section below I show that the rheobase value was often incompatible with some passive electrophysiological feature values, such that good optimization could be achieved with one set or the other, but not both together.

3.2.2.1 Tradeoff Patterns in Data-driven Tests in Subtheshold and at Threshold Electrical Properties

Using data from the Allen Cell Types neuron with ID 471819401, I was able to optimize both AdEx and Izhkevich models, such that both models would agree with rheobase, time constant, resting membrane potential, and input resistance, experimental values. Some tradeoffs were needed in order match all of the values, as these features could not all be perfectly matched at once.

	observations	predictions	Z-Scores
RheobaseTest	190.0 pA	190.48 pA	0
TimeConstantTest	13.8 ms	1.9 ms	0.94
RestingPotentialTest	-77.5 mV	-70.65 mV	0.03
InputResistanceTest	132.0 $M\Omega$	25.47 $M\Omega$	0.74

Table 5: Same as the Table 4 but for the Izhikevich model.

	observations	predictions	Z-Scores
FITest	0.18 Hz/pA	0.18 Hz/pA	0.01
RheobaseTest	70.0 pA	70.26 pA	0

chi_square	0.000083
p_value	1.000000

Table 6: Summary statistics of fit quality using AdEx model, when it is constrained against FITest, and Rheobase Test alone. The very low χ^2 statistic (and high p-value) indicate that there was no evidence that this optimized cell model produced behavior outside of the range of biological neurons of the same type, at least for the features examined here.

3.2.2.2 6239608801 AdEx

The need to match the rheobase appeared to be interfering with the ability of the optimizers to match many other features. The rheobase is essentially the knee of the FI curve. An alternative strategy is to produce models that match the slope of the FI curve. Here I show two out of 12 examples demonstrating that that fitting models to the FISlope and rheobase only, leads to generally better agreement as there is less conflict between these two tests.

Features	observations	predictions	Z-Scores
FITest	0.18 Hz/pA	0.18 Hz/pA	0
RheobaseTest	70.0 pA	66.61 pA	0.04

chi_square	0.00155
p_value	1.00000

Table 7: Similar to Table 6, but for Izhikevich model. Since two major model classes are better able to fit to FITest and Rheobase alone, it suggests that these two measurements might be less conflicted in models.

3.2.2.3 6239608801 Izhikevich

By “good optimization" I mean that the optimized model exhibited behavior that was consistent with all of the features. Models seemed to have particular difficulty in recapitulating an accurate fit for rheobase, while simultaneously satisfying the fitness criteria imposed by the time constant, input resistance, capacitance and resting membrane potential. This was less problematic when using the slope of the FI curve as a feature, suggested that it was not spiking *per se* that caused the problem.

This was also evident when working across datasets. For example, when optimizing against data from both NeuroElectro and the Allen Cell types database, it was typically impossible to satisfy data-derived features from both sources simultaneously, even when the same nominal neuron type was being described.

3.2.3 Experiment Fitted Model Results on Reported Data Types

Because some features derived from the data were incompatible or unreliable, it was necessary to create additional NeuronUnit tests from other feature extraction

Source	Cell ID	Subset of already listed tests	Sample type
Allen	48249376[1]	Y	Single cell
Allen	47181940[1]	Y	Single cell
Allen	623893177[1]	Y	Single cell
Allen	623960880[1]	Y	Single cell
Allen	482493761	N	Single cell
Allen	471819401	N	Single cell
Allen	623893177	N	Single cell
Allen	623960880	N	Single cell
NeuroElectro	Olfactory Mitral Cell	N	Mean of cells
NeuroElectro	Neocortex pyramidal cell layer 5-6	N	Mean of cells
NeuroElectro	Cerebellum Purkinje cell	N	Mean of cells
NeuroElectro	Hippocampus CA1 pyramidal cell	N	Mean of cells

Table 8: Properties of the experimental data used in this section. Two data sources (NeuroElectro and the Allen Cell Types database) were used to fit neuron models. All Allen Cell Types data come from cortical neurons of various types. Allen Cell Types data correspond to individual recorded cells, and NeuroElectro data come from mean feature values reported across many neurons of the same nominal type.

name	CA1 pyramidal	Purkinje	NCP Layer 5-6	Mitral	623960880	623893177	471819401	482493761	6239608801	6238931771	4718194011	4824937611
RheobaseTest	189.24 pA	680.79 pA	213.85 pA	NaN	70.0 pA	190.0 pA	190.0 pA	70.0 pA	70.0 pA	190.0 pA	190.0 pA	70.0 pA
InputResistanceTest	107.08 Mohm	142.06 Mohm	120.67 Mohm	130.08 Mohm	241.0 megaohm	136.0 megaohm	132.0 megaohm	132.0 megaohm	NaN	NaN	NaN	NaN
TimeConstantTest	24.5 ms	NaN	15.73 ms	24.48 ms	23.8 ms	27.8 ms	13.8 ms	24.4 ms	NaN	NaN	NaN	NaN
CapacitanceTest	89.8 pF	620.27 pF	150.58 pF	235.75 pF	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
RestingPotentialTest	-65.23 mV	-61.59 mV	-68.25 mV	-58.14 mV	-65.1 mV	-77.0 mV	-77.5 mV	-71.6 mV	NaN	NaN	NaN	NaN
InjectedCurrentAPWidthTest	1.32 ms	0.41 ms	1.21 ms	1.61 ms	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
InjectedCurrentAPAmplitudeTest	86.36 mV	71.23 mV	80.44 mV	68.4 mV	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
InjectedCurrentAPThresholdTest	-47.6 mV	-46.89 mV	-42.74 mV	-38.9 mV	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
FITest	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.18 Hz/pA	0.12 Hz/pA	0.18 Hz/pA	0.09 Hz/pA

Table 9: Reported Cell Features for 12 cells. The first 4 are taken from NeuroElectro and the remaining 8 from the Allen Cell Types database. Unreported data values are encoded as “NaN".

libraries, as described in the Methods. We used data from four different distinct cell types with data obtained from NeuroElectro and four different single cortical cells from the Allen Cell Types database, for a total of eight sets of feature data to optimize. We then optimized against this data using a suite of nine NeuronUnit tests. Additionally, to explore if FITests, and Rheobase fitted better on their own, an additional four non-unique test sets where also created. The entire set of tests is presented in tabular form (9). See C in the Appendix for the URLS for the data sources.

I then used three different models (AdEx, Izhikevich, and a conductance-based point neuron) and optimized them using these test suites. I attempted to also use GLIF

Model Type	Experimental Data ID	χ^2	p-value
IZHI	4824937611	0.034791	1.000000e+00
IZHI	4718194011	0.051691	1.000000e+00
IZHI	6238931771	0.086530	9.999999e-01
IZHI	6239608801	0.001550	1.000000e+00
IZHI	482493761	1.292924	9.956362e-01
IZHI	471819401	1.443618	9.936050e-01
IZHI	623893177	1.334789	9.951233e-01
IZHI	623960880	1.014464	9.981553e-01
IZHI	Olfactory Bulb Mitral Cell	6915.484007	0.000000e+00
IZHI	Neocortex pyramidal cell layer 5-6	2.443396	9.643182e-01
IZHI	Cerebellum Purkinje cell	19.885113	1.077956e-02
IZHI	Hippocampus CA1 pyramidal cell	1.273070	9.958661e-01

Table 10: Fit quality for two model types optimized against the data in Table 9. Lower χ^2 and higher p-value represents a better fit. p-values > 0.05 lack evidence to reject the null hypothesis that the optimized model comes from the same distribution as the experimental data. p=1 indicates perfects agreement between the optimized model and the experimental data on all measured features.

models here, although test results often looked convincing, GLIF model waveforms looked strange. The result is $3 \times 8 = 24$ model-data combinations. For each each member of this 24 element matrix we wanted to know if the fitted model behaved in a biological plausible manner, we were interested to know if fitted models were convincing mimics of *in vivo* cells, at least with respect to the measurements models were trained to fit. In the process of coalescing results the single 24 element matrix, the matrix was broken into two: one 24 element matrix (Table 8) and a smaller matrix consisting of only the conductance based model test combinations that executed without failure (Table 12).

Model Type	Experimental Data ID	χ^2	p-value
ADEXP	4824937611	0.000017	1.000000e+00
ADEXP	4718194011	0.011029	1.000000e+00
ADEXP	6238931771	0.005062	1.000000e+00
ADEXP	6239608801	0.000083	1.000000e+00
ADEXP	482493761	86.949529	1.887379e-15
ADEXP	471819401	0.370856	9.999575e-01
ADEXP	623893177	0.273030	9.999870e-01
ADEXP	623960880	0.140222	9.999990e-01
ADEXP	Olfactory Bulb Mitral Cell	10.353878	2.410614e-01
ADEXP	Neocortex pyramidal cell layer 5-6	0.013262	1.000000e+00
ADEXP	Cerebellum Purkinje cell	353.692447	0.000000e+00
ADEXP	Hippocampus CA1 pyramidal cell	0.735616	9.994308e-01

Table 11: Same as Table 10, but for AdEx models.

Model type	Experimental Data ID	χ^2	p-value
conductance model	Hippocampus CA1 pyramidal cell	17.21	0.027
conductance model	olf-mit	26487.51	0.0
conductance model	Neo cortex pyramidal cell layer 5-6	2.56	0.95
conductance model	4824937611	2.17	0.97
conductance model	471819401	0.870	0.99
conductance model	482493761	0.036	0.99
conductance model	6238931771	1.441259	0.993641

Table 12: Same as Table 10, but for a single-compartment conductance-based model.

3.2.4 Across Model Performance Comparisons

As stated in the Methods, I use the χ^2 statistic as a summary of optimized model quality, with smaller values reflecting fits that better recapitulate the biological experimental data, and non-significant p-values–lack of evidence that the model disagrees with that data–as evidence of success. The Izhikevich Model and a Conductance-Based

Point Neuron Model were able to achieve such small χ^2 statistics (and non-significant p-values) when seven or eight of the tests (excluding rheobase) were considered together. For example the Izhikevich model fitted to a Hippocampus CA1 pyramidal cell data achieved (χ^2, p-value) = (2.13, 0.98), and the Olfactory Bulb Mitral cell achieved (χ^2, p-value) = (2.02, 0.98). A model whose every feature was exactly equal to the mean observed in the experimental data would have all Z-scores equal to 0, a χ^2 statistic of 0, and a p-value of 1, by definition. Thus an extremely high p-value (such as those above) is evidence that the optimized model is much closer to the mean of the data distribution than a random experimental neuron. This is exactly the result one would expect from successful optimization.

3.2.5 Sources of Optimization Failures

When optimization was not successful, was this due to a fundamental inability of a given reduced model to represent the behavior of a given neuron type? In order to make this claim, I first had to rule out alternative possibilities.

3.2.5.1 Distributions not Well-summarized by the Mean

The optimizer fits to the mean of a feature value, but as shown in section 2.5.1, the mean value of a feature is in some cases a misleading summary of the typical values. For example, the olfactory bulb Mitral cell exhibited bi-modal feature distributions (possibly due to lumping with the Tufted cell, a distinction that may not have been appreciated when the data was originally collected). Beyond that, the Neuroelectro data reflects a mean over different laboratories, animals, and recording epochs. The mean of a population can often be a robust summary, however there are circumstances when this is not true. I plotted all the feature distributions for all the neurons used here (see Appendix), examined these by eye and made note of those where the mean was not a good summary of the typical value (due to multimodality, extreme skew, or small sample size). Note that this only applies to the Neuroelectro data; the other data sources report features for single instances of neurons, so the model being fit is a model of that specific neuron, not of a neuron type more generally.

3.2.5.2 Distributions with an Uncertain Mean

The NeuroElectro data represented distributions over cells of the same nominal type. For some cell types, reduced models were hard to optimize against these data even when the distributions are normal-like, and thus the mean and the mode are well-matched. In order to understand whether the source of these difficulties was in the data themselves, I examined the standard error of the mean (SEM) for each feature. Like the standard deviation, the SEM is a prediction of uncertainty in each measurement, but it reflects uncertainty about the value of the mean itself, rather than simply the variability in the measurement across neurons of the same type. A large SEM might reflect such variability, or simply a small sample size. In either case, a large SEM means that the optimization target may not reflect the true properties of a typical neuron of that type. These SEM values, for several cells and electrophysiological features, are shown in Table 13.

Neuron Type	Rheobase	SpikeThreshold	SpikeHalfWidth	SpikeAmplitude	MembraneTimeConstant	RestingMembranePotential	InputResistance
Hippocampus CA1 pyramidal cell	122.88	1.85	0.12	3.68	3.88	0.72	12.57
Olfactory bulb (main) mitral cell	NaN	5.69	0.12	2.83	5.42	1.39	20.17
Cerebellum Purkinje cell	419.81	2.00	0.05	0.57	NaN	3.69	19.26
Neocortex pyramidal cell layer 5-6	128.87	1.92	0.17	1.49	2.56	1.84	27.93

Table 13: The standard error of the mean (SEM) described the uncertainty that the sample mean value of a feature (across neurons) is close to the population mean. The SEM values describe that uncertainty for the sample means of various features for each of 4 cell types in NeuroElectro.

3.2.6 What is the Value of a Model Based on Average Feature Values?

I am not the first to observe that fitting a model to the mean of feature values across a distribution of cells of the same type may not result in a model that well-described an actual neuron from that distribution. Neuron modelers have some awareness of this problem which has been thoroughly characterized in Marder and Taylor (2011), though it remains underappreciated. This is especially obvious when the feature distribution is multi-modal (such as inset-e in the figure below), but it could also result from hidden covariance structure among the features (Figure 17).

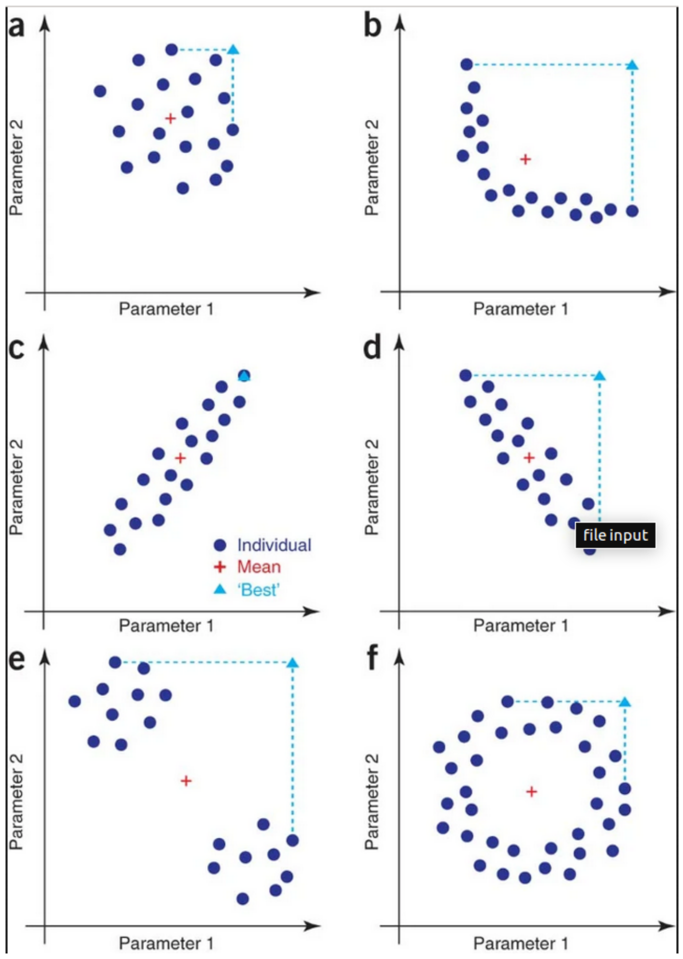


Figure 17: **Lessons from Marder.** As shown in Marder and Taylor (2011), when neurons of a given type are distributed through parameter space, the mean of these parameters across neurons (red cross) is at a separate location from the model one would obtain by using different neurons to contrain different parameters (blue triangle).

Previously in the Methods (Section 2.5.1), we showed that some feature distributions are bimodal. Consider a cell type which had an underlying bimodal distribution for input resistance, with a mean exactly between the two modes. If we optimize a neuron against this mean value, then the optimized neuron does not describe the modes (where most of the cells are actually located) at all. Conversely, suppose that we optimize each neuron from the distribution individually, and then produce a model which just averages across the parameter values of each of these optimized models. Its entirely conceivable that that this “mean model" would produce measurements that were similar to the mean of the feature distribution. However, when we are dealing with a nonlinear dynamical system, this is by no means certain (and perhaps not even likely). This mean model could also exhibit much higher or lower feature values. For example, we expect that even these comparatively simple reduced models have multiple regimes characterized by sharp transitions in parameter space between, for example, regular spiking behavior and bursting behavior, or tonic-bursting vs chattering. What would it even mean to split the difference between two such fundamentally different regimes?

In order to examine this question further in the context of my optimizer applied to reduced models, I created two models from each model class (e.g. Izhikevich) with different sets of parameter values. I simulated each of these models, computed all of the measurements associated with the core NeuronUnit tests (rheobase, input resistance, capacitance, and membrane time constant), and then simply averaged them together across the two models. I call these the “mean features" as each one reflects the mean across two parameterizations of the model. In parallel, I also create a “mean model" (Figure 18) which reflects the average of the parameters themselves, and then measure the same set of features for this mean model. I then ask the question: "Do the features from the mean model match the mean features?" If these match nearly all of the time, we can feel confident that optimizing a model based on the mean features will produce a model that is representative of the parameter space of the neurons individually. In other words, if we had access to the measurements from the individual neurons, and fit models to those, then the mean model obtained above should lie somewhere between those models in parameter space. By contrast, if the mean model does not match the mean features, that would suggest that models obtained by optimizing against mean features may not be representative in this context.

The results are shown below. In Figure 18, I show a representative example using two random locations in parameter space. While some of the differences between the two parameter sets may look small, this is only a reflection of the scale being dominated by the parameters that intrinsically have the largest values.

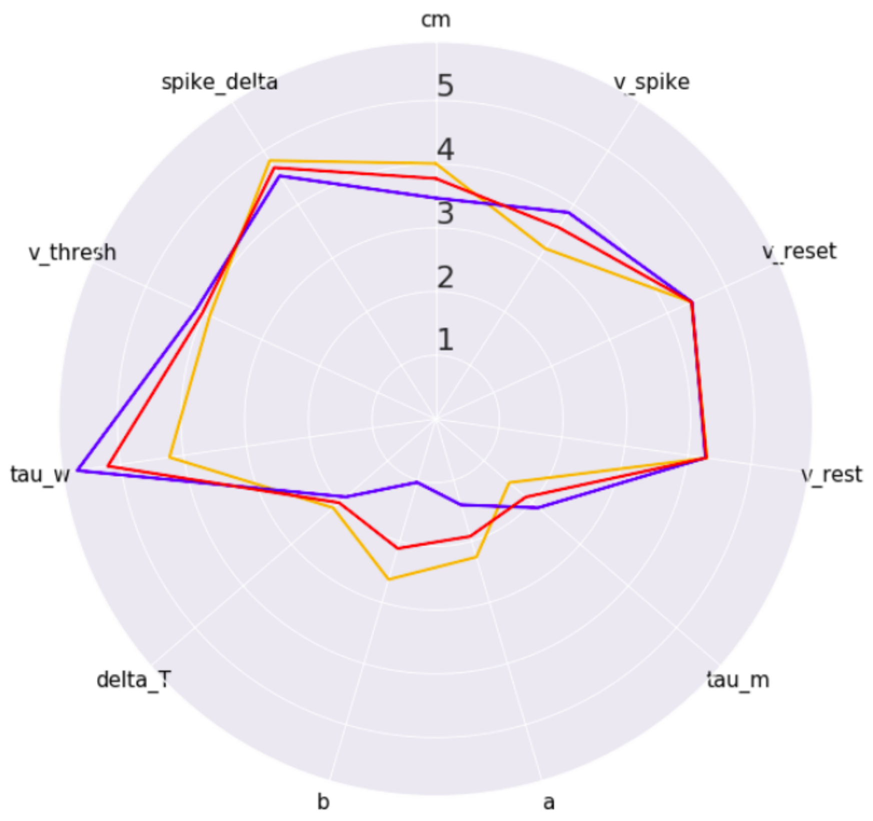


Figure 18: **Construction of a Model from the Mean.** Two random AdEx models (yellow and purple traces) are drawn from multivariate uniformly distributed random number generators. The mean model (red trace) is defined as the location in parameter space exactly in between the two random models in every dimension. Logarithms and absolute values are used to scale the radial dimension for clarity.

I then compute an index of agreement between features derived from the mean model vs mean features observed across the original two models. This index, $\alpha = |\frac{(x-a)}{(b-a)}|$, where a and b are the feature values for the original two models, and x is the feature value for the mean model, can be interpreted as follows: If $\alpha = 0.5$, then the mean model produces features precisely in between those produced by the two original models (Figure 19). If $\alpha = 0$ or $\alpha = 1$, then the mean model is nearly identical (in that feature) to one of the original models. An example of values ranging between 0 and 1 is shown in Figure 20. If $\alpha > 1$, then the mean model produces entirely new feature values that would not be expected from an interpolating between the two original models.

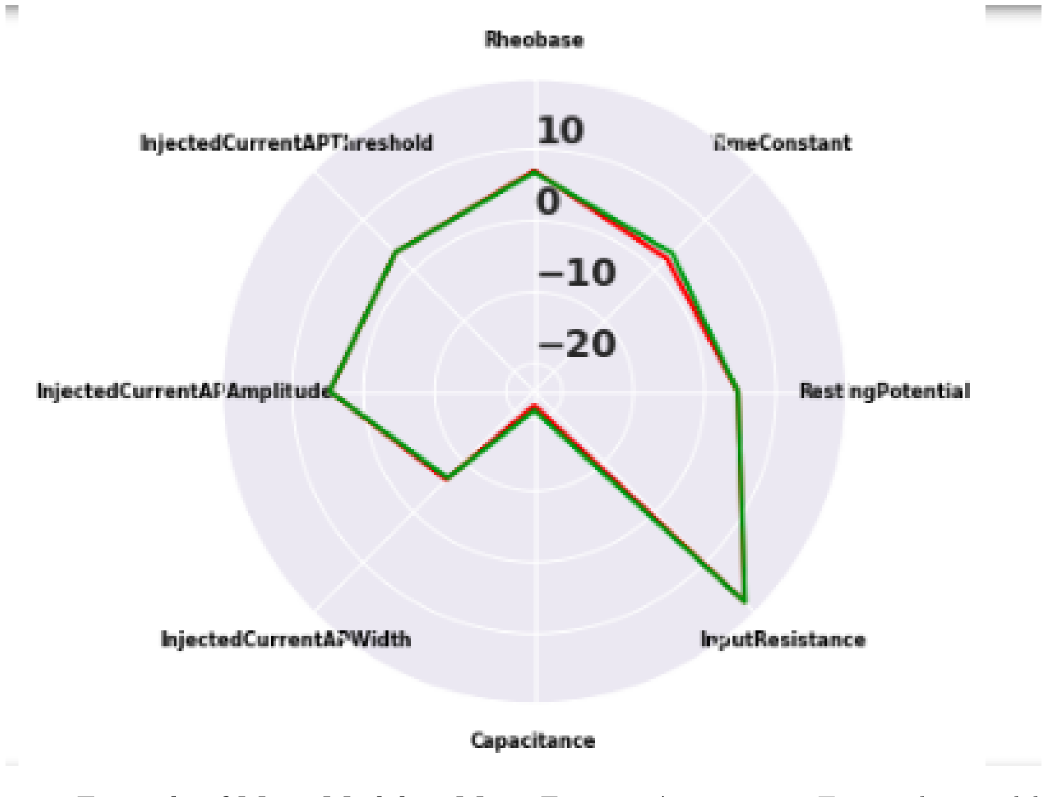


Figure 19: **Example of Mean Model vs Mean Feature Agreement**. Two random models can produce feature values whose mean (green) is equal to the feature values (red) produced by a model defined by the mean of their parameters. In such case the index of agreement $\alpha = 0.5$.

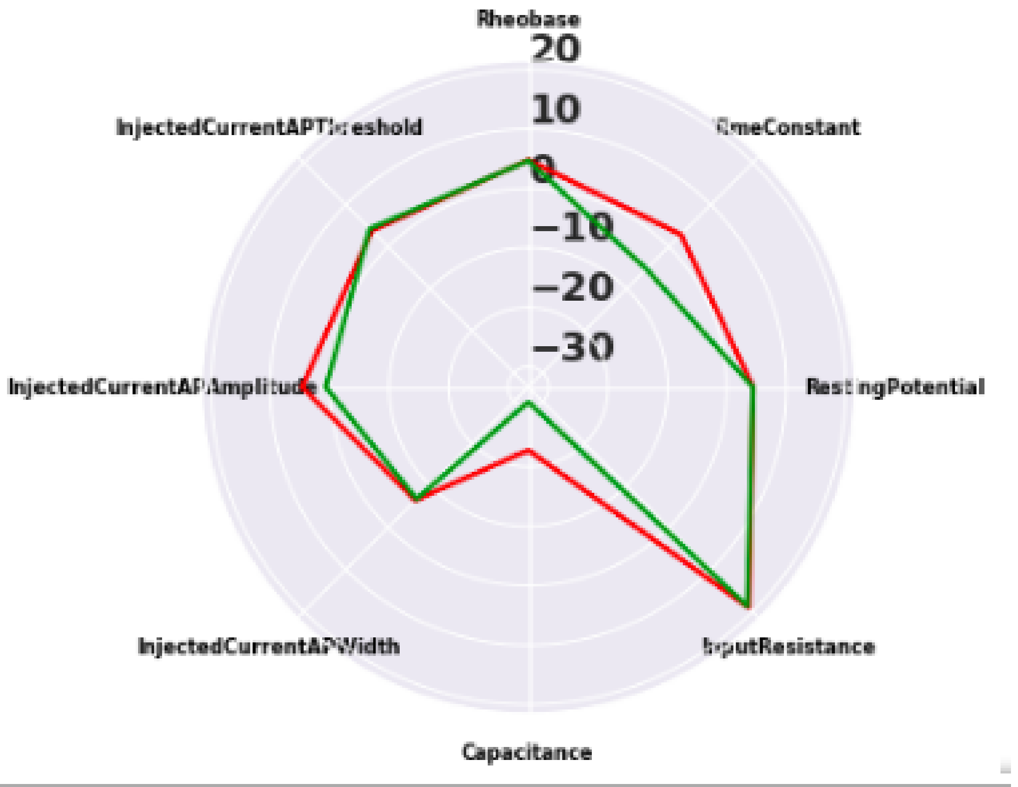


Figure 20: **Example of Mean Model vs Mean Feature Disagreement.** Similar to Figure 19, but now showing several disagreements, between mean model and mean feature value (in Capacitance, Time Constant, and AP Amplitude).

Figures 21 and 22 show that, while $\alpha = 0.5$ is the mode of the distribution as might be expected, considerable mass lies outside the mode, including many values of $\alpha > 1$. Therefore any model optimized against the mean feature value across neuron cannot be equivalent to a model drawn from the mean of those neurons' multivariate parameter distribution.

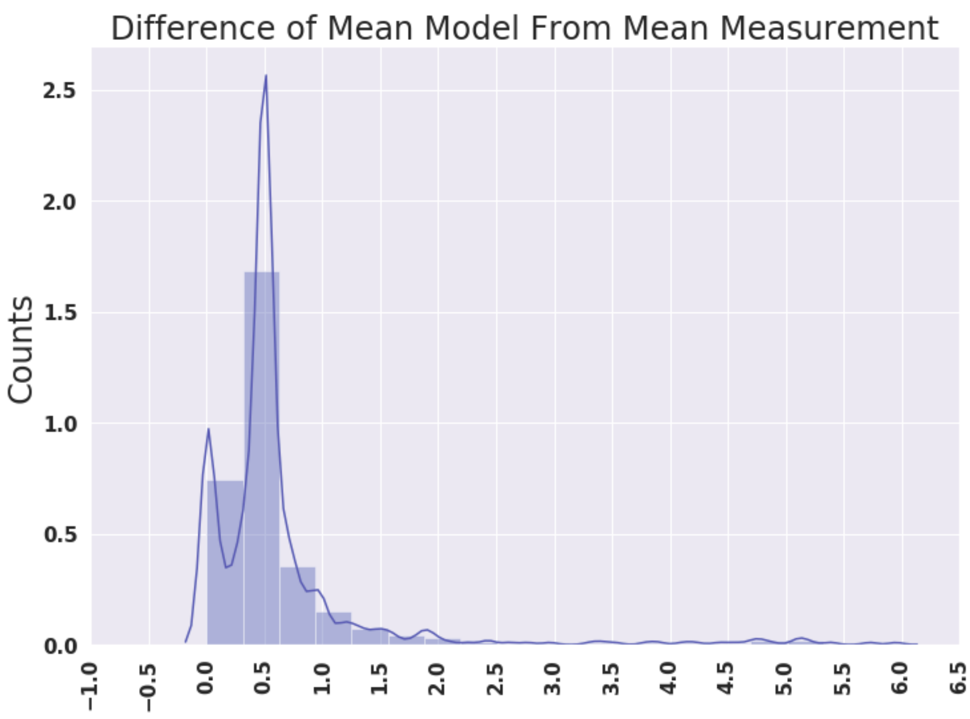


Figure 21: **Mean Model vs Mean Features for AdEx Model.** For 100 pairs of randomly sampled AdEx models, I computed the index α of mean model vs mean feature agreement, defined (see text) such that a value of 0.5 indicates that the mean model produces features identical to those obtained by simply averaging feature values produced by the original models. The mode lies at 0.5, but density in other regions indicates that many mean models do not agree with the mean feature value.

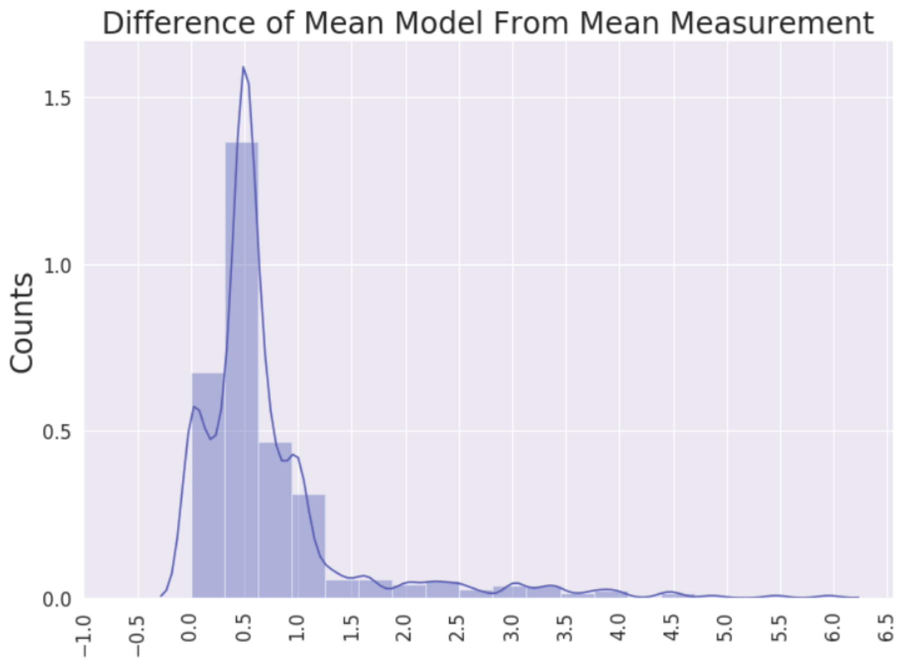


Figure 22: **Mean Model vs Mean Features for the Izhikevich Model.** Same as Figure 21, but for the Izhikevich model.

In Figures 19 and 20, I show the computed feature values from simulations of the constructed as in Figure 18. While in some cases the mean model produces feature values that lie between (are equal to the mean of) the feature values of the original two models, in other cases there is a massive discrepancy. This discrepancy appears more common in the Izhikevich model than in the AdEx model, potentially because there is a greater likelihood that the two original random models describe different dynamical regimes.

During the course of this work I observed that generally the mapping between reduced model parameters space and the feature space is not isomorphic, and thus fitting models to the mean feature values for a distribution of neurons is a dangerous exercise even if the data are normal-like.

3.3 Which Limitations are Due to Abstract Models?

Some of the limitations in the optimization of reduced models may come from their form; the abstractions they encode in their equations and parameters may come at the price of failing to exhibit key electrophysiological behaviors.

3.3.1 A Somatosensory Layer 5 Pyramidal Neuron Model

In order to test this, I extend the same optimization framework to a multi-compartmental, conductance based layer 5 pyramidal cell model (Van Geit *et al.*, 2016a) (Figure 23). This model subserved as one of many component neurons in the original formulation of the Blue Brain Project Markram *et al.* (2015).

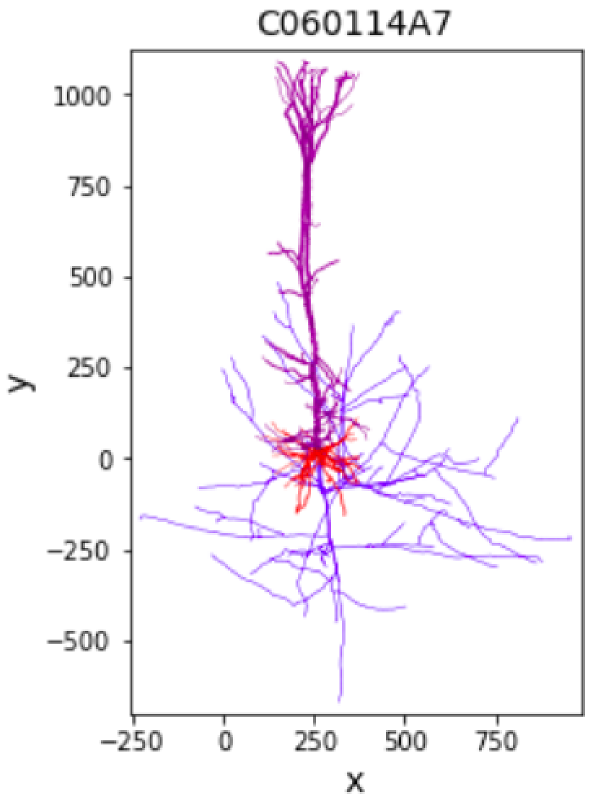


Figure 23: **Visualization of a Reconstructed Layer 5 Pyramidal Neuron.** Unlike the reduced model which has point geometry, the Layer 5 pyramidal neuron is spatially extended. Regions of the neuron with distinct sets of parameters are color coded: apical dendrites (magenta), basal dendrites (red), and axons (purple).

This elaborate biophysical model is philosophically the opposite of the reduced models focused on in the majority of my thesis work. This model incorporates complex, biophysically detailed phenomena instead of excluding them in the hopes of capturing the key dynamics of membrane potential time series. For example, this model is capable of a phenomenon the back-propagating action potential (BaP). BaPs can move travel in reverse direction, when they originate from distal dendrites and arrive at the soma where BaPs are free to sumate with incoming EPSPs (Spruston *et al.* (2013)).

This results in action potential waveforms far more complex than those produced by reduced models (none of which implement BaP waveforms at all) are capable of (Figs. 24 and 25).

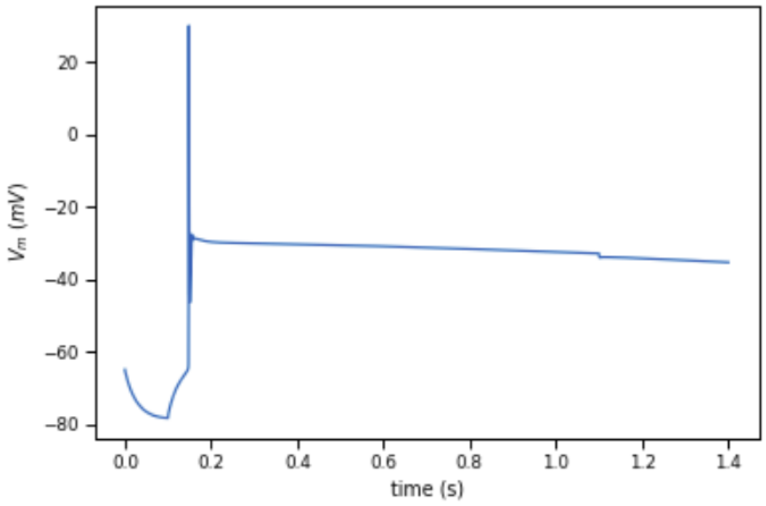


Figure 24: **AP Waveform in the L5PC Model**. A current injection just sufficient for causing a single action potential (AP) is applied to the L5PC model soma, for one second from $100ms - 1100ms$ in this model. The AP waveform is briefer but more complex than reduced model AP shapes. After the AP is complete, the somatic voltage plateaus at a depolarized level for the entire length of the simulation. Genuine plateau potentials do occur in the L5PC dendrites, where Ca^{2+} channels play a role (Zhu, 2000). Plateau potentials in the dendrites typically result in a burst in the soma *in vitro*. By contrast, a plateau potential in the soma is less common. This phenomenon is driven by coupling to the distal dendrites, and are thus obviously neglected in reduced models of point neurons.

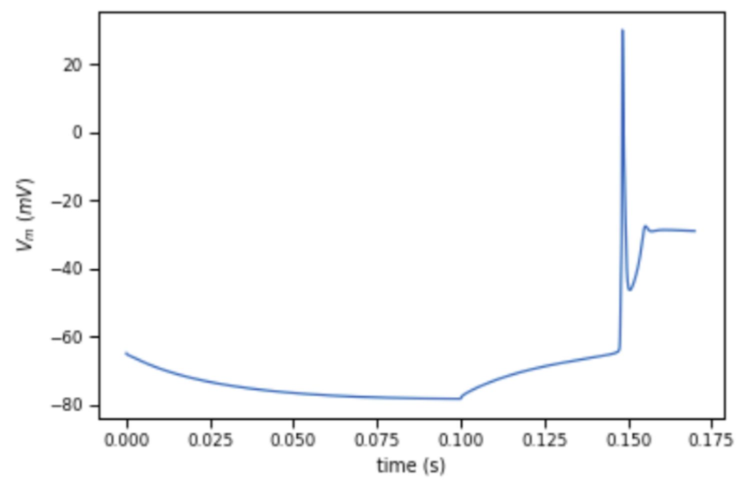


Figure 25: **AP Waveform in the L5PC Model.** Same as Fig. 24, but zoomed into show spike onset and after hyperpolarization.

The model itself is extremely complex and contains many free parameters. Some of these can be seen in the partial list of parameters in Figure 14. For example, these parameters include a number of distinct ionic conductances with distinct values in the soma, apical dendrites, and axon. There are also other parameters (not displayed in Figure 14) which are fixed, i.e. they are not modified during optimization.

An existing model of a layer 5 somatosensory rat, hind leg region was acquired by cloning an the BluePyOpt GitHub Repository (Van Geit *et al.*, 2016a). In order to optimize this model using NeuronUnit tests, I used a novel methodological approach described in the Appendix (B). I “streamed" waveforms that were evoked in the complex model, and assigned them to a temporary NeuronUnit model types as needed. The code used to implement this approach is available in the public repository: (Jarvis, 2020c).

I then applied my optimizer to this model and evaluated the quality of the resulting optimized model (Table 15). The unoptimized model had statistics:
$(\chi^2, p_{value}) = (13.56, 0.094)$
Whereas the optimized model produced statistics:

Parameter Name	Value
gNaTs2_tbar_NaTs2_t.apical	0.0261
gSKv3_1bar_SKv3_1.apical	0.0042
gImbar_Im.apical	0.000143
gNaTa_tbar_NaTa_t.axonal	3.138
gK_Tstbar_K_Tst.axonal	0.0893
gamma_CaDynamics_E2.axonal	0.00291
gNap_Et2bar_Nap_Et2.axonal	0.00683
gSK_E2bar_SK_E2.axonal	0.00710
gCa_HVAbar_Ca_HVA.axonal	0.000990
gK_Pstbar_K_Pst.axonal	0.974
gSKv3_1bar_SKv3_1.axonal	1.022
decay_CaDynamics_E2.axonal	287.2
gCa_LVAstbar_Ca_LVAst.axonal	0.00875
gamma_CaDynamics_E2.somatic	0.000609
gSKv3_1bar_SKv3_1.somatic	0.303
gSK_E2bar_SK_E2.somatic	0.00841
gCa_HVAbar_Ca_HVA.somatic	0.000994
gNaTs2_tbar_NaTs2_t.somatic	0.984
decay_CaDynamics_E2.somatic	210.5
gCa_LVAstbar_Ca_LVAst.somatic	0.000333

Table 14: The Layer 5 Pyramidal Cell model (L5PC, (Van Geit *et al.*, 2016a)) contains hundreds of parameters, but only a subset of these are tunable. A subset of 20, spanning conductances in the soma, apical dendrite, and axon, are shown here. Most models considered in this thesis have < 14 parameters; by contrast, having 20 parameters makes many approaches to model fitting intractable. My optimizer explore the space spanned by these 20 parameters to produce a Primar Somatosensory Cortex Layer 5 Pyramdial Cell model that agrees with the experimental data.

(χ^2, p_{value})=$(6.63, 0.57)$.

Therefore, both the optimized and unoptimized model are not decisively outside the realm of biological plausibility according to the core NeuronUnit features, and optimization clearly improves such plausibility. In both cases, this may in part reflect the highly variable nature of the biological/experimental results, rather than the verisimilitude of the optimized model. Note that high variability of the values recorded for a given feature makes it very easy to achieve a Z-score of low magnitude (by the definition of a Z-score).

Test Name	Observations	Predictions	Z-Scores	SEM
RheobaseTest	213.9 pA	225.0 pA	0.065	128.9
InputResistanceTest	120.7 Mohm	50.7 Mohm	-0.90	27.9
TimeConstantTest	15.7 ms	16.76 ms	0.14	2.6
CapacitanceTest	150.6 pF	330.7 pF	1.28	1.49
RestingPotentialTest	-68.3 mV	-78.1 mV	-1.5	44.0
InjectedCurrentAPWidthTest	1.21 ms	0.15 ms	-1.98	0.175
InjectedCurrentAPAmplitudeTest	80.4 mV	89.6 mV	0.72	1.49
InjectedCurrentAPThresholdTest	-42.7 mV	-59.6 mV	-2.09	1.92

Table 15: **Observed and Predicted features for the Layer 5 Pyramidal Cell**. After equipping the L5PC model for NeuronUnit testing, this model was optimized against L5PC data from NeuroElectro. Observed and predicted values could be made to agree on a subset of tests, such as rheobase and time constant, however in the larger test ensemble there was still a tradeoff in optimizing for one value vs another. While this model falls within the range of biological plausibility ($\chi^2 = 13.6; p = 0.094$), the failure of the optimized model to achieve Z-scores consistently near zero suggests that, even with an extremely flexible conductance based model containing many parameters, the mean feature values reported in Neuroelectro are mostly likely mutually incompatible within a single neuron.

It is also worth noting that optimized models vary depending upon which features are used to generated the constraining NeuronUnit test; one can improve optimization quality by choosing a subset of tests, as there was an irreconcilable trade-off between the membrane time constant, the action potential width, and the input resistance, with optimization only able to match the experimentally-reported means of only 1-2 (but not all 3) of these simultaneously. Given this experimental variability and this trade-off between features, the optimization is at least satisfactory. However, the inability of the optimizer to find an even better solution, with $Z = 0$ across all features, indicated that there are fundamental conflicts between reported mean values across NeuroElectro features. This conflict limits not only the possibility of optimizing reduced models, but also complex, multi-compartmental conductance-based models.

Therefore, reduced models do not appear to be the limiting concern; rather, data quality/consistency is.

3.3.1.1 Caveats to the L5PC Optimization

It is also likely that many of the reported values in NeuroElectro were taken from experiments conducted at non-physiological temperatures. Since temperature modulates action potential width Goldin and Mindlin (2017) (and indeed all kinetic parameters), it is possible that a different result would be obtained if all data had been collected under the same, physiological temperature.

NeuroElectro lumps together layer 5 cortical pyramidal cells neurons from prefrontal cortex, somatosensory cortex, and primary visual cortex together into one dataset. Therefore it is also possible that while mean features across neurons of a single L5PC type (e.g. somatosensory) would have been mutually consistent, the frankenstein resulting from lumping all of these together was the source of the problems.

Finally, the L5PC model was very slow to simulate relative to the reduced models developed in this thesis work. Where a single simulation of a typical reduced model described here is evaluated on average took only 0.0025 seconds, the L5PC model on average took 5.74 seconds; evaluating the rheobase current required 34.8 seconds. Consequently it was difficult to run tens of thousands of simulations (across chromosomes and generations), especially in a development/debugging context. The L5PC model documentation provide by the Blue Brain Project, suggests that $NGEN = 100; \mu = 100$ would define an appropriate GA search scale, however, due to time limitation I used smaller values. I nonetheless do not believe that longer optimization would have changed the major results, as the gains in fit quality would have been small. However this unintentionally supports a major thrust of this thesis: if simulating very complex, very slow models doesn't solve any problems that simulating simple, fast ones does, then why bother with the former?

3.4 Optimized Single Neurons

In the previous sections we saw that optimization of models to the mean features reported across many neurons, even of the same nominal type, is conceptually and empirically flawed. The remainder of the optimization results will focus instead on the optimization of single neurons. In other words, I will produce optimized models that used only features extracted from recordings of the same neuron, and those models will putatively represent any neuron that behaves as that one did. This conveniently eliminates not only variability across neurons of the same type, but also variability across recordings sessions or labs, since by experimental necessity intracellular recordings are collected in a matter of minutes in a single lab.

Table 16 which constraints (i.e. NeuronUnit tests were used to guide this process. These constaints were different from those used in the NeuronUnit tests for two reasons: a) NeuronUnit contains very little data about some of these features and b) the single neuron data used here was collected using a family of suprathreshold stimuli that allow many complex spike-pattern features to be calculated.

Feature Name	Description
adaptation-index	Adaption Index for above $0mV$ spikes
adaptation-index2	Adaption Index for a mixture of above and below $0mV$ spikes
time-to-first-spike	The time in seconds to first spike
mean-AP-amplitude	The average in AP amplitude
spike-half-width	Spike width measured at half amplitude
AHP-depth	After Hyperpolarization Depth
minimum-voltage	Minimum voltage in V_M recording
peak-voltage	Peak voltage in V_M recording
time-to-last-spike	Time elapsed until final spike.
AHP-depth-abs	Mean after-hyperpolarization amplitude.
all-ISI-values	All Inter Spike Interval Values
voltage-base	The base voltage of V_M while undergoing stimulus. Often not the minimum V_M
min-voltage-between-spikes	The minimum voltage between spike in V_M, while neuron undergoing stimulus
Spikecount	Number of spikes observed in model, given appropriate AP

Table 16: These 14 features were identified from Van Geit (2015b) as useful for fitting single neuron models for which responses to suprathreshold stimuli were available. Unlike the features used in NeuroElectro, these features can distinguish between different temporal patterns of spiking. Special care was taken to minimize the impact of AHP depth and Minimum Voltage between spikes, as reduced models struggle to match these due to inherently impoverished dynamics.

3.4.1 Optimization of Blue Brain Project Neurons

I use two data sources for these results. First, I use a somatic current injections conducted on a rat somato-sensory hind limb neurons as part of the Blue Brain Project.

Figure 26 shows an example of an AdEx model optimized against data from such a neuron, which provided a challenging response (to somatic current injection) to fit. For example, while spike rate adaptation can be observed in this neuron, it is not monotonic: while the inter-spike-intervals (ISIs) are generally increasing, the final ISI is actually slightly shorter than the penultimate one. Nonetheless, the optimizer achieves a reasonably good match to this pattern of spikes, as well as to the resting potential and the spike threshold. It performs less well at capturing the hyperpolarization between spikes. Similar results are shown using an Izhikevich model (Figure 27). The AdEx model is somewhat better at matching spike times than the Izhikevich model, consistent with the literature (Rossant *et al.*, 2011).

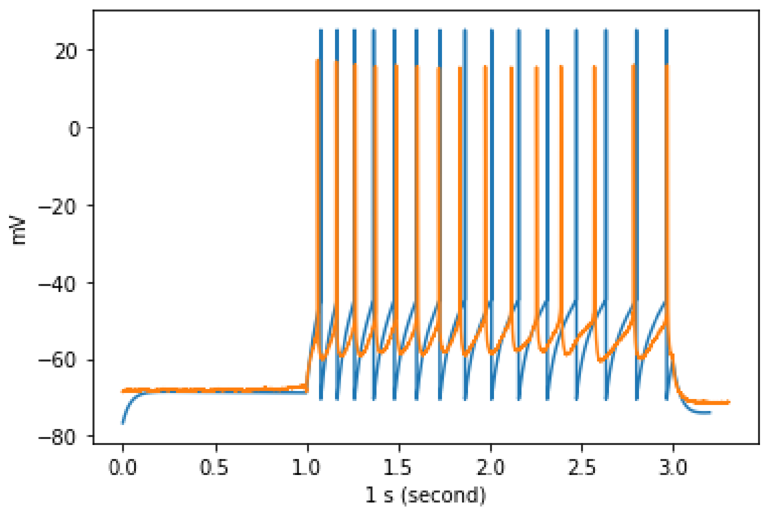


Figure 26: An Adaptive Exponential (AdEx) model was optimized to both the spike times and shapes from a single neuron in the Blue Brain Project (BBP) Microcircuit Portal data (taken from animal ID B95). The real biological neuron is the orange trace, and the simulated trace is shown in blue. While there as some “misses" in the subthreshold behavior in between spikes, and in the amplitudes of the spikes, due to fundamental limitations in the dynamics of the model, the basic pattern of spiking is successfully recapitulated.

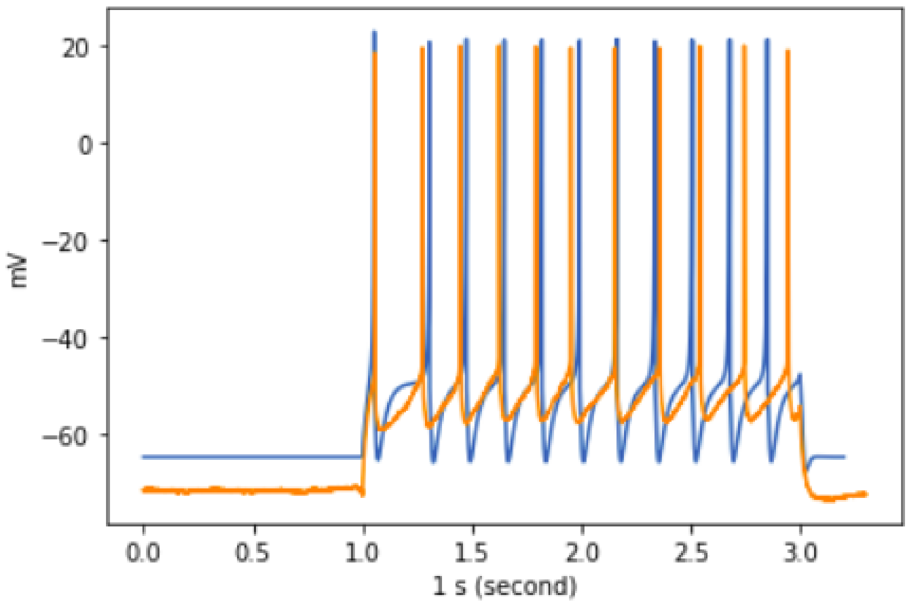


Figure 27: Similar to Fig. 26, but instead using an Izhikevich model. Here optimisation is a bit worse, but not poorer than the best out-of-sample predictions of spike timing from previous efforts.

3.4.2 Allen Cell Types Database Neurons

I then used similar data for mouse neurons taken from primary visual cortex and optimized the AdEx model (which gave the best results for the BBP data considered above). These results were improved over those using the BBP neuron data. As shown in Figures 28 and 29, optimized models of this flavor were capable of reproducing patterns of spikes and (to a better extent than using the BBP data) subthreshold dynamics. The same optimized model was also capable of reproducing the patterns of spikes in response to different amplitudes of somatic current injection (Figs 28 and 29).

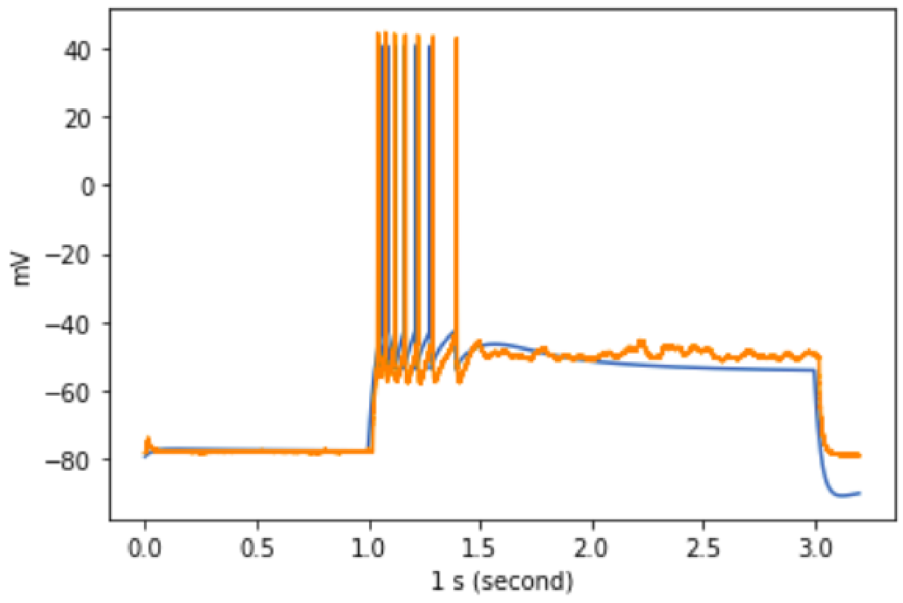


Figure 28: An AdEx Model was fitted to suprathreshold responses from Allen Cell Types cell id 476053392. The optimized model matches the spikes times of the experimental data in response to the same stimulus.

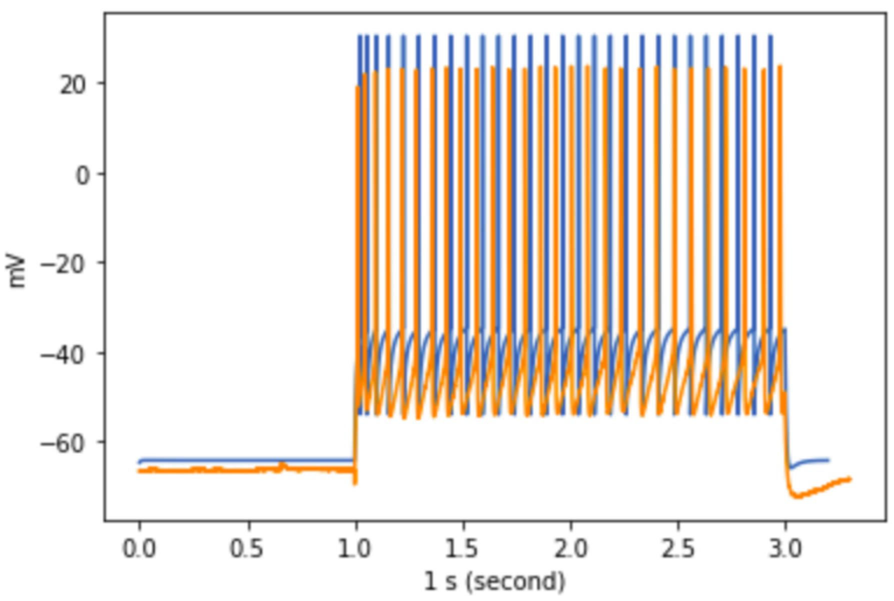


Figure 29: **Optimized AdEx Model from Allen Cell Types (B).** Same as above, but for a somatic current injection of larger magnitude.

3.4.3 Comparison to Best-case Scenarios

Since these result now concern fits to single neurons, I cannot use the χ^2 measure of agreement between the optimized model and the distribution of the data; the distribution now has only a single member per optimization. Instead, I will compare the quality of these results to best-cases scenarios from prior research. I will use the "variance explained ratio" between the experimentally recorded data and the model simulation in response to the same stimulus.

Previously, a competition was held (Naud *et al.*, 2009) inviting participants to predict spike times (in response to specific patterns of somatic current injection) for well-characterized cortical neuron. The best models in this competition were only able to predict $\sim 86\%$ of spike times (Fig. 30), and this under conditions with fluctuating somatic currents that actually produce more spike-timing regularity than square currents (Mainen and Sejnowski, 1995). Consequently, it is unlikely that *in silico* models can be expected to produce a perfect match of spike times in response to square wave current injection, tempering expectations for my optimizer. As noted in the previous optimization efforts (Druckmann *et al.*, 2007), it is probably mistaken to fit models to precise spike times; real cortical neurons produce and receive noisy currents, and likely rarely exhibit identical spike trains even in response to the same nominal stimulus. Optimization on derived features of spike timing and rate, such as statistics of distributions of ISIs, of F-I curves, is a more promising approach.

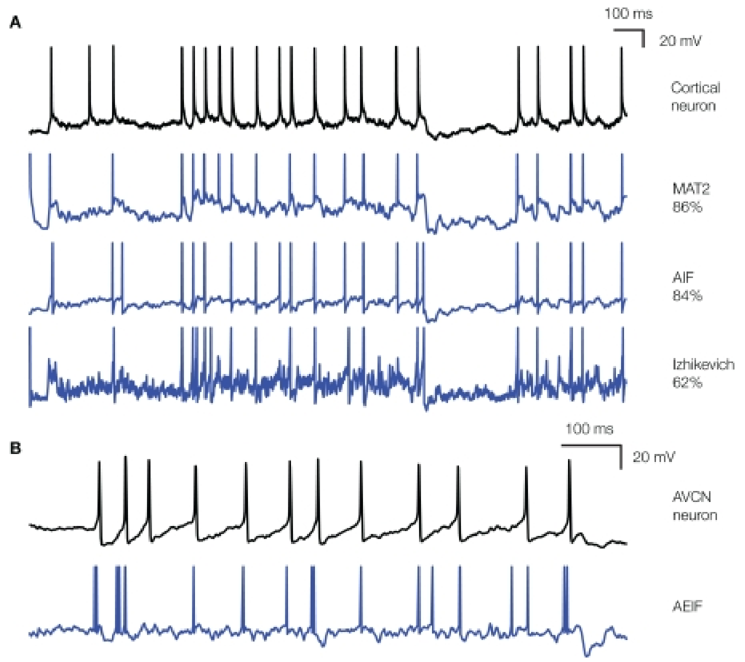


Figure 30: **Conflicts between Spike Timing and Spike Shape**. This figure from Rossant *et al.* (2011) shows how reduced models that can fit spike times typically cannot also fit spike shape. Several models are shown in this figure including the Izhikevich model used in current work and a “MAT2" model, which is a more well-equipped version of the AdEx model. The MAT2 model correctly predicts 86% of experimental spike times, but exhibits a spike threshold (a components of spike shape) that is much higher in the model compared to the experiment. The Izhikevich model only predicts 62% of spike times (this is also consistent with the results in this thesis), but it displays richer subthreshold and more accurate peri-threshold behavior than the more mercenary MAT2 model.

3.5 Published Models vs Optimized Models

Neural models and real neurons may have divergent features. Ideally, we would identify such features and modify the parameters (or form) of the models to eliminate this divergence. This divergence may arise for two reasons: A the model class was not flexible enough to simultaneously satisfy all of the constraints associated with those features, or B: the model was incorrectly fitted to constraints that are inherently conflicting, due to issues in the underlying experimental data (see results for NeuroElectro data in previous sections).

In the absence of complete knowledge about the sources of model/experiment divergence for a given model, it is unknown which features should be used to fix the problem. Modifying parameters to bring some features into alignment with experimental data may cause other features to fall out of alignment. Ideally, we would identify that subset of features which, when handed to the optimizer, would produce a model which satisfies not only that feature subset, but all other features as well.

I presented some features of membrane potential waveforms back in Section 2.5. Here I conducted a multivariate analysis analyzing hundreds of such features, asking which were useful for optimizing published neuron models to better reflect experimental data. This high dimensional feature space is then summarized by a lower-dimensional subspace of key features on which optimized models differ from their pre-optimized published counterparts, as well as from experimental data.

3.5.1 Models and Features

In order to conduct this analysis I used 972 neuron models found in NeuroML-DB (Birgiolas *et al.*, 2016) and 448 experiments on cortical neurons *in vitro* from Gouwens *et al.* (2018) (example in Fig. 31) and Markram (2006) (example in Fig. 32). Although missing data imputation was used successfully to fill out the dataset, handling features that could not be fully imputed based on the available waveforms, about half of all initial models and experiments were nonetheless excluded from a final analysis, because they did not all capable of meeting inclusion criteria. These include straightforward rules such as “does this neuron exhibit spikes?", but also requirements about which stimuli were tested, and whether these were appropriate for computing the features that I used. Ultimately, 47 features were used for the subsequent analysis (out of a total of 466 from the initial list that are technically computable in some scenarios; see the Appendix for the full list).

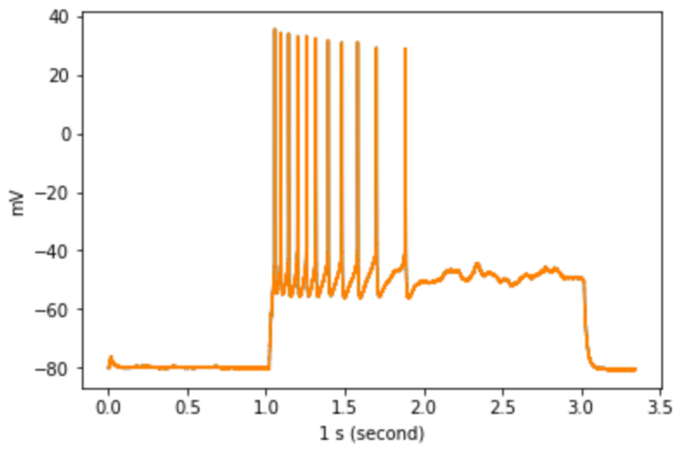


Figure 31: **Example data from the Allen Cell Types database.** Response of a cortical cell to a supra-threshold stimulus in the Allen Brain Institute Cell Types Database (Gouwens *et al.*, 2018).

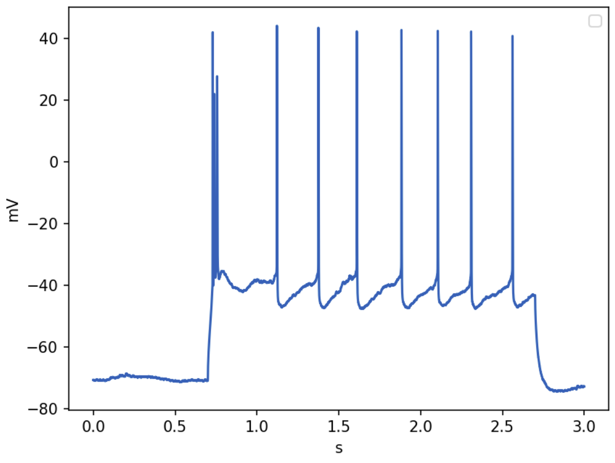


Figure 32: **Example Data From the Blue Brain Project.** Response of a cortical cell to a supra-threshold stimulus from the Blue Brain Microcircuit Portal (Toledo *et al.*, 2016).

I applied feature extraction to the raw data from these experiments, using three approaches, which I adapted to NeuronUnit: (1) EFEL, developed by the Human Brain Project and described in section 2.6.4.2, (2) the Allen Institute SDK, developed as a companion to the Cell Types Database (but not exclusively so) and (3) Druckmann *et al.* (2013), a general-purpose set of features for spike pattern analysis, as previously implemented for NeuronUnit in Birgiolas (2019). A common patterns was to apply somatic current injection at fixed multiples of rheobase, and extract features from the resulting spike patterns (Table 17).

3.5.2 Identification of Salient Features

In order to identify the electrophysiological measurements or “features" that characterized variance in the models and the experimental data, as well as their differences, I performed Sparse Principle Component Analysis (Sparse PCA) (Zou

Injection 1	Injection 2	Injection 3
1.0× Rheobase current	1.5× Rheobase current	3.0× Rheobase current

Table 17: Under the so-called three-step protocol, the rheobase current is first determined uniquely for each model or experiment, and features extracted from the response to this stimulus. Then mutiples of rheobase (×1.5 and ×3.0) are provided as stimuli, producing additional features that can be extracted using the library provided by Van Geit (2015b). In the experiments, similar protocols had been run (but using additive increments rather than multiples of rheobase). I used interpolation to identify which of these additive increments to use as a proxy for the multiples of rheobase required to extract the current features.

et al., 2006) on the ensemble. Sparse PCA, unlike conventional PCA, yields readily interpretable components, each of which includes only a handful of features, rather than an indecipherable mix of all possible features. A low-dimensional embedding of models and data can then be obtained by re-plotting their features along the principal axes suggested by these components. Sparse PCA yields an interpretable list of features, that build the principle components.

The non-zero loadings of Principle Component 1 are shown in Table 18. This principal component did not distinguish between models and experiments, instead capturing a source of variance that was common to both of them. By contrast, the second Principal Component discriminated models and experiments, indicating that it had captured some systematic difference between most published models and the experimental data for similar (but not necessarily the same) neurons. It's non-zero loadings are shown in Table 19. Models and experiments shared the same breadth of variability across the first principal component, with only slightly more variance in the experiments than in the data. Allen Cell Types Database experiments exhibited the greatest variability out of all models and experiments (Figure 34). Altogether, these two principal components were sufficient to partition models and data into three separate clusters (right panel in Figure 34). The full set of PCs are shown in the figure below. Perhaps surprisingly, models clustered tightly and varied less than experimental data–one subset of Allen Cell Types database experiments even formed their own cluster–suggesting that perhaps past modeling efforts have been insufficiently bold in accounting for the diversity of cortical neuron types.

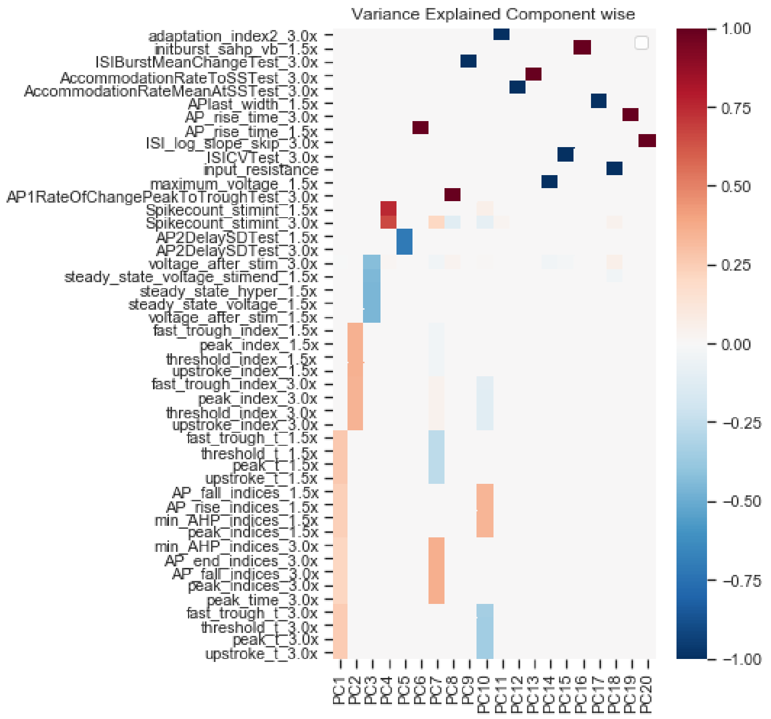


Figure 33: Sparse PCA was applied to the collection of features extracted from models and data features. Component loadings from each feature were examined. Features are organized hierarchically, such that features with similar PC loadings are near each other.

Of the 47 features considered, the features described in the table above make up the first two prinicipal components of sparse PCA decomposition, and are responsible for $\sim 50\%$ of the overall variance (left panel in Fig. 34) in the ensemble of models and data. Interestingly these features mostly belong to the Allen SDK feature extraction set, with two exceptions: *peak-indices-1.5x* and *min-AHP-indices-1.5-times* which

Feature Name	Feature Description	Extraction Library	Stimulus Strength
upstroke-t	The time of upstrokes, this is the below V_T first upward phase of AP	Allen	1.5× Rheobase
peak-t	Time(s) maximum V_M occurs	Allen	1.5× Rheobase
threshold-t	Time(s) V_T is surpassed	Allen	1.5× Rheobase
fast-trough-t	the times when when begginings of troughs were detected	Allen	1.5× Rheobase
fast-trough-t	Same as above but at 3× Rheobase	Allen	3.0× Rheobase
upstroke-t	Same as above but at 3× Rheobase	Allen	3.0× Rheobase
peak-t	Same as above but at 3× Rheobase	Allen	3.0× Rheobase
threshold-t	Same as above but at 3× Rheobase	Allen	3.0× Rheobase
peak-indices	Indexs into array where peak voltages occur	EFEL	1.5× Rheobase
min-AHP-indices	Indexs into array where minimum After Hyperpolarisation occur	EFEL	1.5× Rheobase

Table 18: I identified the features associated with principal component 1 in the sparse PCA decomposition shown in the figure below: 33. These features explained $\frac{1}{3}$ of all of the variance (across models and data together), and were mainly associated with the timing of spikes and spike-associated events. Notably, this principal component was not useful for separate models from experimental data.

Feature Name	Feature Description	Extraction Library	Stimulus Strength
fast-trough-index	Index into array when begging of trough occurs	Allen	1.5× Rheobase
peak-index-1.5x	Index into array when peaks occurs	Allen	1.5× Rheobase
upstroke-index-1.5x	index into array of detection of first upward phase of AP	Allen	1.5× Rheobase
threshold-index-1.5x	Description	Allen	1.5× Rheobase
fast-trough-time	The time when a trough is commenced	Allen	1.5× Rheobase
fast-trough-index	Indexs into array when the start of a trough is entered	Allen	3.0× Rheobase
peak-index	indexs into array when voltage peak(s) occur	Allen	3.0× Rheobase
upstroke-index	Index into array when first upward phase of a spike commences	Allen	3.0× Rheobase
threshold-index	Index into array when threshold(s) are surpassed	Allen	3.0× Rheobase

Table 19: The second principal component from the sparse PCA of Figure 33 contained features which describe spike shape. Here, “index" describes the location in the array of the extracted spike where an event occurs, thus it is relative to the timing of the spike itself, not to the timing of the stimulus. These features contributed to the separation of models from experimental data.

belong to EFEL. This suggests that the Allen SDK might be nearly sufficient for both characterizing and discriminating between existing models and data.

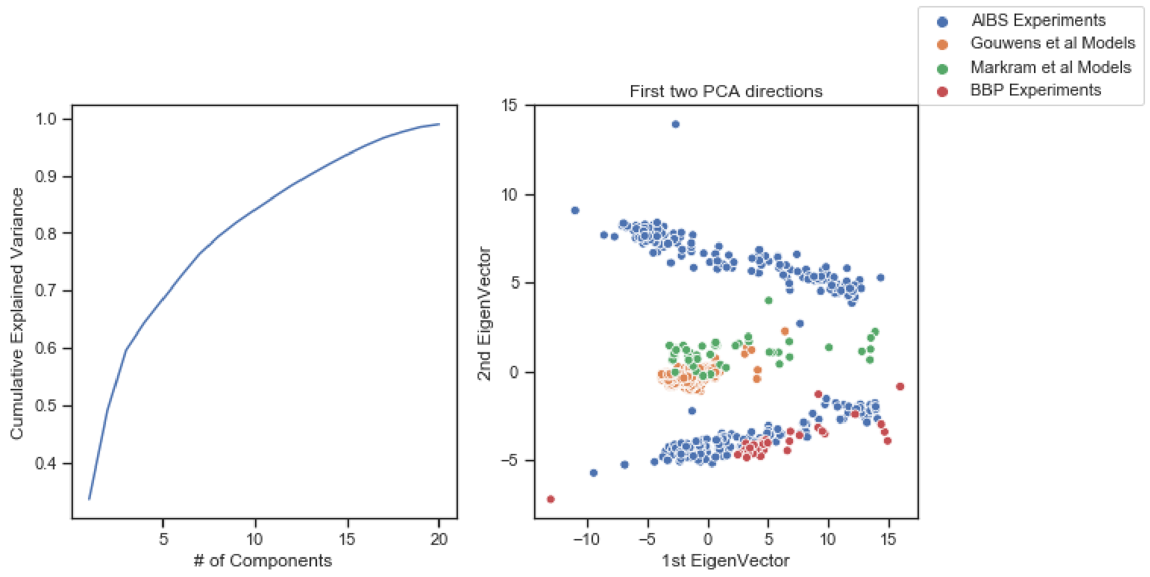


Figure 34: **Sparce PCA Clusters and Variance Explained.** Left panel: Sparse PCA could explain $\sim 95\%$ of the variance in the ensemble of model and experimental data features using ~ 15 components, but the first two were sufficient to explain $\sim 50\%$ of the variance. Right panel: A cluster analysis revealed revealed three major clusters of feature values. Models and experimental data could be distinguished by PC 2. Individual dots refer to experiments from the Allen Institute Cell Types database (blue), model published by the Allen Institute (orange), experiments published by the Blue Brain Project (red), and models published by the Blue Brain Project (orange).

A small majority of features identified by sparse PCA are derived from the $1.5\times$ rheobase stimulus, slightly fewer are from the $3.0\times$ rheobase stimulus. This may be because spike time and spike count variability in the most sensitive (highest slope) part of the FI curve, slightly above rheobase, is greater than under the highest current injections, where the neuron is much close to responding as quickly as possible (i.e. near its absolute refractory period).

3.5.3 What Distinguishes Published Models from Experimental Data?

Disagreement between models and real neurons may reflect limitations of model design and can be investigated by probing the features identified by a classifiers trained to distinguish these two populations. Figure 36 shows the distributions of two features for models (red) and for experimental data (blue). Rather than the distributions being separable, the major observation is that the distribution of these features across published models is simply much narrower. The diversity of feature values in real neurons is simply not being adequately captured by published models.

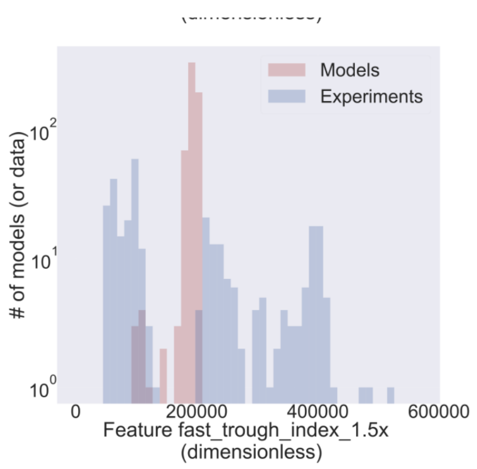


Figure 35: **Comparison of Model and Experiment Feature Distributions (A).** The variance of single features across biological neurons (blue) is much greater than that observed in published models. The feature shown here (fast-trough-index) reflects the latencies of the post-spike trough relative to its peak. Biological neurons clearly exhibit a wide range of such latencies, where as models exhibit a narrow range possibly limited by their underlying dynamics.

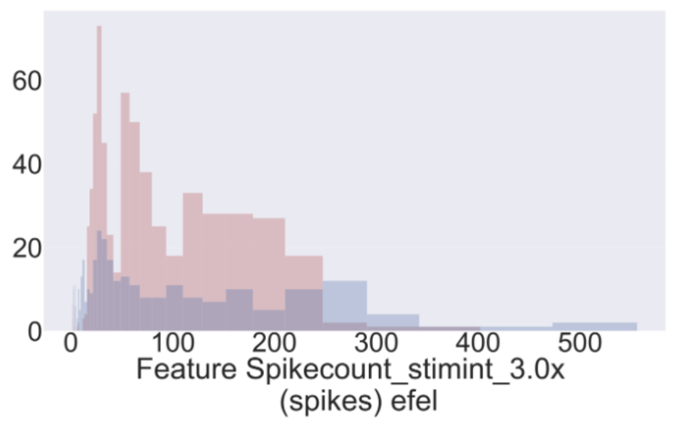
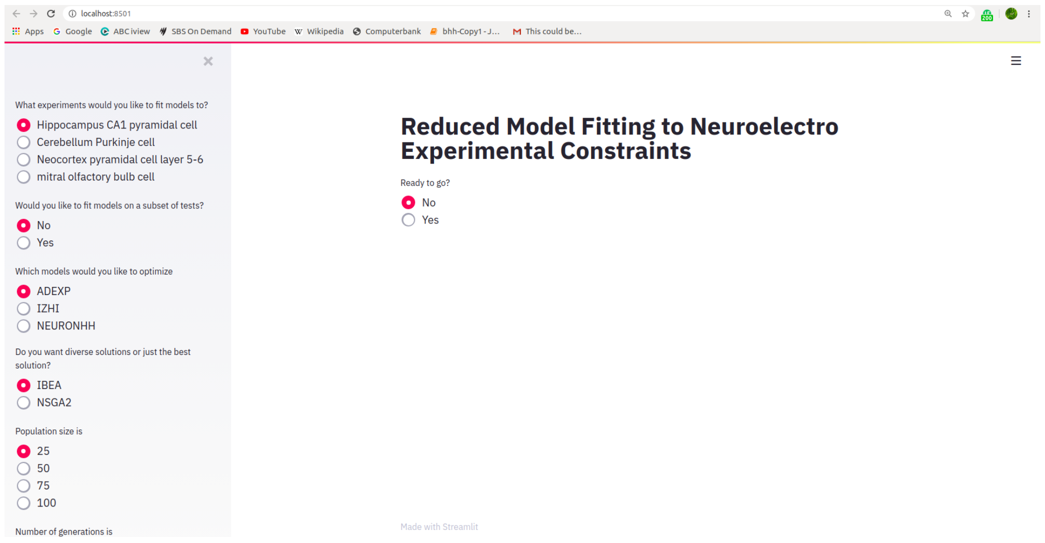


Figure 36: **Comparison of Model and Experiment Feature Distributions (B).** Unlike Figure 35, some features showed comparably wide distributions in model and biological neurons. The distribution of spike counts is similarly skewed in both cases. This suggests that, in contrast to spike shapes, reduced models are able to reflect the diversity of overall firing rate statistics seen in biology.

3.6 A Web Application for Optimization and Visualization

In order to make this optimization framework available to researchers, students, and anyone who would prefer not to work with the code itself, I developed a web application that implements its major components. I developed this tool using the python library “Streamlit" (Steamlit-Team, 2020), which facilitates the transformation of algorithms and visualizations to a web application. Fortunately for the developer, coding in Streamlit web framework requires no handling of html, javascript, or other distractions. Instead, Streamlit provides tools for converting Python code directly into interactive tables and plots.

This web application is initialize with a choice of three models and four data-sets (Figure 37), although it would be straight-forward to add additional ones.



What experiments would you like to fit models to?
[✓ Hippocampus CA1 pyramidal cell]
Cerebellum Purkinje cell
Neocortex pyramidal cell layer 5-6
mitral olfactory bulb cell

Would you like to fit models on a subset of tests?
[✓ No]
Yes

Which models would you like to optimize
[✓ ADEXP]
IZHI
NEURONHH

Do you want diverse solutions or just the best solution?
[✓ IBEA]
NSGA2

Population size is
[✓ 25]
50
75
100

Number of generations is

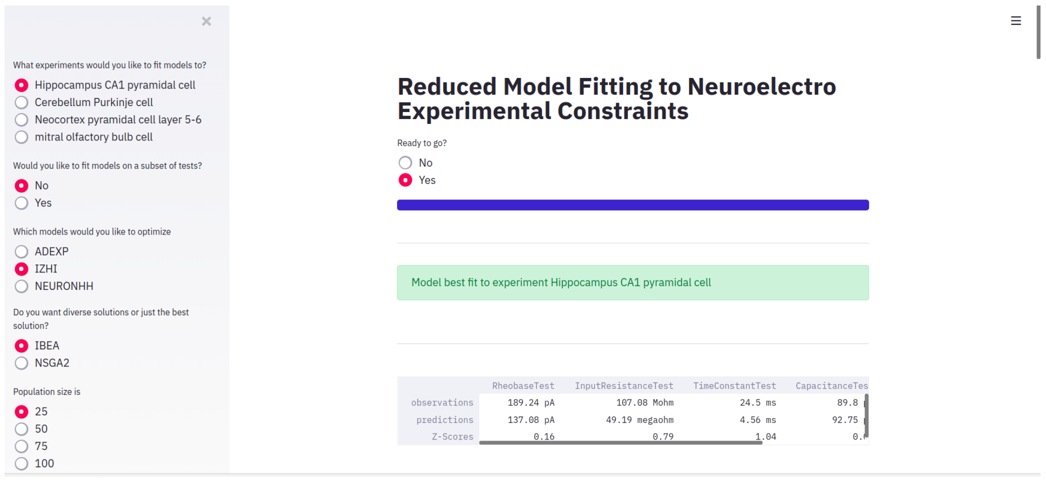
Reduced Model Fitting to Neuroelectro Experimental Constraints

Ready to go?
[✓ No]
Yes

Made with Streamlit

Figure 37: **A Web Application for Optimization: Part A**. I created a web application to guide a user through the entire optimization process for reduced neuron models. The side-pane of the web application provides users with a choice of three models, and four data sets that can be used to fit data.

The user is given a choice of very simple optimisation choices including what model type to use and the experimental data that will be used to guide the optimizer (Figure 38).


What experiments would you like to fit models to?

[✓ Hippocampus CA1 pyramidal cell] Cerebellum Purkinje cell Neocortex pyramidal cell layer 5-6 mitral olfactory bulb cell

Would you like to fit models on a subset of tests?

[✓ No] Yes

Which models would you like to optimize

ADEXP [✓ IZHI] NEURONHH

Do you want diverse solutions or just the best solution?

[✓ IBEA] NSGA2

Population size is

[✓ 25] 50 75 100

Reduced Model Fitting to Neuroelectro Experimental Constraints

Ready to go?

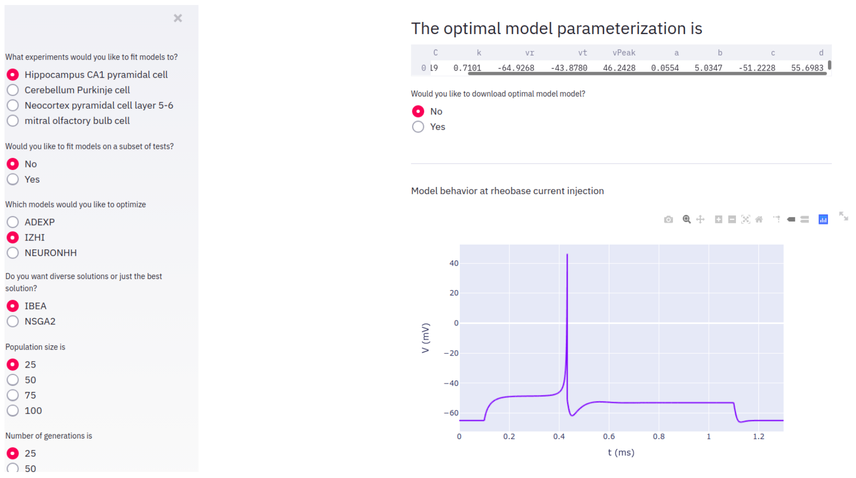
No [✓ Yes]

Model best fit to experiment Hippocampus CA1 pyramidal cell

	RheobaseTest	InputResistanceTest	TimeConstantTest	CapacitanceTes
observations	189.24 pA	107.08 Mohm	24.5 ms	89.8
predictions	137.08 pA	49.19 megaohm	4.56 ms	92.75
Z-Scores	0.16	0.79	1.04	0.

Figure 38: **A Web Application for Optimization: Part B**. Once the optimizer (running on the web server) has completed finished optimizing a model against NeuroElectro data, a table showing model/data agreement is shown to the user. The user can also optionally scroll through a visualisation of model fitness metrics, such as the χ^2 test value. They can also inspect interactive model waveforms, and download a serialized representation ready to run locally in Python.

When optimization is complete the user is shown the parameters of the optimized model as well as key membrane potential traces (i.e. response to the rheobase current injection) for inspection (Figure 39).


What experiments would you like to fit models to?
[✓ Hippocampus CA1 pyramidal cell] Cerebellum Purkinje cell Neocortex pyramidal cell layer 5-6 mitral olfactory bulb cell

Would you like to fit models on a subset of tests?
[✓ No] Yes

Which models would you like to optimize
ADEXP [✓ IZHI] NEURONHH

Do you want diverse solutions or just the best solution?
[✓ IBEA] NSGA2

Population size is
[✓ 25] 50 75 100

Number of generations is
[✓ 25] 50

The optimal model parameterization is

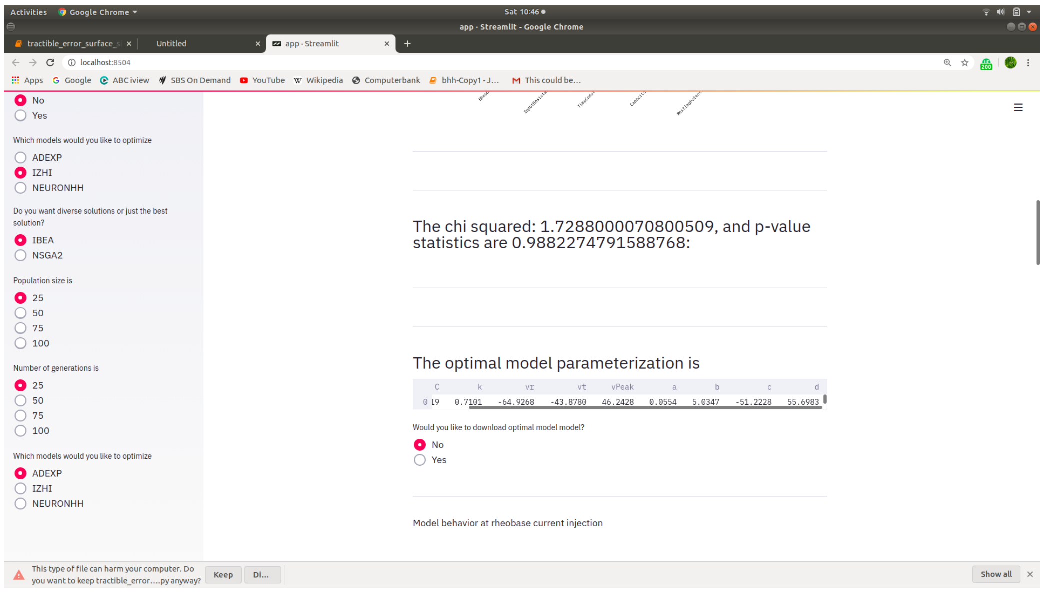
	C	k	vr	vt	vPeak	a	b	c	d
0	l9	0.7101	-64.9268	-43.8780	46.2428	0.0554	5.0347	-51.2228	55.6983

Would you like to download optimal model model?
[✓ No] Yes

Model behavior at rheobase current injection

Figure 39: **A Web Application for Optimization: Part C**. The application shows an interactive visualisation of the optimized model neuron firing in response to (its computed) rheobase current (shown above), and also under a hyperpolarizing current used to compute other features (not shown above).
.

Finally, the user is shown the χ^2 statistic (and p-value) for the optimized model, and provided with a link is provided to download these parameters for their own use (Figure 40). Future work will allow the user will be able to download a NeuroML file of the optimized model.



[✓ No] Yes

Which models would you like to optimize

ADEXP [✓ IZHI] NEURONHH

Do you want diverse solutions or just the best solution?

[✓ IBEA] NSGA2

Population size is

[✓ 25] 50 75 100

Number of generations is

[✓ 25] 50 75 100

Which models would you like to optimize

[✓ ADEXP] IZHI NEURONHH

The chi squared: 1.7288000070800509, and p-value statistics are 0.9882274791588768:

The optimal model parameterization is

	C	k	vr	vt	vPeak	a	b	c	d
0	19	0.7101	-64.9268	-43.8780	46.2428	0.0554	5.0347	-51.2228	55.6983

Would you like to download optimal model model?

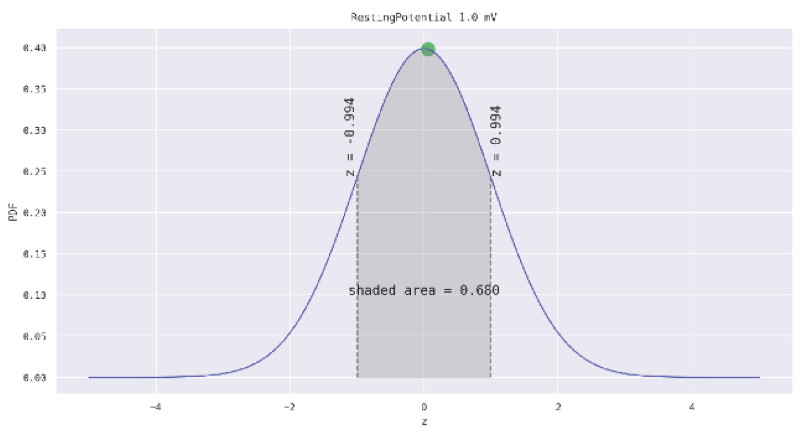
[✓ No] Yes

Model behavior at rheobase current injection

This type of file can harm your computer. Do you want to keep tractible_error....py anyway?

Figure 40: **A Web Application for Optimization: Part D**. The application provides a prompt to download an optimized version of the model. In future work this feature will supportthe download of a NeuroML version of the model for use on alternative simulators.
.

Because some of the details of the scoring associated with optimization are unfamiliar to most people, another visualization is provided (Figure 41 which helps the user visualize the meaning of a Z-score in an optimization context, and displays this information for each feature used to constrain the optimization. Additional information about the stimuli used and the time taken to run the simulations are also provided.



The IZHI model is the fastest. Over three different current injection strengths mean Model speed was 3.36 ms, to find unkown rheoase value it takes 0.7 seconds

Do you want to try again on a different model to see how that would look?

Want to keep model on screen and compare to new model?

[✓ No] Yes

Figure 41: **A Web Application for Optimization: Part E**. The application describes additional statistics about the computed features, including an explanation of the Z-score and how it was computed for a given feature. It also provides information about model performance.

Chapter 4

DISCUSSION

In this thesis I have described a number of methods that I developed and implemented to optimize reduced neuron models against experimental data collected from real neurons. I then verified that this optimizer works in the sense of being able to recover ground truth models given only simulated data from those models. I showed that successful optimization depends upon certain characteristics of the experimental data and the features extracted from it, and identified a flaw in the approach of optimizing against aggregated data from neural populations. I demonstrated that the optimizer does an adequate job at fitting reduced neuron models to real experimental data from single neurons, and that certain models (and cell types) are easier to optimize than others. I showed that published models differ from the experimental data they ought to be explaining, on which features they differ most, and I show proto-typed analysis pathway that is capable of demonstrating if and when optimized models can better match experimental data, compared to other published models. Finally, I presented a tool to bring this optimization framework to the masses.

Efficient optimization of reduced neuron models is significantly non-trivial, as an alternative, obtaining merely satisfactory biologically plausible models is several degrees easier. Below I provide a number of examples of optimization pitfalls which were not obvious at the start of my research, but which I discovered during my research, and which were on ongoing source of confusion, not just for me, but for my whole research team. I believe that it is important to share these conceptual traps with my readership, including those who may wish to continue such optimization work.

4.1 What is Required for Successful Optimization?

For optimization to both succeed and be useful, several criteria must be met (Van Geit *et al.*, 2007):

- Relevance: The objective function should reflect fundamental and important properties of the data that a good model would reproduce.
- Speed: The objective function should be fast to calculate, since typically a large number (potentially millions) of evaluations are performed during the search, many of which may require re-simulation of the model.
- Efficient Convergence: The solution space should be as continuous and convex as possible, so that the search algorithm can rapidly converge to a global optimum.

4.1.1 Relevance of the Objective Function

4.1.1.1 Source of Data Constraining the Objective Function

Due to the abundance and diversity of data available through the NeuroElectro Project Tripathy *et al.* (2014), my initial work relied heavily on that data source. However, that data is enriched in passive membrane properties as well as the details of individual action potential waveforms. But what distinguishes, say, a Layer 5 pyramidal cell in motor cortex from a Layer 5 pyramidal cell in visual cortex, in terms of the computational principles that systems neuroscientists might care about (e.g. decoding, information, etc.) is more likely to be reflected in the patterns of spiking and not the dynamics of single spikes or subthreshold behavior. Furthermore, most reduced models are not implemented in way that allows for richly detailed action potential waveforms to be reproduced.

Together, this implies that reduced models optimized against data exclusively from NeuroElecto may be the worst of both worlds: they fail to capture the sub-millisecond dynamics of the action potential encoded in that data, while also failing to exhibit any of the suprathreshold dynamics (e.g. types of bursting) that distinguish one cell type from another, in the mind of a systems neuroscientist. This problem is mitigated by including complementary data sources (like the Allen Cell Types database) that can address these dynamics. Future optimization efforts should take care to identify the data sources that capture the dimensions of the experimental data along which meaningful differences between cell types can be resolved, and which models are rich enough to express.

4.1.1.2 Number of Components in the Objective Function

Theoretically every additional component included in the objective function–every constraint derived from some experimental feature–should make the error surface a better answer to the question, “Is this a realistic model of the neuron of interest?". Although they might increase the computation time required per objective function evaluation, such additional components may rapidly exclude large volumes of parameter space from unnecessary exploration. For example, when modeling a bursting neuron, including burst-statistics as one of the features under evaluation will quickly exclude non-bursty regions of parameter space from consideration.

And yet a successful optimization recipe should not naively involve the use of all available computable features. Some features might be biologically irrelevant, or impossible for some model class to reproduce, or have extremely discontinuous error surfaces. This makes the task of neuronal model optimization, much more supervised and much less automatic compared to many other machine learning paradigms –careful human guidance is needed to curate the appropriate features for the task.

4.1.2 Speed of Optimization

4.1.2.1 Speed of the Objective Function

A large fraction of the compute cycles spent evaluating each parameter set are expended on identifying the rheobase current, thus unlocking the calculation of several subsequent measurements. Fortunately, for slower model implementations this can be sped up significantly through parallelism, as described in Section 2.4. In some applications, parallel code scheduling is incompatible with with the just-in-time (JIT) compilation approach described in Section 2.3.2.4, so parallelism sometimes trade off against raw model simulation speed.

4.1.2.2 Speed of Parallel Exploration

In this work it was sometimes memory pressure and not clock speed that produced the major bottleneck. This was especially true when mining features from pre-existing databases. In these contexts I made use of a delayed iterator provided by the Dask library Dask (Rocklin, 2015) to stream very large amounts of data in memory-friendly chunks as needed. Delayed evaluation using the Dask library also resolved conflicts Additionally delayed evaluation caused fewer conflicts with JIT code.

4.2 Sensitivity to Error Surface Quality

In the spirit of the list above provided by Van Geit *et al.* (2007), the next item should be called “Efficient Convergence", but this efficiency really comes down to the nature of the error surface created by the choice of models, tests, and experimental data.

4.2.1 Objective Function Dimensionality vs Model Parameter Dimensionality

If a given class of models is capable of describing the behavior of a given real neuron, then the number of independent, reliable tests used to generate the objective function can be as large as desired; more tests just provide more information to quickly rule out irrelevant regions of parameter space. On the other hand if a model class is only capable of describing a fraction of the real neuron’s behavior, too many distinct tests will result in an optimization problem that can never be fully satisfied. Since “all models are wrong, but some are useful" (Box, 1976), let us imagine that we usually in the latter case.

4.2.2 When Does Genetic Optimization Get Stuck?

Genetic algorithms are known as derivative-free optimizers, since they do not follow any gradient down the error surface, or even know of the existence of such a gradient. Chromosomes only survive and reproduce differentially according to their location on the error surface. Thus, genetic optimization is never truly "stuck" inside a local minima on the error surface, as mutation or crossover can always produce new chromosomes outside the basin of attraction. Despite this robustness, just like in gradient descent, genetic algorithms can only be guided by information in the error surface. When the objective function has a low-dimensionality, for example when it is based upon tests that mostly compute small variations on the same small number of features of the simulation output, it may not provide enough information to distinguish one location in parameter space from another one close by, even though the first may be closer to the optimal solution than the second. In other words, many regions of parameter space may be locally flat at a mesoscopic scale, and local minima at a microscopic scale may thus be difficult to escape. No lower error solution may be available within a reasonable distance (in parameter space) from the current one. A variety of potential error surfaces are presented in Figure 42.

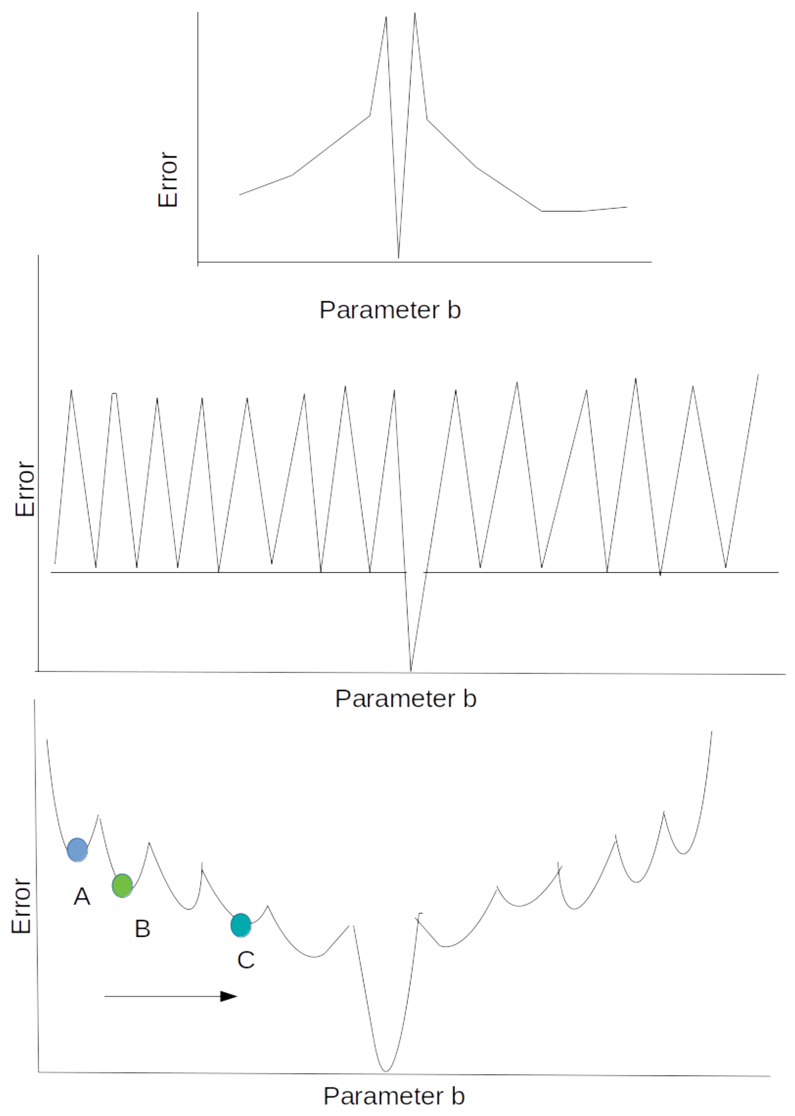


Figure 42: **Challenging Error Surfaces**. Fitness here should be relabeled "Error". The top panel shows an error surface where an optimizer is unlikely to identify the global minimum error. Any exploration of the region leading towards that minimum is likely to be abandoned prematurely. Only an extremely lucky set of initial chromosomes or a random mutation might result in exploration of the region immediately around the global minimum. The middle panel is more hopeful, showing an error surface that does not actively block the global minimum from being explored. However, the surface is still mostly uninformative. The bottom panel shows a cross-section of Rastrigrin's function (described in the Introduction). Despite its hype, this is really the least challenging of the three error surfaces shown here, as there as at least long-range structure to the error surface that a genetic algorithm can exploit.

4.2.3 Defects in the Error Surface

The real error surfaces that guide optimization here have “defects", for example discontinuities (due perhaps to bifurcations in the underlying dynamical system, e.g. from spiking to non-spiking) or to deep local minima, that make optimization challenging. I coined the term “corrugated" to describe surfaces low amplitude oscillatory disturbances across the error surface.

Some optimization techniques require a perfectly convex error surface to converge. Genetic algorithms are more tolerant, up to a point, but an extremely high dimensional optimization problem with a large number of optima can still be intractable. So which error surface defects are truly harmful? This largely comes down to scale. The corrugation observed in the error surfaces here was typically on a much smaller scale than the long-range structure that guides optimization. Consider something analogous to a signal-to-noise ratio (SNR), describing the information that guides optimization vs the wrinkles that impede it. When SNR is larger, defects are less consequential. Figure 43 gives an example of an error surface that, while not totally continuous or convex, is nonetheless easily handled by the optimizer.

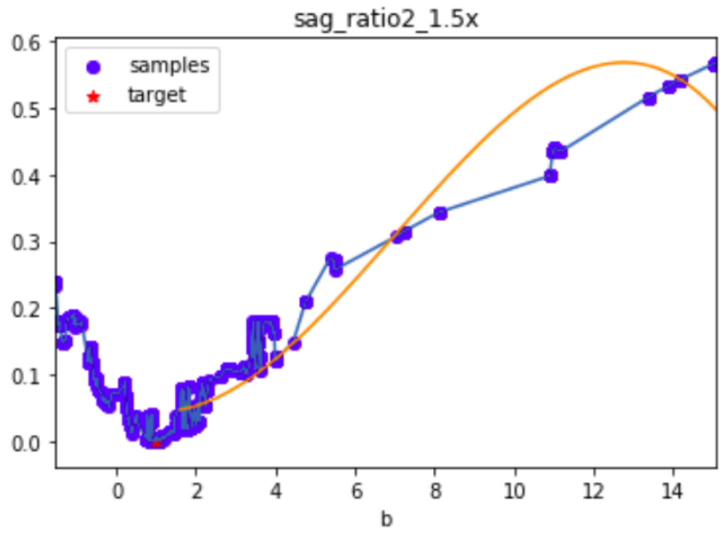


Figure 43: **A Non-convex but Manageable Error Surface**. The vertical axis shows error (model-data disagreement) for “sag-ratio" feature computed at 1.5 × rheobase as the value fo the b parameter in the Izhikevich model is slowly varied. The blue dots show the actual error values, and the orange curve shows a low-order polynomoial regression fit. While the fit is clearly not perfect, the fact that the error surface can be approximated by a low-order polynomial suggests that it will not be difficult to find the minimum (red dot).

I observed that the NSGA2 selection algorithm was particular vulnerable to defects in the error surface, and required a higher SNR to obtain optimal solutions. NSGA2 is a fundamentally conservative and short-range approach to evolution, so it may simply lack the drive to escape problematic regions of the error surface. Unlike NSGA2, IBEA was less sensitive to such defects, and required a lower SNR to converge rapidly.

However, in any algorithm as the optimum is approached, the rate of mutation must slow down to enable efficient short-range exploitation of the peri-optimum region. one should not expect to see evidence of efficient-learning. In later phase of learning when the optimizer will be more sensitive to small amplitude corrugations, which become large relative to the smaller improvements of error.

The multiobjective function is derived from the objective functions associated with each feature used in optimization. As such, the multiobjective function will be more corrugated if more of the component objective functions are corrugated. Consequently, optimization is more likely to converge if the number of corrguated components is kept to a minimum. In practice, I observed satisfactory solutions even when only $> \frac{1}{2}$ total number of objectives had no corrugation. For example, in a four-objective problem, if the 4th objective is uninformative, but not actively misleading, inclusion of that 4th objective may only slow down the speed of optimizer convergence, but not actually change the final outcome. By contrast, if that $4th$ objective is actively misleading, the optimizer will likely find a satisfactory (but non-optimal) solution by compromising with the dominant 3 objective functions.

4.2.4 Contingent Discontinuities

Some tests used to compute the objective function may depend on the results of other tests. They may depend on the measured value of one feature, for example, a test of the action potential width at half-height depends on the height measured from threshold which depends on the threshold. Or they may depend on a stimulus parameter derived from a previous test; for example, computing the first inter-spike interval (ISI) at 1.5x rheobase first requires computing rheobase, and then multiplying the rheobase value by 1.5 to generate the stimulus for the ISI test. Such an ISI test–and its results–is thus “contingent" on the results of the rheobase test. This has confounding implications. Suppose that as some model parameter X is increased, the cell becomes more excitable. All things being equal, more excitability would be associated with a lower rheobase, and with a narrower first inter-spike interval at a fixed current. But because the rheobase determines the value of the actual current injected in the ISI test (i.e. the ISI test is contingent), the ISI could go up or down; it would go down if the direct effect of greater excitability associated with increased X dominates; it would go up if the indirect effect of a smaller current injection dominates. In fact, it is impossible (or at least impractical) to predict which of these will "win", and the resulting error surface for the ISI test becomes extremely corrugated, as small increments in X cause increases and then decreases in the error of the objective function, with no discernible pattern.

4.2.4.1 Examples of Contingent Discontinuities

Figure 44 provides a concrete example of the discontinuous error surface that results from such contingencies.

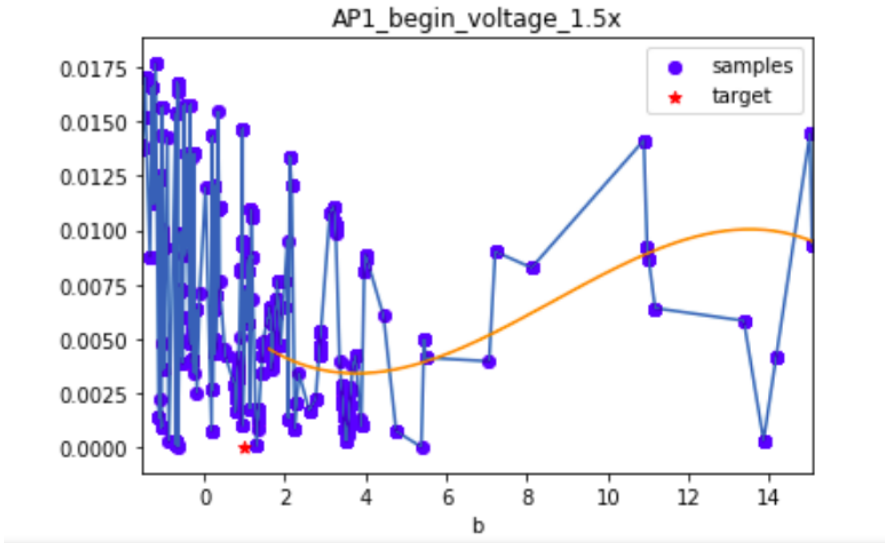


Figure 44: **A Non-convex and Unmanageable Error Surface**. Similar to Figure 43, but showing the error in another feature (the membrane potential at the start of the AP waveform) as the same parameter is varied. This error surface is extremely corrugated, the polynomial fit has no hope of approximating what is going on for $b < 6$, and the optimizer stands little change of finding the global minimum, if such a minimum is even meaningful here.

The problem is even more extreme when the contingent test can produce missing values. For example, an ISI test depends on their being an inter-spike interval to measure, i.e. it requires a second spike to be produced. If there is no second spike, this test will emit a missing value. Thus as X is increased, the error surface associated with the ISI test will be pocked with missing values every time the underlying change in excitability is offset too much by the ensuing change in rheobase-derived injected current. An example of this even more challenging case is given in Figure 45.

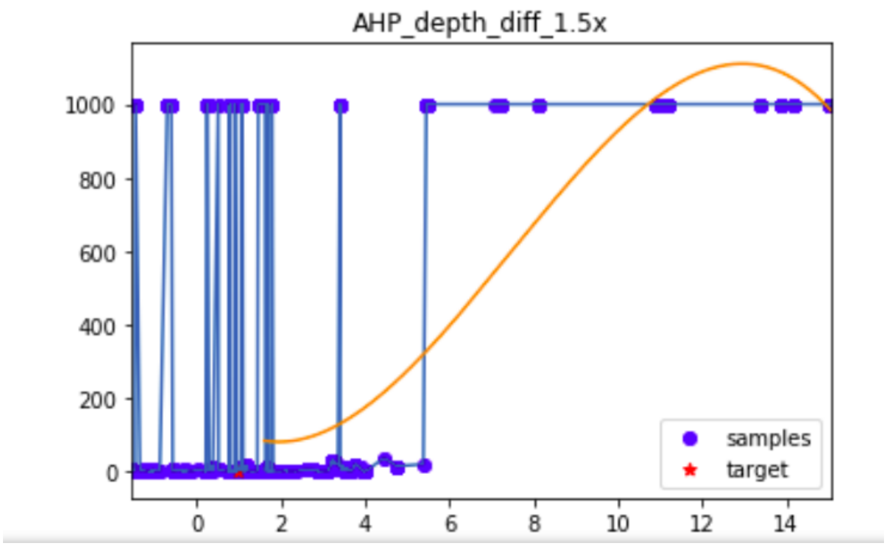


Figure 45: **Another Non-convex and Unmanageable Error Surface**. Similar to Figure 44, but now for differences in the depth of the after-hyperpolarization across repeated spikes. This feature is actually uncomputable for some values of the parameter b varied along the x-axis, because as this parameter changes, the rheobase changes, and the number of spikes observed at the rheobase varies between 1 and 2. When the number of spikes is 1, any parameters that describes differences across spikes is undefined. Because the optimizer cannot work with missing data, a very larger error (1000) is imputed. However, this means that it is nearly impossible to identify the global minimum, because many promising chromosomes mutate onto the peaks of the error surface and do not make it into subsequent generations. A genetic algorithm sees the region $b < 6$ as essentially random.

4.2.4.2 Causes of Contingent Discontinuities

And an error surface plagued with too many missing values is essentially unusable. These contingent discontinuities present a major problem to the logic of contingent testing that underlies most of the optimization presented here. I verified that the reasoning above was matched by the observed changes in simulated behavior in response to changes in model parameters. I found that slowly varying a single model parameter in a reduce model can cause two extracted features to vary inconsistent ways (Figures 46 and 47).

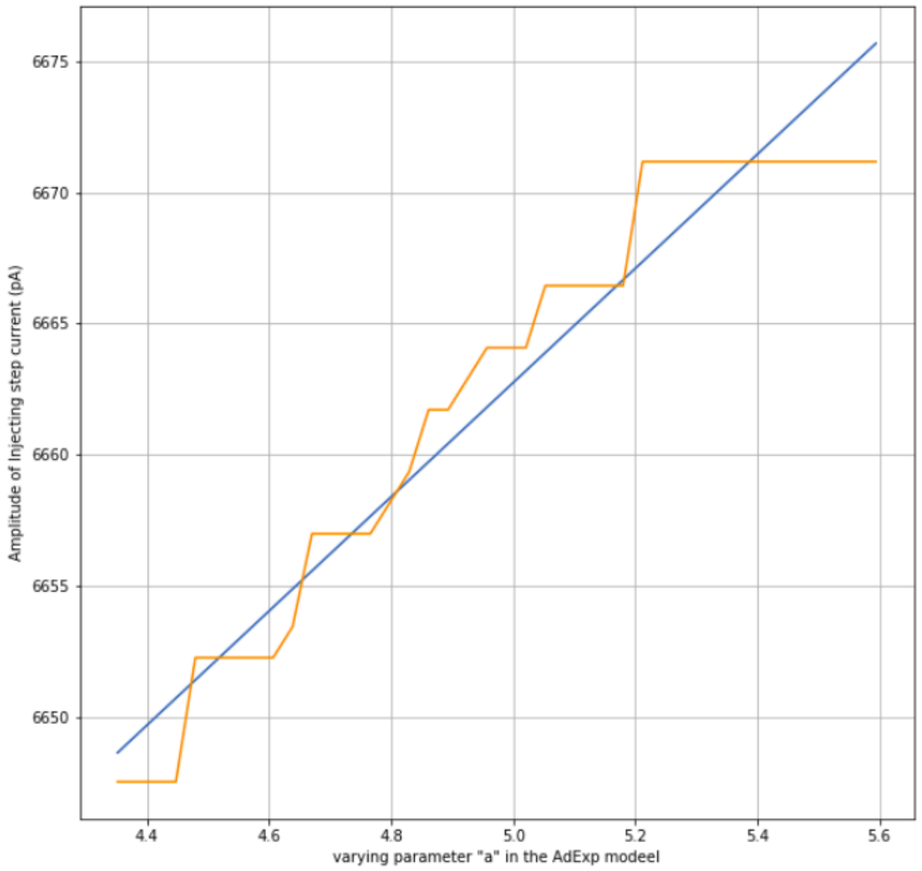


Figure 46: **The Causes of Corrugation in Error Surfaces: Part A**. I identified the main cause of corrugated error surfaces by re-calculating feature values as single parameters were varied. Here, I vary parameter a in the AdEx model. The orange trace shows the computed rheobase current as this parameter is varied. Note that it grows in a step-like manner, not according to the smooth linear fit (blue). During periods when the rheobase is not increasing (but the parameter value is), the parameter may cause some other feature, computed at the current that causes 14 spikes to fire to vary in one direction. Once the 14 spike current jumps, the stimulus used to compute the feature has changed, so that same feature may vary in the other direction. Consequently, a computed feature-value may zig-zag, rather than exhibiting a smooth change, as a parameter is varied.

In Figure 47, I show a more concrete example of the same phenomena.

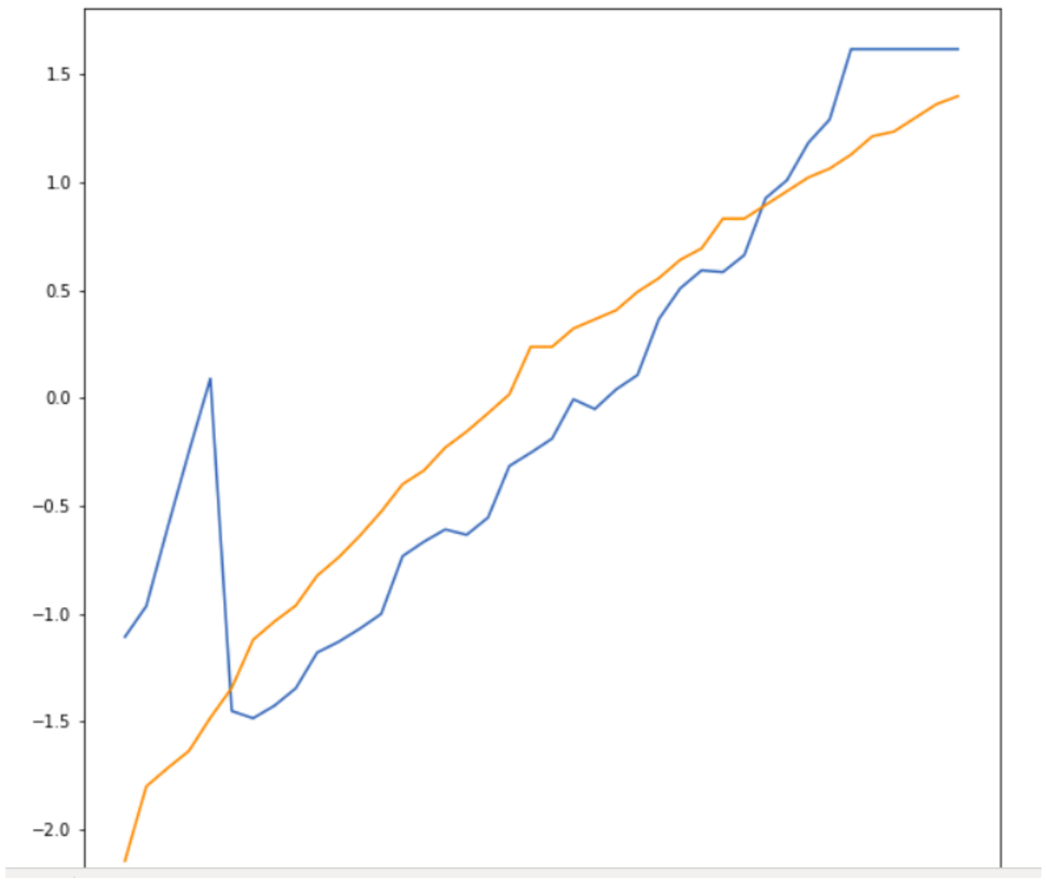


Figure 47: **The Causes of Corrugation in Error Surfaces: Part B**. Using an Izhikevich model, I vary a single odel parameter (a) as in Figure 46, and plot the rheobase current (orange, normalized to its mean value) but also compute the value of the spike threshold (blue, also normalized). Note the non-linear behavior of the response of this feature value to the change in the model parameter. This non-linear behavior is mainly driven by a change in the amplitude of the injected current used to evoke it (the rheobase current), and not to underlying non-linearity in the location of the spike threshold for a fixed current.

This can also be visualized in the waveforms themselves. In Figure 48 the location (in time) of the threshold relative to the peak of the action potential varies in unpredictable ways as a single parameter of the model is increased.

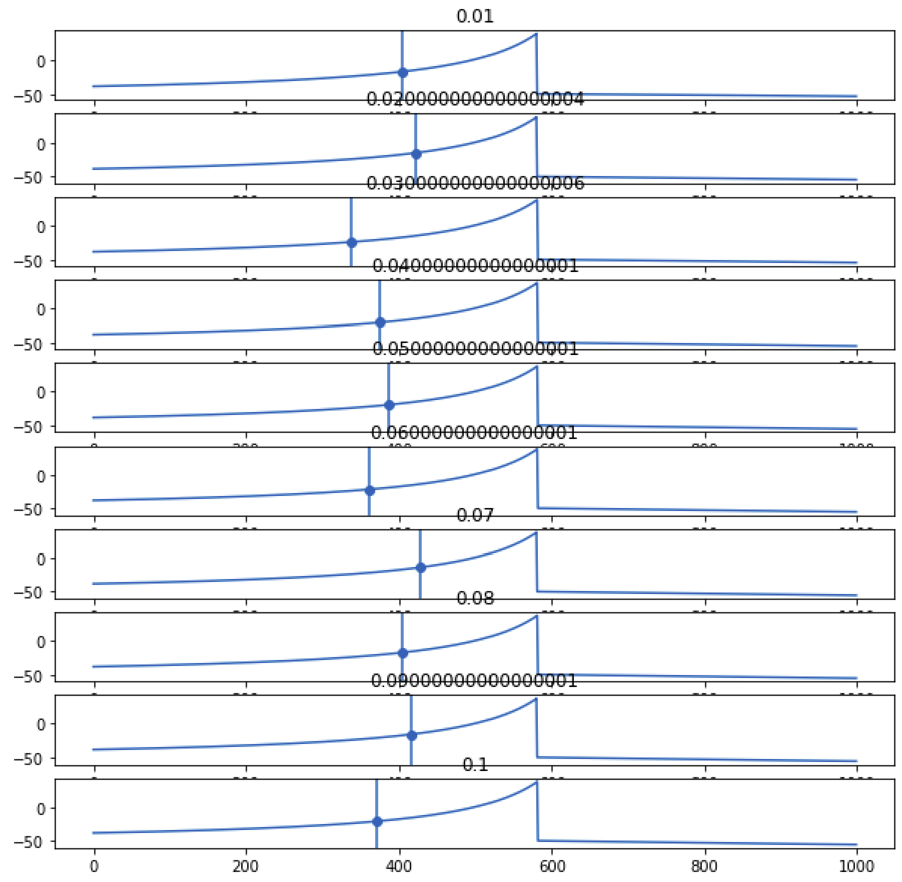


Figure 48: **The Causes of Corrugation in Error Surfaces: Part C**. In order to see this effect directly in the responses themselves, I plot the action potential waveforms for the Izhikevich model as the value of parameter a increases in small steps from the bottom panel to the top panel. At each value of this parameter, rheobase is computed and the waveform of the spike evoked at rheobase is extracted. Each such spike is aligned across panel, so that differences in the lead up to that spike can be examined. The spike threshold, identified by the moment when the slope of the membrane potential reaches a target value, is shown with each blue dot, and the time that the threshold is reached is indicated by each vertical line. It is clear that while the time of the threshold changes as a is varied, it does not do so systematically. Therefore the resulting error surface will not be useful for optimization. This demonstrates that making features contingent upon the responses to the rheobase current results in major challenges to optimizaton.

4.2.4.3 Overcoming Contingent Discontinuities

With sufficient care taken to avoid too many corrugations in the error surface, optimization can still be viable. However, this may mean discarding otherwise useful features that could in principle distinguish between competing regions of parameter space. An alternative approach is to discard the rheobase entirely as a contingency, making tests depend not on the rheobase value obtained from each parameterization of the model, but on the rheobase observed in the experimental data itself. In other words, if the rheobase of the biological neuron is 100 pA, then the 1.5$\times$ rheobase ISI test should be performed with a current injection of 150 pA, even if the rheobase of the current model parameterization is some entirely different value.

Another approach is to dispense with the rheobase entirely, and simply test using a fixed set of current amplitudes that span the suprathreshold portion of the F-I curve, e.g. 200pA, 350pA, and 500 pA for a typical neuron. This seems extremely direct, but it in some cases it fails to explore the most interesting peri-threshold portion of the F-I curve, where the dynamics of single spikes contain a great deal of information about peri-threshold dynamics. For example, an after-hyperpolarization that is visible after single spike at rheobase may become completely swamped by the combination of inward pipette current and sodium current at values of injected current that are high above threshold.

4.2.5 Does a Genetic Algorithm Adequately Report the Contours of the Error Surface?

The error surface is vast, corresponding to all possible combinations of parameters at infinite resolution. Naturally, this is only sparsely and strategically sampled. Consequently there is no guarantee that it is has been adequately explored, and that lower error parameter sets do not exist somewhere that the optimizer did not adequately explore.

Next I describe how to visibly “clean" corrugations from the error surface when using an approach based on a small list of non-contingent errors vs an alternative approach using models whose measurements are contingent on parameter dependent current values.

For each model a current is found that forces the model to fire at 12 spikes. Although not the exactly the same as rheobase, the algorithm is structurally identical, and the problems are the same. When eliciting a pre-determined spike count for any model parameterization (1 or x). This algorithm has the same propensity as the rheobase algorithm to introduce small current excesses into readings of subsequent tests, in this case EFEL tests of spike train shape. Figures 49 and 50 below show error surfaces from each paradigm. Each of these figures depicts a heatmap of a 2D cross-section of the error surface. The sensitivity of the objective function to systematic changes (grid search) of two parameters, centered around the optimizer’s own solution, was explored. All other parameters were held constant at the optimal values. This effectively depicts a cross-section of the error surface near that solution.

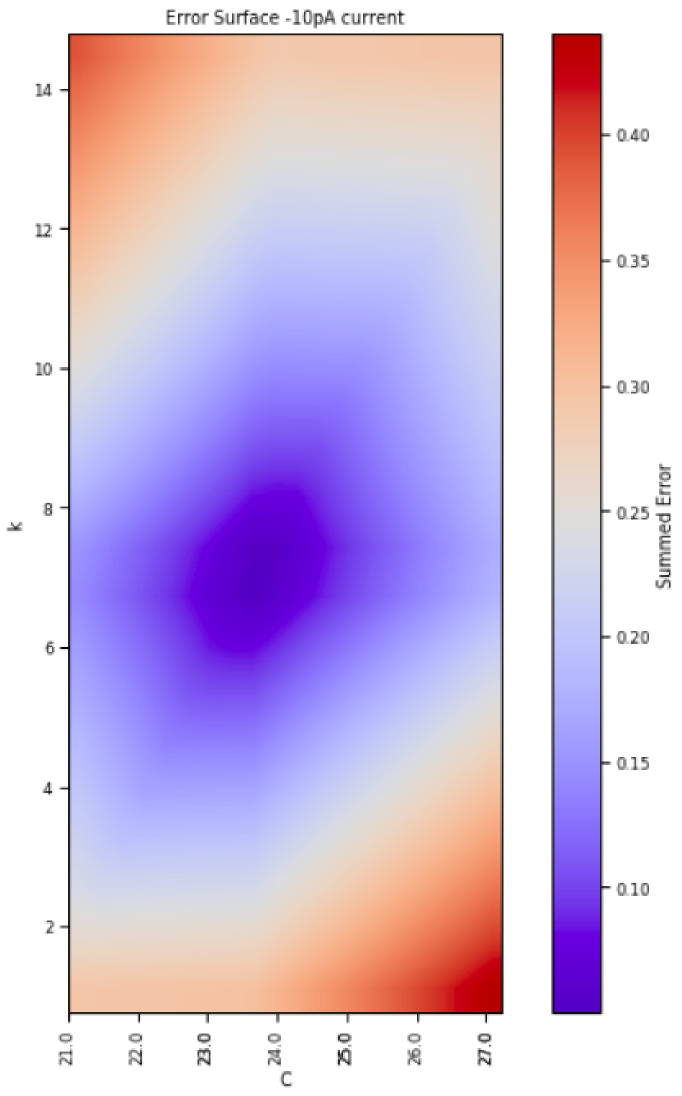


Figure 49: **2D Cross-section of the Error Surface for an Izhikevich Model with No Contingent Tests**. The model was optimized against simulated data using 5 features. Any other features whose calculation is contingent on the rheobase were deliberately excluded. A 2D grid search was then applied (for parameters C and k) around the resulting optimal set of parameters in order to visualize the local error surface. Although only 2 dimensions are explored here, there is no evidence of corrugation in this error surface, indicating that manageable, convex error surfaces can be realized when contingencies are removed.

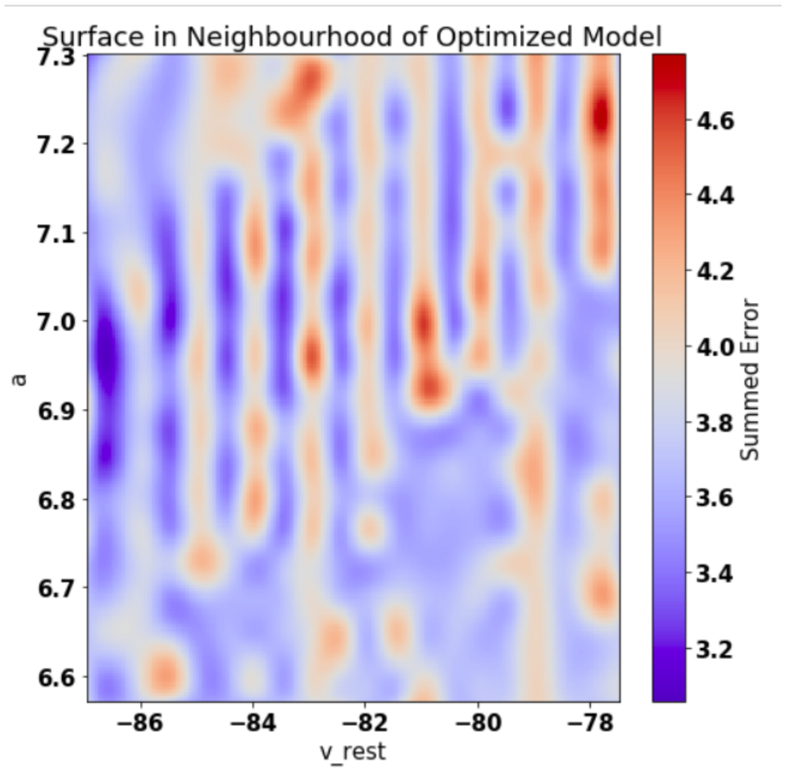


Figure 50: **2D Cross-section of the Error Surface for an AdEx Model using EFEL Features.** Similar to Figure 49 except using an AdEx model and all 14 EFEL features. As the approach for creating multispiking model fits also contained contingent tests, the error surface is complex and Rastrigin-like

I suspect that other pre-existing optimizers that aim to fit neuronal models are largely ignorant of the error surface they face. In fact, without comprehensive data about the convergence rates of various competing approaches to optimization, we cannot now how efficiently each obtains its solutions, nor about alternative solutions that may have been missed.

Nonetheless, I am confident that genetic algorithms (in general) are preferable to to exhaustive search (which is impractical for all but the smallest models) and to gradient-descent-like approaches (due to the nature of the error surface). However, successful optimization may benefit from periodic exhaustive, local grids searches of the parameters space near the optimized values, in order to evaluate whether the error surface was tractable in the first place.

4.3 Satisficing Versus Optimizing

Unless hunting for a new prime number, few are willing to run compute jobs lasting for months, thus there are steeply limited budgets for exploring solution spaces. It therefore seems prudent to accept solutions which are are not optimal but are “good enough", especially if these solutions can be obtained in a small fraction of the time. Not all real neurons, even of the same nominal type, have identical properties, so perhaps we should embrace such variability in optimization results as well. If the highest objective is to recover biologically plausible models, and many fitted models can easily meet this objective, then weaknesses in the ability of the optimizer to succeed in identifying the exact optimum in a very complex error surface can be tolerated.

Fortunately the coupling of Neuronunit to a Genetic Algorithm facilitates either optimizing or “satisficing" as appropriate. The term “satisfice" means that a measured property is either deemed optimal or merely satisfactory (Simon, 1956). Although we may not know if a true optimum has been achieved, by using NeuronUnit in the evaluation, we can determine if the result is satisfactory enough, and then terminate optimization early. This could be done by tracking the χ^2 statistic during optimization and ending as soon as it drops below a certain level.

4.4 Appropriate Optimization Constraints

4.4.1 Conflicting Optimization Constraints

I identified multiple conflicts between features used for optimization. In some cases, the conflicts may arise from limitations of the model, i.e. the model may lack the richness to lie at precisely the points in features space that some real neurons lie in. However, I found that even for biophysically detailed neuron models, this limitation was sometimes still observed. Thus, there must be some other source of these conflicts, which I identified as a conceptual pitfall in optimization: feature values computed from the mean of the corresponding features across many neurons (as in NeuroElectro) may not actually describe any real, single neuron (or in some cases any possible neuron). This was typical when combining the Rheobase, Input Resistance, and Time Constant, for example.

4.4.2 Conflicts in Neuroelectro Measurements

Membrane Time Constant measurements were consistently harder to use for guiding optimization, being incompatible with other features in almost all model types (including conductance based models, reduced models, and including all data types (NeuroElectro data and Allen Cell Types single experiment data). This value is proportional to cell surface area, and it may be that cells taken from, for example, slices of different thicknesses have different capacitances but that this has minimal impact on other measurements. Alternatively, most capacitance measurements could be in error due to recording pipette capacitance artifacts. Regardless, when the membrane time constant was not included, optimized models were still able to behave in most other respects like their experimental counterparts

As discussed in section 2.4 rheobase is strictly defined as the minimum current injection to evoke exactly one spike. It does not fully define the FI curve, and there are an infinite number of FI curves that contain the points $(Rheobase^{-}, 0)$ and $(Rheobase^{+}, > 0)$.

Interestingly, if selecting only one suprathreshold feature to pair in optimization with the remaining physiological features, the FI slope was a better fit than the rheobase. It may be that by considering the whole FI curve in optimization counter-intuitively makes optimization more flexible by allowing small misses on matching the rheobase in exchange for a better overall match to the remaining suprathreshold spike counts.

4.4.3 Sufficient Optimization Constraints

Avoiding conflict constraints, which tests are minimally sufficient to produce good optimization results? Tests that worked within optimization: Via *Elephant* toolchain: FITests, Rheobase, Capacitance, Input Resistance, Time Constant, Resting Membrane Potential. Via.

Druckmann *et al.* (2007) optimized neuron models using only suprathreshold stimuli by considering (1) spike rate; (2) an accommodation index; (3) latency to first spike;(4) average AP overshoot; (5) average depth of after hyperpolarization (AHP); and (6) average AP width. However, when optimizing reduced neuron models, I found that the those 6 features measurements were insufficient for optimization, and additional constraints were required. Overall, I found that the following features from the EFEL library were sufficient:

1. AHP-depth
2. all-ISI-values
3. Spikecount
4. adaptation-index
5. mean-AP-amplitude
6. min-voltage-between-spikes
7. minimum-voltage
8. peak-voltage
9. spike-half-width
10. time-to-first-spike
11. time-to-last-spike
12. voltage-base

4.5 How Good was the Optimizer?

When the optimizer is applied to the Izhikevich model for each of the four different classes of experimental cell types, it obtains solutions that are within the range of empirical variability. Because the optimizer was not able to resolve conflicts between certain measurements, such as usually rheobase vs (time constant,input resistance, and membrane capacitance) performance is be even better if some of these conflicting features are removed.

In the single-compartment conductance-based model and in the AdEx model, even better solutions were obtained, but in these cases it was the need to match experimental input resistance that posed the greatest drag on solution quality.

4.6 Distinction of Optimization Approach From Other Approaches

There are several conceptual and technical differences between the work described here and what has been done previously.

First, I used a formal testing framework–normally used to evaluate model quality–to drive the optimization itself. This means that we can evaluate not only technical performance of the optimizer, but the actual quality of the solutions, and compare these across models and cell types. Other modeling efforts have employed data-driven testing in model development workflows, but all these efforts have been based on non-standard 'in-house' model types and execution environments. By contrast, this work expands a pre-existing community driven, standardized model testing framework, NeuronUnit, so that model validation, optimization, and re-use is broadly applied.

Second, I unified many sources of experimental data and distinct feature types (subthreshold and suprathreshold) into a unified testing suite. This has the potential to result in optimized models that reproduce not only specific patterns of spike but whole input/output transformations. Reproducing these transformations is necessary if there is any hope for reduced models to be used in network models.

Third, I leveraged the large number of cortical neuron and neuronal network models are available in the standardized NeuroML format. This allowed me evaluate the quality of existing models, identify their discrepancies against corresponding experimental data, and improve upon dozens of them in one stroke. Although the Allen Institute for Brain Science modeling project and the Blue Brain project both rigorously analyzed their single cell models, to the best of my knowledge there has not been an overarching meta-analysis across different cell and network model sources.

4.7 Generalizing Behavior Across Neuron States

Some neurons may have different behaviors under different stimulation paradigms. For example, the cerebellar Purkinje cell is sensitive to intricately patterned dendritic input current combinations. Depending on that cell's recent history of synaptic stimulation, it may toggle between coincidence detection and integration modes Ratté *et al.* (2013). If the experimental data being used for optimization only describes a subset of these behaviors, then the optimized model may not generalize to the neuron's other behaviors. This may not even require the neuron to have state-dependence; the neuron may simply have two very different behavioral regimes, where one regime is difficult to predict even from a large number of observations of behavior in the other regime.

Making a model that does generalize may thus require a number of stimuli from each behavioral regime. I included only a handful of subthreshold and suprathreshold stimuli (which was more than some previous efforts that only included suprathreshold stimuli), and through NeuroML-DB I also have access to many more precomputed stimuli. One data sets permitted, I explored the entire the FI curve. It remains possible that a fully general model may require still more stimuli, such as sinusoidal stimuli of frozen noise currents. was as much I did here. A fitted model may not generalize to current stimulus at higher or lower current strengths, Unless the model is fitted with constraints informed by those different experiments. As discussed above, the true FI curve is described by a gradient and a bias. When one fits to both the slope and the bias of the FI curve, it means that the model will at least be able to recapitulate the right number of spikes for a given current strength.

Nonetheless, in the cell types I studied here, even optimizing against simple subthreshold electrophysiological features alone (Input resistance, Membrane Time Constant, Resting Potential, and Capacitance), in conjunction with a single suprathreshold feature (Rheobase) was often sufficient to reproduce the remainder of the observed suprathreshold behaviors.

4.8 Largest Possible Survey of Features

The NeuronUnit core, the Electrophysiology Feature Extraction Library (Van Geit, 2015b)) EFEL, the Allen SDK, and the tests derived from (Druckmann *et al.*, 2008) all contain independently written algorithms for computing features from simulated membrane potentials. In some cases, the same feature (e.g. action potential threshold) is computed in many different ways across these feature extraction libraries. All compute the threshold by first taking the first difference of the membrane potential, but they differ in subsequent steps, for example what value the first difference must reach to be considered at or beyond threshold. This is inconvenient, but one can simply select one (or more) of the alternatives and apply it consistently. More troubling are the cases where these algorithms differ in the stimulus used to generate the feature. For example, the suprathreshold tests of (Druckmann *et al.*, 2008) are based on a multiple of rheobase (e.g. 1.5× or 3×), whereas those used by The Allen Insitute are based on additive increments from rheobase (e.g. +20 pA or +40 pA). This means that cannot simply be applied to the same membrane potential trace, as those who collected the experimental data are likely to have chosen either multiples or additive increments of rheobase, but not both, depending on which lab they happen to work for.

As discussed in the Methods, I wrote code to restructure the Allen Cell Types and Blue Brain Project data sets, sorting traces into (approximate) multiples of rheobase, and the interpolating to select the sweep (in one data set) that is most appropriate for computing the feature defined in an otherwise-incompatible feature library. In this way I was able to generate the same set of features from distinct datasets, even those that had use different stimuli.

One can conceive of more mathematically savvy types of imputation, and these could be the basis for future work. If a more robust form of imputation was achieve, it could act as a Rosetta stone to link all the above feature extraction libraries together, allowing nearly any *in vitro* dataset that is being collected to be used to produce a canonical and universal set of features for model optimization.

4.9 Cross-Validation for Feature Selection: One Feature Set to Rule Them All

The features identified during multivariate analysis utlilized the entire collection of models, data, and stimuli. By contrast, a machine learning approach would have held out some of these for cross-validation, checking to see whether the features identified on a training set were in fact that features that best cluster, discriminate, or simply explain variance in the a held-out set of models and data. For example, future work could identify a canonical set of stimuli and features fitness criteria, criteria that can train models to best satisfy multiple experimental protocols, rather than just one particular experiment. These features would be shuffled (using stratification according to the stimuli that generated them) the objectives into “train" and “test" sets. Cross-Validation may provide a strategy for dealing with conflicted features, such as the input resistance test. If a trained model failed to score remotely well on such a test, this might identify that the model was overfit or simply that the test was unsatisfiable (given a model fit to the other tests). This would allow for quick identification of which test combinations cannot work in practice, while also producing more generalizable models.

4.10 Which is the Best Model Class for Producing Optimized Reduced Models?

In previous spike-time prediction competitions (Naud *et al.*, 2009), multi-time-scale variants of the AdEx model performed best. It also had an admirable performance here, however the Izhikevich model was the overall winner.

Considering only features of at or below threshold spiking, the features that where used to get results below come from the tables

By consulting tables: (8,12), and aggregating scores using python pandas, and averaging across χ^2 statistics (which reflected 12 tests) for all neuron types, the Izhikevich model had the lowest mean χ^2 (AdExp=28.55; Izhikevich=1.39), and the highest average $p-value$. However, the Izhikevich model did not have the lowest χ^2 for every cell type (see the Appendix for details).

The AdEx model suffered from its performance on the Olfactory Bulb Mitral Cell and the Cerebellum Purkinje cell (two of the largest cells types that were optimized). However, if you remove these two cells, and then judge the optimized models for the remaining cell types, the AdEx model is the winner by χ^2 (AdExp=0.51; Izhikevich=0.99).

When consulting some of the traces for fitted models, it seemed as if either the

AdExp and Izhikevich models lack the flexibility to model diverse cortical neuron activity, or the NeuroElectro data was difficult to fit.

Consider two sets of features: RheobaseTest, TimeConstantTest, RestingPotentialTest, InputResistanceTest or RheobaseTest, FITest. If you consider only the Allen Cell Types data (all cortical neurons), with these two sets of features, the GLIF model is associated with $\chi^2 = 1.0$; however, if you consider other neurons (and tests), the mean χ^2 value jumps to 364.2. The GLIF model was able to recapitulate the FI curve slope exactly; however, this caused conflicts with the ability to fit the time constant and rheobase. Even when the GLIF model was able to optimize cells to be within a biologically plausible range, simulation was slow and the quality of fits was not astounding. It is unclear why anyone would choose to use this model outside of a very narrow range of applications.

Why did the Izhikevich model perform best? I hypothesized that this was related to its broader coverage of dynamical regimes (as shown in (Izhikevich, 2003)). To test this, I asked how many of these dozens of regimes were “occupied" by the optimization solutions obtained here. Specific optimization results (Section 4.12.3). I developed the Izhikevich model, such that regime type was an explicit integer model parameter. Numbers 3-7 encoded regime types 1-7, where regimes 1,2,3 follow identical governing equations and so do not require unique numeric identifiers. I show that all but one of these regimes were occupied except for one associated with the activity of a dorsal LGN thalamocortical cell. The ones associated with a barrel cortex low-threshold spiking (LTS) interneuron and a Layer 5 visual cortex fast-spiking (FS) interneuron were occupied the most often.

4.11 Parameter Boundaries

The error surface explored by the optimizer cannot extend infinitely far in all directions. It must be initialized with boundaries that contain plausible parameter values. If these boundaries are too narrow, they may exclude the optimal set of parameter values. However, if they are too wide, then parameter sets that produce models well outside the range of biological plausibility will be produced. Such implausible models undermine the assumptions of the features being calculated, and result in discontinuous or otherwise uninformative error surfaces where the optimizer wastes time exploring and may struggle to escape. An example of parameter boundaries from the literature is shown in Figure 52.

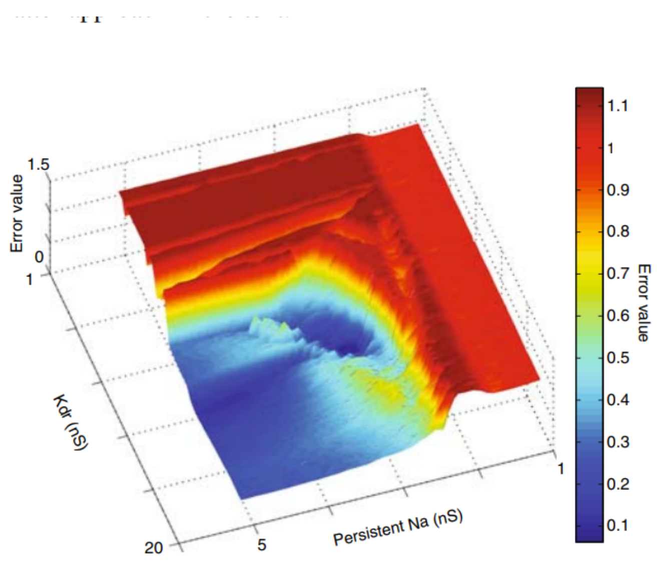


Figure 51: **Parameter Boundaries Can Frame the Search Space.** This error surface plot (from Van Geit *et al.* (2007, 2008)) shows how model/data agreement varies as two conductances from a biophysically-realistic conductance-based model are varied. The large error values for $Na < 2$ and $Kdr < 2$ suggest that the decision not to search below the value 1 for either of these parameter was reasonable. Similar grid searches could help to justify the parameter boundaries for optimization problems, provided that they can be conducted at low computational cost relative to optimization itself.

For example, some parameter sets may cause a divergence in either the simulated membrane potential or in the features extracted from it. This will be encoded as either “not a number" (NaN) or inf. A single (NaN) or inf can infect the entire multiobjective function (i.e. any sum with inf will be inf). Because the optimizer can only succeed when it can distinguish better parameter sets from worse parameter sets, any chromosomes that get stuck in regions of parameter space with (NaN) or inf may not escape, even through mutation or crossover. The error surface is locally “flat" in a sense, and thus uninformative. In order for the optimizer to survive the existence of such regions, the population size must be sufficiently large that a large number of chromosomes will avoid being initialized there.

In the opposite case, consider what might happen if the parameter boundaries are too narrow. Figure 52 shows another case from the literature, and in this case the global minimum error is positioned in a deep and narrow well. It would be easy to have chosen parameter boundaries that miss this well entirely, resulting in a failure to obtain the optimal parameter set.

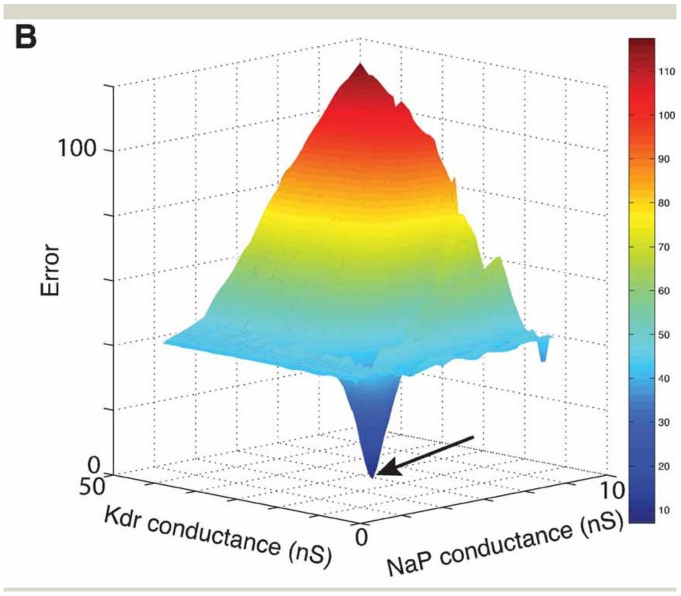


Figure 52: **A Challenging Location for the Optimal Value**. Again from Van Geit *et al.* (2008) I show an example of well-chosen parameter boundaries. These boundaries enclose the (narrow, deep) global minimum. Notably, choosing to explore only one quadrant of this space, where the error is roughly invariant to the parameter values, would have led to missing this global minimum entirely.

A possible solution the problems above is to employ algorithms that peek beyond parameter boundaries and reports back on model stability. This would be done only sparingly (otherwise it is equivalent to simply expanding the boundaries).

4.12 Future Work

4.12.1 Fixing Currents

Here I identified a problem of contingent tests, which cause sometimes intractable defects in the error surface. This could be solved by specifying all injected currents in advance and fitting directly to the entire FI curve. Having been fit, features that are defined near rheobase (such as spike threshold) should then be computed at a pre-determined current.

4.12.2 Model Exchange

The NeuroML model exchange format (Gleeson *et al.*, 2010) offers a portable, simulator agnostic, description of a neuron model including its parameters. A logical next step is to capture the results of optimization and use them to specify these parameters, resulting in an optimized model that anyone can use.

4.12.3 Embedding Models

Reduced models could be especially valuable when embedded into a larger network that simulated a part of the brain The Allen Institute followed an approach like, encasing a "core" of biophysically accurate models inside a "shell" of surrounding, simple, fast and reduced GLIF models (Billeh *et al.*, 2020). Since almost all cortical neurons experience “tonic" synaptic input, often originating from outside of the local circuit, such reduced models could be used to provide this input in a semi-realistic way, even as the neurons they target are simulated in greater detail. Currently, this is often handled statistically rather than dynamically, by simulating some number of point processes to provide such input. However, this makes strong assumptions about the nature of that input, and does not allow it to change dynamically the way it would if it originated from an actual neuronal or network simulation. Reduced models can also be extended to include multiple compartments, allowing Local Field Potential (LFP) analysis to be included among the benefits of such an approach to network modeling. The statistical approach also severs the link between cause and effect, and does not have any phenomenological relationship to known physics. Finally, Izhikevitch and AdExp models are commonly utilized in neuromorphic spiking neural networks in artificial intelligence and biomedical modelling contexts, suggesting that their optimization in software could result in superior implementations of identified cell types in hardware.

REFERENCES

Ball, G., P. Aljabar, S. Zebari, N. Tusor, T. Arichi, N. Merchant, E. C. Robinson, E. Ogundipe, D. Rueckert, A. D. Edwards *et al.*, "Rich-club organization of the newborn human brain", Proceedings of the National Academy of Sciences **111**, 20, 7456–7461 (2014).

Banino, A., C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil *et al.*, "Vector-based navigation using grid-like representations in artificial agents", Nature **557**, 7705, 429–433 (2018).

Billeh, Y. N., B. Cai, S. L. Gratiy, K. Dai, R. Iyer, N. W. Gouwens, R. Abbasi-Asl, X. Jia, J. H. Siegle, S. R. Olsen *et al.*, "Systematic integration of structural and functional data into multi-scale models of mouse primary visual cortex", Neuron (2020).

Birgiolas, J., *Towards Brains in the Cloud: A Biophysically Realistic Computational Model of Olfactory Bulb*, Ph.D. thesis, Arizona State University (2019).

Birgiolas, J., R. Gerkin and S. Crook, "Rapid selection of neuroml models via neuroml-db. org", NEURON **28**, 10, 2063–2090 (2016).

Brette, R. and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity", Journal of neurophysiology **94**, 5, 3637–3642 (2005).

Carnevale, N. T. and M. L. Hines, *The NEURON book* (Cambridge University Press, 2006).

Colquhoun, D., "Ion channels of excitable cells (methods in neurosciences vol. 19): edited by toshio narahashi, academic press, 1994,£ 65.00 (xiii+ 387 pages) isbn 0 12 185287 3", Trends in Biochemical Sciences **19**, 9, 389 (1994).

Davison, A., "Adexp neuron with deltat=0 doesnt produce spikes with brian backend issue 370 neuralensemble/pynn", URL https://github.com/NeuralEnsemble/PyNN/issues/370 (2020a).

Davison, A., "pynn.neuron implementation of adexp is unstable, gives poor results issue 266 neuralensemble/pynn", URL https://github.com/NeuralEnsemble/PyNN/issues/266 (2020b).

Davison, A. P., D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet and P. Yger, "Pynn: a common interface for neuronal network simulators", Frontiers in neuroinformatics **2**, 11 (2009).

Deb, K., S. Agrawal, A. Pratap and T. Meyarivan, “A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii”, in “International conference on parallel problem solving from nature”, pp. 849–858 (Springer, 2000).

Denker, M., A. Yegenoglu and S. Grün, “Collaborative HPC-enabled workflows on the HBP Collaboratory using the Elephant framework”, in “Neuroinformatics 2018”, p. P19 (2018), URL https://abstracts.g-node.org/conference/NI2018/abstracts#/uuid/023bec4e-0c35-4563-81ce-2c6fac282abd.

Di Luca, M., D. Nutt, W. Oertel, P. Boyer, J. Jaarsma, F. Destrebecq and V. Quoidbach, “Towards earlier diagnosis and treatment of disorders of the brain. bulletin of the world health organization”, URL $http://doi.org/10.2471/blt.17.206599$ (2018).

Draganski, B. and A. May, “Training-induced structural changes in the adult human brain”, Behavioural Brain Research **192**, 1, 137–142 (2008).

Druckmann, S., Y. Banitt, A. A. Gidon, F. Schürmann, H. Markram and I. Segev, “A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data”, Frontiers in neuroscience **1**, 1 (2007).

Druckmann, S., T. K. Berger, S. Hill, F. Schürmann, H. Markram and I. Segev, “Evaluating automated parameter constraining procedures of neuron models by experimental and surrogate data”, Biological cybernetics **99**, 4-5, 371 (2008).

Druckmann, S., S. Hill, F. Schürmann, H. Markram and I. Segev, “A hierarchical structure of cortical interneuron electrical diversity revealed by automated statistical analysis”, Cerebral Cortex **23**, 12, 2994–3006 (2013).

EFEL-Developers, “Efel documentation”, URL https://efel.readthedocs.io/en/latest/eFeatures.html (2018).

Fortin, F.-A., F.-M. De Rainville, M.-A. Gardner, M. Parizeau and C. Gagné, “DEAP: Evolutionary algorithms made easy”, Journal of Machine Learning Research **13**, 2171–2175 (2012).

Friedrich, P., M. Vella, A. I. Gulyás, T. F. Freund and S. Káli, “A flexible, interactive software tool for fitting the parameters of neuronal models”, Frontiers in neuroinformatics **8**, 63 (2014).

Garcia, S., D. Guarino, F. Jaillet, T. Jennings, R. Prpper, P. L. Rautenberg, C. C. Rodgers, A. Sobolev, T. Wachtler, P. Yger and A. P. Davison, “Neo: an object model for handling electrophysiology data in multiple formats”, Front Neuroinform **8**, 10 (2014a).

Garcia, S., D. Guarino, F. Jaillet, T. R. Jennings, R. Pröpper, P. L. Rautenberg, C. Rodgers, A. Sobolev, T. Wachtler, P. Yger *et al.*, "Neo: an object model for handling electrophysiology data in multiple formats", Frontiers in neuroinformatics **8**, 10 (2014b).

Gerkin, R. C., J. Birgiolas, R. J. Jarvis, C. Omar and S. M. Crook, "Neuronunit: A package for data-driven validation of neuron models using sciunit", bioRxiv URL https://www.biorxiv.org/content/early/2019/06/09/665331 (2019).

Gerstner, W., W. M. Kistler, R. Naud and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition* (Cambridge University Press, 2014).

Gleeson, P., M. Cantarelli, B. Marin, A. Quintana, M. Earnshaw, S. Sadeh, E. Piasini, J. Birgiolas, R. C. Cannon, N. A. Cayco-Gajic *et al.*, "Open source brain: a collaborative resource for visualizing, analyzing, simulating, and developing standardized models of neurons and circuits", Neuron **103**, 3, 395–411 (2019).

Gleeson, P., S. Crook, R. C. Cannon, M. L. Hines, G. O. Billings, M. Farinella, T. M. Morse, A. P. Davison, S. Ray, U. S. Bhalla *et al.*, "Neuroml: a language for describing data driven models of neurons and networks with a high degree of biological detail", PLoS Comput Biol **6**, 6, e1000815 (2010).

Goldin, M. A. and G. B. Mindlin, "Temperature manipulation of neuronal dynamics in a forebrain motor control nucleus", PLoS computational biology **13**, 8, e1005699 (2017).

Gouwens, N. W., J. Berg, D. Feng, S. A. Sorensen, H. Zeng, M. J. Hawrylycz, C. Koch and A. Arkhipov, "Systematic generation of biophysically detailed models for diverse cortical neuron types", Nature communications **9**, 1, 1–13 (2018).

Hertäg, L., J. Hass, T. Golovko and D. Durstewitz, "An approximation to the adaptive exponential integrate-and-fire neuron model allows fast and predictive fitting to physiological data", Frontiers in computational neuroscience **6**, 62 (2012).

Hodgkin, A. L. and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve", The Journal of physiology **117**, 4, 500 (1952).

Institute, A., "Allen cell types database", URL https://celltypes.brain-map.org (2015).

Izhikevich, E. M., "Simple model of spiking neurons", IEEE Transactions on neural networks **14**, 6, 1569–1572 (2003).

Jarvis, R., URL https://github.com/fun-zoological-computing/BluePyOpt/blob/master/examples/bpo_nu_fusion/allen_efel_nu_deployed_tests_only-thesis.ipynb (2020a).

Jarvis, R., “Jithub: A collection of jit-enabled reduced neuron models”, URL https://github.com/russelljjarvis/jit_hub.git (2020b).

Jarvis, R. J., URL https://github.com/BlueBrain/BluePyOpt/blob/master/examples/l5pc/L5PC.ipynb (2020c).

Jones, E. G., “Golgi, cajal and the neuron doctrine”, Journal of the History of the Neurosciences **8**, 2, 170–178 (1999).

Lam, S. K., A. Pitrou and S. Seibert, “Numba: A llvm-based python jit compiler”, in “Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC”, pp. 1–6 (2015).

Maechler, M., “Package ‘diptest”’, R Package Version 0.75–5. R: a language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing (2013).

Mainen, Z. F. and T. J. Sejnowski, “Reliability of spike timing in neocortical neurons”, Science **268**, 5216, 1503–1506 (1995).

Marder, E. and A. L. Taylor, “Multiple models to capture the variability in biological neurons and networks”, Nature neuroscience **14**, 2, 133–138 (2011).

Markram, H., “The blue brain project”, Nature Reviews Neuroscience **7**, 2, 153–160 (2006).

Markram, H., E. Muller, S. Ramaswamy, M. W. Reimann, M. Abdellah, C. A. Sanchez, A. Ailamaki, L. Alonso-Nanclares, N. Antille, S. Arsever *et al.*, “Reconstruction and simulation of neocortical microcircuitry”, Cell **163**, 2, 456–492 (2015).

Marr, D. and T. Poggio, “From understanding computation to understanding neural circuitry”, (1976).

Meunier, C. and I. Segev, “Playing the devil’s advocate: is the hodgkin–huxley model useful?”, Trends in neurosciences **25**, 11, 558–563 (2002).

Naud, R., T. Berger, L. Badel, A. Roth and W. Gerstner, “Quantitative single-neuron modeling: competition 2008”, Front. Neur. Conference Abstract: Neuroinformatics 2009 (2009).

Omar, C., J. Aldrich and R. C. Gerkin, “Collaborative infrastructure for test-driven scientific model validation”, in “Companion Proceedings of the 36th International Conference on Software Engineering”, pp. 524–527 (2014).

Quadrato, G., T. Nguyen, E. Z. Macosko, J. L. Sherwood, S. M. Yang, D. R. Berger, N. Maria, J. Scholvin, M. Goldman, J. P. Kinney *et al.*, "Cell diversity and network dynamics in photosensitive human brain organoids", Nature **545**, 7652, 48–53 (2017).

Rall, W., "Electrophysiology of a dendritic neuron model", Biophysical journal **2**, 2, 145–167 (1962).

Ramaswamy, S., J.-D. Courcol, M. Abdellah, S. R. Adaszewski, N. Antille, S. Arsever, G. Atenekeng, A. Bilgili, Y. Brukau, A. Chalimourda *et al.*, "The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex", Frontiers in neural circuits **9**, 44 (2015).

Rastrigin, L. A., "Systems of extremal control", Nauka (1974).

Ratté, S., S. Hong, E. De Schutter and S. A. Prescott, "Impact of neuronal properties on network coding: roles of spike initiation dynamics and robust synchrony transfer", Neuron **78**, 5, 758–772 (2013).

Rocklin, M., "Dask: Parallel computation with blocked algorithms and task scheduling", in "Proceedings of the 14th python in science conference", No. 130-136 (Citeseer, 2015).

Rossant, C., D. F. Goodman, B. Fontaine, J. Platkiewicz, A. K. Magnusson and R. Brette, "Fitting neuron models to spike trains", Frontiers in neuroscience **5**, 9 (2011).

Rossant, C., D. F. Goodman, J. Platkiewicz and R. Brette, "Automatic fitting of spiking neuron models to electrophysiological recordings", Frontiers in neuroinformatics **4**, 2 (2010).

Shepherd, G. M., *Foundations of the neuron doctrine* (Oxford University Press, 2015).

Simon, H. A., "Rational choice and the structure of the environment", Psychological review **63**, 2, 129 (1956).

Spruston, N., M. Hausser, G. J. Stuart *et al.*, "Information processing in dendrites and spines", in "Fundamental neuroscience", (Academic Press, 2013).

Steamlit-Team, "Streamlit 0.69.2 documentation", URL https://docs.streamlit.io/en/stable/, accessed: 2020-10-26 (2020).

Stimberg, M., R. Brette and D. F. Goodman, "Brian 2, an intuitive and efficient neural simulator", Elife **8**, e47314 (2019a).

Stimberg, M., R. Brette and D. F. Goodman, "Brian2modelfitting 0.3 documentation", (2019b).

Teeter, C., R. Iyer, V. Menon, N. Gouwens, D. Feng, J. Berg, A. Szafer, N. Cain, H. Zeng, M. Hawrylycz *et al.*, "Generalized leaky integrate-and-fire models classify multiple neuron types", Nature communications **9**, 1, 1–15 (2018).

Teeters, J. L., K. Godfrey, R. Young, C. Dang, C. Friedsam, B. Wark, H. Asari, S. Peron, N. Li, A. Peyrache *et al.*, "Neurodata without borders: creating a common data format for neurophysiology", Neuron **88**, 4, 629–634 (2015).

Toledo, M., M. Telefont and H. Markram, "Electrophysiological properties of neurons in the rat somatosensory cortex at postnatal day 13-16", URL microcircuits.epfl.ch/#/article/article_4_eph (2016).

Tripathy, S. J., J. Savitskaya, S. D. Burton, N. N. Urban and R. C. Gerkin, "Neuroelectro: a window to the world's neuron electrophysiology data", Frontiers in neuroinformatics **8**, 40 (2014).

Van Geit, W., "Efel documentation", URL https://efel.readthedocs.io/en/latest/eFeatures.html (2015a).

Van Geit, W., "Electrophys feature extraction library (efel)", URL https://github.com/BlueBrain/eFEL (2015b).

Van Geit, W., P. Achard and E. De Schutter, "Neurofitter: a parameter tuning package for a wide range of electrophysiological neuron models", Frontiers in neuroinformatics **1**, 1 (2007).

Van Geit, W., E. De Schutter and P. Achard, "Automated neuron model optimization techniques: a review", Biological Cybernetics **99**, 4-5, 241–251 (2008).

Van Geit, W., M. Gevaert, G. Chindemi, C. Rössert, J.-D. Courcol, E. B. Muller, F. Schürmann, I. Segev and H. Markram, "Bluepyopt: leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience", Frontiers in neuroinformatics **10**, 17 (2016a).

Van Geit, W., M. Gevaert, G. Chindemi, C. Rössert, J.-D. Courcol, E. B. Muller, F. Schürmann, I. Segev and H. Markram, "Bluepyopt: Leveraging open source software and cloud infrastructure to optimise model parameters in neuroscience", Frontiers in Neuroinformatics **10**, 17, URL http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2016.00017/abstract (2016b).

Vella, M. and P. Gleeson, "Neurotune", URL https://github.com/NeuralEnsemble/neurotune (2012).

Zhu, J. J., "Maturation of layer 5 neocortical pyramidal neurons: amplifying salient layer 1 and layer 4 inputs by ca2+ action potentials in adult rat tuft dendrites", The Journal of physiology **526**, 3, 571–587 (2000).

Zou, H., T. Hastie and R. Tibshirani, “Sparse principal component analysis”, Journal of computational and graphical statistics **15**, 2, 265–286 (2006).

APPENDIX A

CODE DETAILS

All of the optimization work newly described here depends on code I developed in two GitHub repositories:

- Neuronunit https://github.com/russelljjarvis/neuronunit
- BluepyOpt https://github.com/russelljjarvis/BluePyOpt
- Features https://github.com/russelljjarvis/AllenEFELDruckmanFeatures
- JIT models https://github.com/russelljjarvis/jit_hub

APPENDIX B

COMPLEX MODELS AND STREAMING REMOTE DATA

* Often model data is streamed from a remote machine for example NeuroML-DB. In such cases where local simulator capabilities are not required, it is more useful to make “streamed" remote models eligible for NeuronUnit tests. The code idiom for NeuronUnit scoring of streamed models is as follows:

```
import requests
from neuronunit.models import static_model as sm
model_scores = []
# List cell models desired from from the remote machine.
NeuroMLDB_requests = ["Traub L3-Pyramidal-Cell","Traub-Relay-Cell"]
for data in NeuroMLDB_requests:
    # Get those cell models
    remote_model = requests.get(data)
    # Get one membrane potential trace per model
    vm = remote_model.json['vm']
    # Make the trace usable by NeuronUnit
    SM = sm.StaticModel(vm)
    # Get a NeuronUnit score for the model
    score = test_suite.judge(SM)
    model_scores.append(score)
```

This idiom was also utilized for complex models that lack NeuronUnit support. In the majority of cases not much is gained by re-implenting complex model simulator code. Instead, one can simply source simulated models as if they are being streamed remotely when in fact they coexist in the same python program.

APPENDIX C

ALLEN EXPERIMENTAL DATA FROM WEB PAGES IN EPHYSIOLOGICAL REPORT FORM

*

specimen id 623960880
http://celltypes.brain-map.org/mouse/experiment/electrophysiology/623960880 specimen id 623893177
http://celltypes.brain-map.org/mouse/experiment/electrophysiology/623893177 specimen id 482493761
http://celltypes.brain-map.org/mouse/experiment/electrophysiology/482493761 specimen id 471819401
http://celltypes.brain-map.org/mouse/experiment/electrophysiology/471819401

APPENDIX D

JULIA IMPLEMENTATION

* I also created an implementation of some cell models in the Julia programming language. This provides even more rapid simulation, intuitive code, and no specialized computing hardware. These are documented at https://github.com/russelljjarvis/NeuronUnitOpt.jl

APPENDIX E

TABLES OF FITTED PARAMETERS AND TEST SCORES FOR MODELS AND DATA USED HERE

	C	k	vr	vt	vPeak	a	b	c	d	regime-type
experiment										
4824937611	190.317718	0.766233	-59.750208	-40.787616	42.848335	0.012862	-1.021203	-58.379143	48.911618	6
4718194011	162.783673	1.583689	-66.487855	-48.634210	30.282912	0.057584	-1.159059	-56.473425	135.693471	7
6238931771	187.915295	1.130626	-63.584364	-49.477821	37.111777	0.060086	8.714604	-57.494334	89.571208	1-3
6239608801	190.848367	0.770070	-65.461028	-49.496538	31.461532	0.058770	13.916167	-58.159122	38.738638	7
482493761	70.602307	1.548712	-55.046278	-41.489000	28.329709	0.139405	13.190249	-46.060902	78.195696	5
471819401	120.971231	1.257246	-70.654637	-47.630679	30.290391	0.056750	-0.489954	-56.034978	138.825641	7
623893177	179.359764	1.533041	-56.883517	-35.949970	43.403127	0.101416	-0.536745	-45.430678	148.242124	7
623960880	198.460449	1.598985	-56.469868	-43.352802	48.817602	0.186193	-1.357591	-47.647756	112.958369	6
olf_mit	166.596093	1.029132	-61.718758	-45.325164	48.016614	0.011900	-1.736929	-55.525865	124.999833	1-3
Neocortex pyramidal cell layer 5-6	181.287370	0.850828	-63.836556	-57.404674	25.079898	0.106505	12.399936	-54.668152	85.950173	7
Cerebellum Purkinje cell	24.623228	1.048620	-61.377334	-48.928462	29.775470	0.100447	11.356889	-48.328073	14.756462	6
Hippocampus CA1 pyramidal cell	100.863146	0.713266	-65.220970	-45.078885	29.810374	0.067234	14.615355	-52.826220	97.845439	5

Table 20: Best Model Fitted Parameters Izhikevich model, the final parameter value, regime number, is a meta parameter that usually modifies u the recovery variable in the Izhikevich equation. Notice too, that it looks like many row elements are duplicated except for differing by '1'. Row elements that are purely numeric denote Allen cell type specimen id's. The extra '1' denotes an easy test set consisting of only FITest, and Rheobase Test.

*

	cm	v_spike	v_reset	v_rest	tau_m	a	b	delta_T	tau_w	v_thresh	spike_delta
experiment											
4824937611	224.255872	-38.561812	-77.704682	-51.608658	58.179308	2.244428	7.084609	2.755020	335.852562	-32.236088	27.968109
4718194011	294.893733	-54.915890	-30.929193	-51.337599	53.050157	4.361102	6.922987	4.611617	238.701988	-19.880977	38.495153
6238931771	291.891401	-43.377835	-75.236520	-52.386590	52.730179	1.937321	3.589253	3.682518	349.415730	-18.847586	45.698272
6239608801	147.166079	-45.027618	-32.175811	-51.521467	51.323969	1.787643	5.553571	4.143261	158.661633	-25.749411	15.471554
482493761	207.087442	-20.733319	-58.211088	-49.915523	33.883649	17.839201	4.651201	5.623945	120.739149	-36.109076	62.658869
471819401	239.622839	-39.219402	-66.030658	-39.475175	24.427497	14.761501	1.466630	7.449099	28.434677	-18.631515	25.017075
623893177	722.882514	-26.376059	-54.078109	-45.220146	32.107865	4.329680	9.104058	2.763061	303.871685	-34.368089	20.093434
623960880	178.059407	-20.577870	-26.568107	-46.302651	9.848878	19.140343	10.363827	6.901561	306.946112	-36.222350	56.319297
olf_mit	297.396881	-51.163281	-69.576191	-57.891142	54.798123	16.591664	3.118892	3.912469	344.409426	-23.650788	45.583662
Neocortex pyramidal cell layer 5-6	170.450121	-24.032164	-51.492037	-67.754010	25.453872	6.973876	1.479477	5.686814	230.131055	-32.842579	31.386174
Cerebellum Purkinje cell	717.111118	-48.432357	-34.206502	-62.769947	24.473459	5.282332	11.820173	9.344534	257.794528	-31.630191	40.033220
Hippocampus CA1 pyramidal cell	113.878716	-27.492145	-30.350509	-65.176063	13.857807	18.884843	9.699535	7.508661	349.611826	-37.237156	34.804450

Table 21: Although it looks like many row elements are duplicated except for differing by '1'. Row elements that are purely numeric denote Allen cell type specimen id's. The extra '1' denotes an easy test set consisting of only FITest, and Rheobase Test. Absence of '1' denotes a more conflicted test set of Rheobase, Time Constant, Input Resistance and Resting Membrane Potential. Olfactory Mitral cell is contracted to just "olf-mit". Olfactory mitral cells, and cerebellum cells where two universal optimization failures.

APPENDIX F

ACROSS MODEL PERFORMANCE COMPARISONS

model type	exp-cell	χ^2	p-value
conductance model	Hippocampus CA1 pyramidal cell	17.21	0.027
conductance model	olf-mit	26487.51	0.0
conductance model	Neo cortex pyramidal cell layer 5-6	2.56	0.95
conductance model	4824937611	2.17	0.97
conductance model	471819401	0.870	0.99
conductance model	482493761	0.036	0.99

Table 22: Comparable χ^2 for optimized results of the conductance based model. Not all models could be evaluated, as optimization took a long time.

	model type	exp_cell	χ^2	p-value
0	IZHI	4824937611	0.034791	1.000000e+00
0	IZHI	4718194011	0.051691	1.000000e+00
0	IZHI	6238931771	0.086530	9.999999e-01
0	IZHI	6239608801	0.001550	1.000000e+00
0	IZHI	482493761	1.292924	9.956362e-01
0	IZHI	471819401	1.443618	9.936050e-01
0	IZHI	623893177	1.334789	9.951233e-01
0	IZHI	623960880	1.014464	9.981553e-01
0	IZHI	olf_mit	6915.484007	0.000000e+00
0	IZHI	Neocortex pyramidal cell layer 5-6	2.443396	9.643182e-01
0	IZHI	Cerebellum Purkinje cell	19.885113	1.077956e-02
0	IZHI	Hippocampus CA1 pyramidal cell	1.273070	9.958661e-01
0	ADEXP	4824937611	0.000017	1.000000e+00
0	ADEXP	4718194011	0.011029	1.000000e+00
0	ADEXP	6238931771	0.005062	1.000000e+00
0	ADEXP	6239608801	0.000083	1.000000e+00
0	ADEXP	482493761	86.949529	1.887379e-15
0	ADEXP	471819401	0.370856	9.999575e-01
0	ADEXP	623893177	0.273030	9.999870e-01
0	ADEXP	623960880	0.140222	9.999990e-01
0	ADEXP	olf_mit	10.353878	2.410614e-01
0	ADEXP	Neocortex pyramidal cell layer 5-6	0.013262	1.000000e+00
0	ADEXP	Cerebellum Purkinje cell	353.692447	0.000000e+00
0	ADEXP	Hippocampus CA1 pyramidal cell	0.735616	9.994308e-01

List of Complete Set of 47 Electrophysiological Features Used Here

1. AP1RateOfChangePeakToTroughTest-3.0x
2. AP2DelaySDTest-1.5x
3. AP2DelaySDTest-3.0x
4. AP-end-indices-3.0x
5. AP-fall-indices-1.5x
6. AP-fall-indices-3.0x
7. AP-rise-indices-1.5x
8. AP-rise-time-1.5x
9. AP-rise-time-3.0x
10. APlast-width-1.5x
11. AccommodationRateMeanAtSSTest-3.0x
12. AccommodationRateToSSTest-3.0x
13. ISIBurstMeanChangeTest-3.0x
14. ISICVTest-3.0x
15. ISI-log-slope-skip-3.0x
16. Spikecount-stimint-1.5x
17. Spikecount-stimint-3.0x
18. adaptation-index2-3.0x
19. fast-trough-index-1.5x
20. fast-trough-index-3.0x
21. fast-trough-t-1.5x
22. fast-trough-t-3.0x
23. initburst-sahp-vb-1.5x
24. input-resistance
25. maximum-voltage-1.5x
26. min-AHP-indices-1.5x
27. min-AHP-indices-3.0x
28. peak-index-1.5x
29. peak-index-3.0x
30. peak-indices-1.5x
31. peak-indices-3.0x
32. peak-t-1.5x
33. peak-t-3.0x
34. peak-time-3.0x
35. steady-state-hyper-1.5x
36. steady-state-voltage-1.5x
37. steady-state-voltage-stimed-1.5x
38. threshold-index-1.5x
39. threshold-index-3.0x
40. threshold-t-1.5x

41. threshold-t-3.0x
42. upstroke-index-1.5x
43. upstroke-index-3.0x
44. upstroke-t-1.5x
45. upstroke-t-3.0x
46. voltage-after-stim-1.5x
47. voltage-after-stim-3.0x