FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

ODYNN : An Optimization Suite for Biological Neural Circuits

ODYNN : Optimization for Dynamic Neural Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Marc Javin
Matrikelnummer 1637645

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat Radu Grasu
Betreuung2: Dott.mag. Ramin M. Hasani

Wien, 3. Oktober 2018

Marc Javin	Radu Grasu

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

ODYNN : An Optimization Suite for Biological Neural Circuits

ODYNN : Optimization for Dynamic Neural Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Marc Javin
Registration Number 1637645

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat Radu Grasu
Advisor2: Dott.mag. Ramin M. Hasani

Vienna, 3rd October, 2018

Marc Javin

Radu Grasu

Erklärung zur Verfassung der Arbeit

Marc Javin
Schäffergasse 2, 103 1040 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Oktober 2018

Marc Javin

Acknowledgements

I would like to thank my supervisors **Prof. Radu Grosu**, and especially **Mag. Ramin M. Hasani** for giving me the opportunity to work on this project, for motivating and helping me when I needed, and for the hours of exchange and discussion. Because of our strong cooperation, the pronoun *we* will be used throughout this document and will refer to both Ramin and I.

I am grateful to **Manuel Zimmer** for providing calcium imaging data and insights about neurosciences, in particular on C. elegans.

I am also thankful to my supervisor from my home university **Guillaume Beslon** for advising me and answering my numerous questions.

Finally, I want to deeply thank **Aina Pons** for her unconditional and continuous support.

Kurzfassung

Um eine Skalierbarkeit von modelierten neuronalen Schaltkreisen mit detalierten physikalischen Eigenschaften zu ermöglichen, werden neuartige Optimierungswerkzeuge benötigt. In dieser Arbeit, entwickeln wir eine Hybrid Methode um die Parameter von neuronalen Modellen für gegebene Objekte in einer überwachten Lernumgebung anzupassen. Die entwickelte Methode basiert auf Algorithmen der zufällig initialisierten stochastischen Gradientenverfahren.

Wir präsentieren ODYNN, ein Optimierungssystem für neuronale Netze. ODYNN wurde entwickelt für die Simulation und Optimierung biologischer neuronaler Schaltkreismodelle. Diese Modelle beschreiben weitläufiges verhalten neuronaler Prozesse und werden zum Testen neurowissenschaftlicher Theorien verwendet. Durch verschiedene Funktionen, wird es ermöglicht beliebig strukturierte neuronale Schaltkreise zu entwerfen und zu optimieren um spezifisches verhalten zu bestimmen. Diese Funktionen beinalten unter anderem der Möglichkeit Experimente mit verschiedenen realistischen Modellen, als auch künstlich wiederkehrenden neuronalen Netzwerken (insbesondere Long Short Term Memory) durchzuführen. Des Weiteren ist es möglich Kalziumwerte einfließen zu lassen.

Für eine gegebene neuronale Struktur und einem gegebenem neuronalen bzw. synaptischen Model adaptiert ODYNN zuerst einen Random-Search Algorithmus und anschließend einen Adaptive-Momentum Lernalgorithmus, der auf dem Gradientenverfahren basiert um die Kalziumwerte der gewünschten eingegebenen/ausgegebenen Prozesse.

Des Weiteren führen wir Optimisierungen einzelner Neuronen durch für biophysikalisch realistische, komplexe Neuronmodelle durch. Diese werden durch Partielle Differentialgleicheungen realisiert und untersuchen den Parameterraum um auf die neuronalen Prozesse auf zelulärer Ebene zu folgern. Darüber hinaus optimieren wir die synaptischen und neuronalen Parameter in kleinen neuronalen Schaltkreisen mithilfe des hybriden optimierungs Ansatzes und können so die Leistung der Lernmodell zur Formulierung beliebiger Prozesse einschätzen. Außerdem veranschaulichen wir, dass es mit ODYNN möglich ist, effizient, innerhalb einer angemessenen Optimisierungsdauer für größerer neuronaler Schaltkreis Parameterräume zu skalieren. Wir optimierern neuronale Schaltkreise, wie den “Tap-Withdrawl”, ein neuronales System basierend auf dem Nervensystem eines Erdwurms *C.elegans*, welches reflexartige Reaktionen als Antwort auf mechanische Stimulationen induziert. Als auch den zur Fortbewegung des *C. elegans* Nematode, welcher

verantwortlich ist um die Wanderwellen der Muskeln zu generieren, mit denen der Wurm sich fortbewegt.

Abstract

Modeling neural circuits with detailed biophysical properties requires novel optimization toolkits to enable scalability. In this study, we develop a hybrid method based on random-initialization of stochastic gradient descent algorithm to tune the parameters of neural circuit models, for a given objective in a supervised learning setting.

We present ODYNN, an optimization suite for dynamic neural networks. ODYNN is designed for simulating and optimizing biological neural circuits to model multi-scale behaviors exhibited by the neural dynamics, and to test neuroscience related hypotheses. It is enhanced with features such as performing experiments with different biophysically realistic neuronal models as well as artificial recurrent neural networks (in particular Long Short Term Memory), incorporating calcium imaging data, to design arbitrarily structured neural circuits and optimizing them to govern specific behaviors.

For a given neural circuit structure, a given neuronal and synaptic model, ODYNN adopts a random search optimization step followed by an adaptive-momentum gradient-based learning algorithm to learn the calcium imaging data of desired input/output dynamics.

We perform single neuron optimizations for biophysically realistic complex neuron models which are realized by partial differential equations, and explore in their parameter space to reason about neuronal dynamics at cell level resolution. We then perform synaptic and neural parameter optimization in small neuronal circuits, by the hybrid optimization approach and assess the performance by learning models to express arbitrarily chosen dynamics. Furthermore, we demonstrate that ODYNN is able to scale up to the tuning of larger neural circuits' parameter spaces, efficiently, in a reasonable optimization duration. We optimize neural circuits such as the tap-withdrawal, a neural system from the nervous system of the soil worm, *C. elegans*, that induces reflexive response as a result of mechanical input stimulations; and also the forward locomotion circuit of the *C. elegans* nematode which is responsible for generating traveling waves into the muscle cells to generate the worm's forward crawling.

Contents

CHAPTER 1

Introduction

One of the objectives of Neuroscience is to study and understand the emergent properties of neurons and neural circuits. For this purpose, Caenorhabditis elegans (*C. elegans*), a tiny worm of about 1 mm long, is of great interest. Indeed, due to its relatively simple cell structure, it has been the first multicellular organism with a completely sequenced genome, and is still the only one with a fully mapped connectome (neural connections) [62]. *C. elegans* is most likely the world's best-understood animal [8]. Its nervous system consists of 302 neural cells and about 8000 synapses [61]. It expresses ability for governing complex behaviors [63, 13, 54, 35], which makes it an attractive model organism to investigate how behavior emerges from a small brain. The worm's nervous system has been analysed through multi-scale behavioral and computational experimentations [29, 50], and attempts on modeling its emergent behavior have been conducted through the openworm project [59]. Studies suggest that there exists an important need for a neuron modeling toolkit which includes electro-physiological properties of the neural circuits [50, 37].

Computational models of nervous systems based on neuronal models, such as Hodgkin-Huxley [27], enable to perform neuronal behavior simulations while reasonably approximating biophysical dynamics of cells and circuits. A key challenge in such systems is handling large scale model parameters when simulating larger circuits. The reason for this challenge is the lack of experimental data [22, 40] to reduce the parameter space. Indeed, despite the transparency of *C. elegans*, measuring biophysical properties experimentally is not a trivial task [14].

It is therefore of most importance for such simulation infrastructure to be supplied by an optimization suite to handle the large scale parameter spaces of the detailed nervous system models. Such optimizations have been performed on single neurons using evolutionary strategies [14, 60] or hybrid policies [11]. However, they are often specialized in spiking neurons, while the majority of *C. elegans* neurons are shown to be non-spiking

[29]. Furthermore, they are not well-suited for high dimensional parameter spaces, i.e. neural circuit optimization.

In the present study we propose the use of stochastic gradient descent algorithms [31] on neuron and circuit optimization tasks. We develop a python API, which we call *ODYNN*, implemented with state-of-the-art deep learning toolkits such as Tensorflow [4].

Currently, there exist relevant neuronal circuit simulators for the nervous system modeling. These softwares include NEURON [25], GENESIS [28] and Brian [21]. They supply various models and permit efficient simulation of complex neuronal circuits. ODYNN API, in addition to providing the ability to simulate circuits with various neuronal models, offers - similarly to BRIAN - a simple way to use user-defined models, and enables the use of gradient based optimization by defining differentiable cost functions to address the scaling issue of neuronal circuit optimization. We evaluate ODYNN's simulation and optimization performance by fitting neurons with calcium imaging data, and study the abilities to recover biophysical parameters with noised simulated data among different models. We also explore the feasibility of tuning neural circuits of various sizes. In particular, we obtain promising results when optimizing the Forward Locomotion circuit of C. Elegans.

On a higher level of abstraction, we train artificial recurrent neural networks, Long Short Term Memory (LSTM) [26] to reproduce biological traces, and analyze their integration in neural circuits.

Such optimization toolkit enables exploration of the neuronal dynamics within neural circuits. Once one can design better neuronal models, biological experiments can be performed *in silico*, thus dramatically reducing the costs and time of these processes.

The work is structured as follows: In chapter 2 we give an overview of previous works on optimizing biophysical neural models. In chapter 3, we describe those models and present the ones we utilize. We then discuss our solutions for simulation and optimization, and present the ODYNN API. We represent the result of our experimentations in chapter 4 and, finally, we conclude our work in chapter 5.

CHAPTER 2

State of the Art

Optimizing Biological Neural Networks (BNN) is a challenging task due to the high degrees of nonlinearity and the large parameter spaces. Several approaches have been experimented for this purpose, namely, hand tuning, parameter space exploration, evolutionary strategies, bifurcation analysis and gradient-based techniques.

We introduce in this chapter precedent works and their strategies in a concise way.

When studying simple models or well known neurons, i.e. when the number of unknown parameters is in a reasonable range, one can adopt hand-tuning [56, 43] based on analytical solutions and biological insights. The advantage of this approach is that, unlike algorithms, one quickly gets insights about the dynamics of the system given a parameter set. This method is also reasonable when the objective is to explore the evolution of behavior depending on the parameter values [44]. It is also straight forward to implement, since no code needs to be written and brings fast feed back. However, as the number of free variables grows, hand-tuning quickly becomes unfeasible.

Parameter space exploration uses computational power to automatically evaluate different sets of parameters. It can be effected using a regular grid or random selection. This can be useful for understanding the role of some particular variables [16], discovering potential behaviors of a model, or classifying regions with respect to their outputted behaviors [11, 47]. With this technique, one can characterize the space and gain more insights on the expressive power of neural models [46, 20]. The main shortcoming is an intensive computational cost, which exponentially increases with the amount of trainable parameters.Moreover, for a neural circuit model as a dynamic system, one has to take care of unstable regions which can easily get picked up by the search algorithm. On the opposite, regions of interest might be missed.

More recently , evolutionary strategies such as genetic algorithms have been deployed in the neuronal circuit optimization [60, 30, 6, 14]. Those methods are computationally efficient and perform often very well at finding global optima. Nevertheless, they

require choosing numerous algorithm parameters whose impact on the result can be significant. Furthermore, they require one evaluation per individual, which can become computationally very expensive if not parallelized.

Bifurcation analysis uses the mathematical theories on nonlinear dynamical systems to map global neural dynamics to the parameter space. This permits the localization of subspaces producing a desired dynamic [23]. It can also provide a definition of the parameter space by identifying the transition regions between two types of activities [9]. Again, this approach becomes very costly with a high dimensional space, but mostly does not take any time information into account.

Gradient-based optimization techniques (such as the back-propagation algorithm [52]) has recently showed tremendous success in finding local optimal solutions for artificial feed-forward and recurrent neural networks [41, 55, 58]. For more biophysical-based neural networks however, they have not been used in large scale yet. This is due to the complexity of the biophysical models and their intrinsic and external feedback mechanisms, which makes the gradient propagation paths non-homogeneous and therefore difficult to train. Moreover, biophysical neuronal models are a subset of recurrent neural networks which have shown to face difficulties during training due to the vanishing and exploding gradient effects [10, 45].

Bhalla and Bower used gradient descent as a second step of a hybrid strategy to identify good parameter sets for multi-compartment models of mitral and granule cells of the olfactory bulb[11]. In order to calculate a gradient, they randomly sampled points for several parameters. As a matter of fact, a disadvantage of this approach is the need to determine the direction in parameter space in which the cost function decreases. Moreover, there is high risk of getting stuck into local optima, especially when the objective function is steep, which might be the case for such dynamic systems as BNNs. However, their key advantage is their computational speed.

In this study, we show that with a careful initialization of a complex neuronal model and a good design for the learning setting, one can tackle the gradient issues of a back-propagation methodology for neuronal circuits and easily scale it to relatively large neuronal circuits' optimization.

We adopt Adam [31] (for adaptive moment estimation), a recent stochastic gradient descent algorithm, for the first time for the optimization of complex neuronal circuits. We equip our gradient-based optimization technique with a random search to get into better parameter space regions. Random search strategies themselves have recently shown competitive performance to their gradient-based counterpart [64, 53, 38]. We take advantage of their performance as an initial step before performing gradient decent to further improve the quality of the learned biophysical neural networks.

CHAPTER 3

Methods

In this chapter we introduce our step-by-step methodology to design our neuronal circuit optimization suite. In Section section 3.1, we motivate the design and optimization of neural circuits. In section 3.2, we introduced the existing neural and synaptic models, and present the ones we selected for experiments shown in chapter 4. We then discuss the need and challenges of optimization (section 3.3) and simulation (section 3.4) of such circuits, together with our choices in these concerns. We finally describe *ODYNN* API in section 3.5.

3.1 Neural circuit simulation and optimization

One of the key question of neuroscience is how complex behaviors emerge from neural systems. For instance, C. elegans, with only 302 neurons, is capable of expressing behaviors such as crawling, mating or food seeking [63, 13, 54, 35]. The worm's nervous system has been analysed through multi-scale behavioral analyses [29]; [50], and attempts on modeling its emergent behavior have been conducted in the OpenWorm project [59]. For the latter, there is an important need for a neuron modeling toolkit to include electro-physiological properties of the neural circuits [50, 37]. As explained in Section 3.2, the biophysical neural models possess a great amount of free parameters, creating a need for a suitable optimization suite to handle large parameter space. We aim at developing such a toolkit to enable models to reproduce behaviors being observed in living animals, and to test neuroscientific hypotheses.

The ultimate goal of the work we commence is graphically shown on figure 3.1. We aim at providing a tool capable of reproducing a given neural circuit behavior, at the condition of providing enough measurements from experiments. This tool should be able to provide neuronal and synaptic models, and their optimizations.
Achieving this goal would allow neuroscientists to perform experiments *in silico*, thus drastically reducing the cost and time required compared to *in vitro* experiments. This

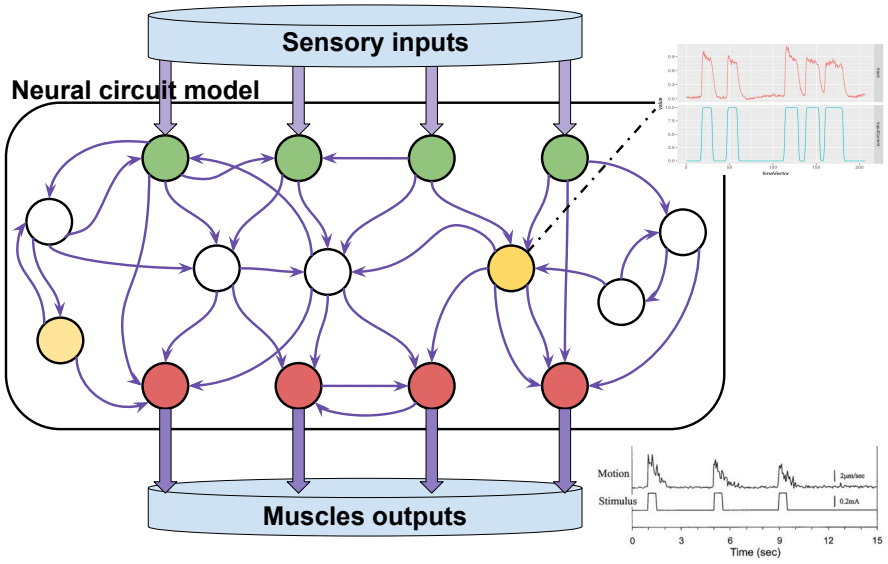


Figure 3.1: Overview of the work: Given a neural circuit which architecture is known, model this circuit and, using recordings of sensory inputs, muscle and neuron activities, search in the parameter space to reproduce the circuit's observed behaviors.

would also lead to a greater understanding of neural circuits dynamics. Throughout this chapter, we present our chosen methods to reach this objective.

3.2 Neuron and synapse models

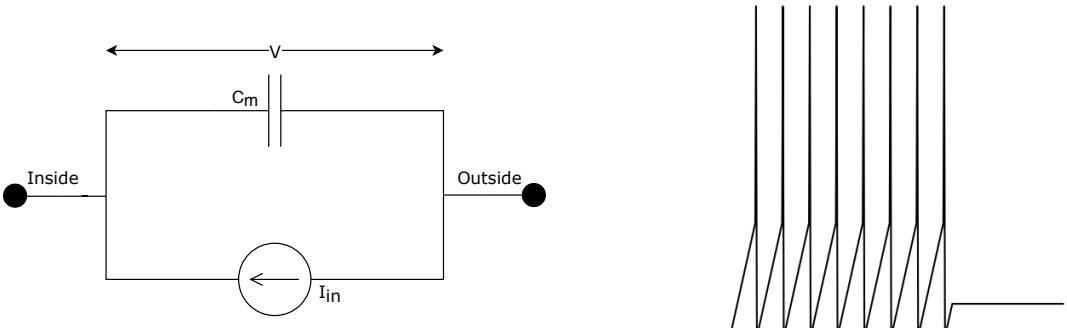
We explain here how we model neuronal circuits. First, we present the most common conductance-based models for biophysical neurons and the ones we select, and then describe the synaptic transmission models.

Within a neural circuit, information underlies the state of the individual neurons, i.e. their membrane potentials, ion concentrations and the state of their ion channels. A neuron is delimited by its membrane. At each of its side are accumulated ions of different types and electrical charges which create an electrical potential. Various ion channels are distributed along the membrane and allow the ions to flow when they are open and activated. These flows create electrical currents and modify the neural potential. Neurons interact with each other via synapses generating currents which influence their potentials. Several mathematical models have been developed reproducing neural dynamics up to different degrees.

3.2.1 Conductance based neuron models

This subsection describes well-known conductance-based models for neurons. We explain their principles to familiarize the reader with biophysical models.

Integrate and Fire [5]



(a) Electrical representation

(b) Typical voltage trace

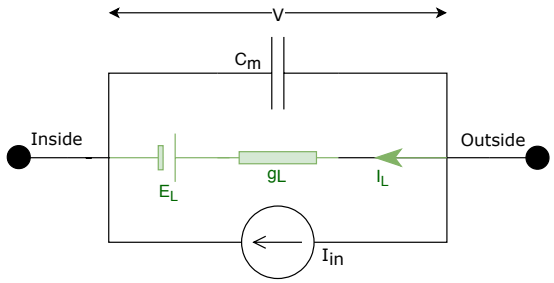
Figure 3.2: Integrate and Fire model

Integrate and Fire is one of the simplest and earliest models, developed in 1907 [5]. It can be visualized in figure 3.2a. The neuron membrane is represented as a capacitance C_m, on which is applied an input current I_{in}, resulting in a change of the membrane potential as follows [33]:

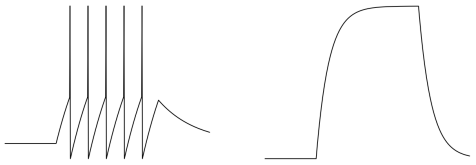
$$C_m \frac{dV(t)}{dt} = I_{in}(t) \tag{3.1}$$

where C_m is the membrane capacitance, V the membrane potential and $I_{in}(t)$ the input current at time t.

This comes together with a certain threshold, V_{thr}, at which a delta function spike occurs, the voltage is then reset to a certain resting value V_{rest}. An example is shown in figure 3.2b.
The model was developed after observing that a substantial number of neurons are spiking (sudden peak of voltage following a strong enough stimulation). It takes advantage of the fact that action potentials are very similar among different spiking neurons, and with time within a particular one. Nevertheless, its main limitation is the lack of time memory, since a potential reached below the threshold V_{thr} will remain even without any input current (cf. figure 3.2b), which doesn't happen in reality.



(a) Electrical representation



(b) Voltage trace with threshold V_{thr} (Leaky Integrate and Fire)

(c) Voltage trace without threshold V_{thr} (Leaky Integrate)

Figure 3.3: Leaky Integrate (and Fire) model

Leaky Integrate and Fire

As highlighted in figure 3.3a, Leaky Integrate and Fire adds a conductance g_L in parallel with the capacitor to overcome the time-memory problem. This conductance models ion channels generating currents by letting ions flow through the neuron's membrane. A generator E_L represents the reversal potential of the membrane, i.e. the resting potential. We then obtain a classical RC circuit, leading to the following equation [33]:

$$C_m \frac{dV(t)}{dt} = I_{in}(t) + g_L \cdot (V(t) - E_L) \tag{3.2}$$

or equivalently,

$$\tau_m \frac{dV(t)}{dt} = R_L I_{in}(t) + (V(t) - E_L) \tag{3.3}$$

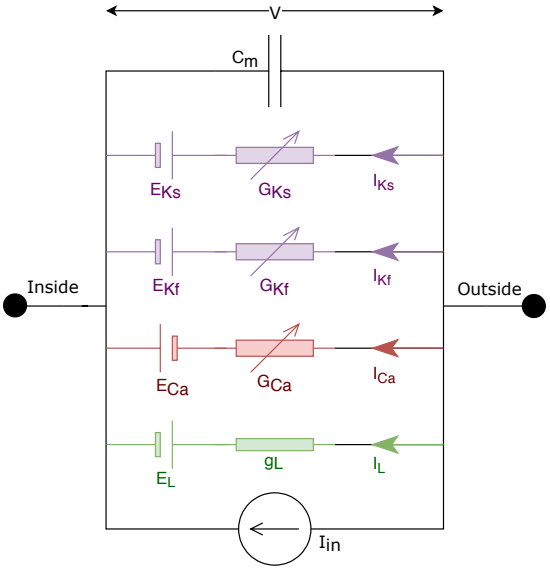
τ being the time constant $R_L.C_m$ and R_L the channel resistance.

Leaky Integrate

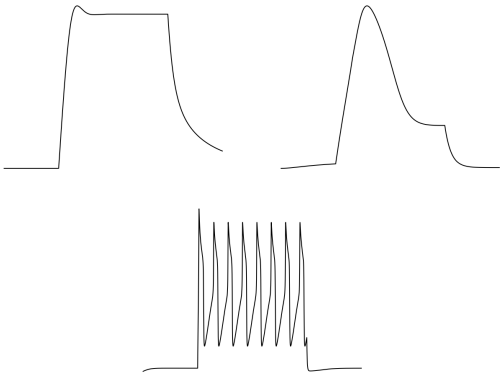
Leaky Integrate keeps the previous modeling of the neuron membrane. However, no voltage threshold for firing is used. The effect of this threshold is exhibited in figures 3.3b and 3.3c. This permits modeling of non spiking neurons, such as the majority of neuron classes found in C. elegans [29].
This model has shown good performances for imitating global real neuron behaviors.

Hodgkin Huxley [27]

As in the Leaky Integrate and Fire model, the membrane is represented by a capacitor, in parallel with a conductance. Nevertheless, in addition to the passive leaky conductance g_L, several conductances are added in parallel, combined with their associated reversal potentials. These new conductances model different types of ion channels, and typically vary with time and voltage, thus leading to a greater complexity.
We show in figure 3.4a an example of a Hodgkin Huxley model which we will use later.



(a) Electrical representation



(b) Possible voltage traces

Figure 3.4: Integrate and Fire model

Three additional channels are used : G_{Ca}, G_{Ks} and G_{Kf}. Unlike g_L, those conductances are not static and vary with time and voltage, we explain how in the next subsection. Like previously, we obtain :

$$C_m \frac{dV}{dt} = I_{in}(t) + \sum_c I_c(t, V) \tag{3.4}$$

where I_{in} is the input current and the I_c's are the currents at the different conductances.

The Hodgkin Huxley model is very powerful and allow a precise modeling including real biophysical parameters. The dynamic conductances results in a wide range of possible behaviors depending on the chosen values for the parameters, we display some of them in figure 3.4b .

3.2.2 Neuronal models developed in our experimentations

To enable fitting of real neural data, we need a first realistic model. Also, to investigate the influence of the model complexity on optimization, we develop two simpler models. Finally, we create a neuron model based on LSTM networks. We describe here these models.

C. elegans Hodgkin Huxley

Because we want to fit real biological data, we need a complex and accurate model of C. elegans neuron. Models such as the Fitzhugh-Nagumo [15] or the Morris-Lecar model [42] (not described here) would not be sufficient, since such phenomenological models do not provide a sufficient level of detail for the inner neuron mechanisms.

Active currents are found in C. elegans neurons, hence justifying the use of a Hodgkin-Huxley type model [22].

Due to the small size of the C. elegans neurons, a single compartmental model is sufficient, in accordance with earlier modeling studies [22][63].

We select a rather simple Hodgkin Huxley model used by OpenWorm [59], which is an open source project aiming at modeling C. elegans. The model is displayed in figure 3.4a. It is composed of a leak channel together with two sodium (fast and slow) and one calcium channel :

1. Leak channel for potassium, with constant conductance :

$$I_L(t) = g_L \cdot (V - E_L) \tag{3.5}$$

2. 'Slow' potassium channel :

$$I_{ks}(V,t) = n(t,V) g_{ks} \cdot (V - E_K) \tag{3.6}$$

3. 'Fast' potassium channel :

$$I_{kf}(V,t) = p(t,V)^4 q(t,V) g_{kf} \cdot (V - E_K) \tag{3.7}$$

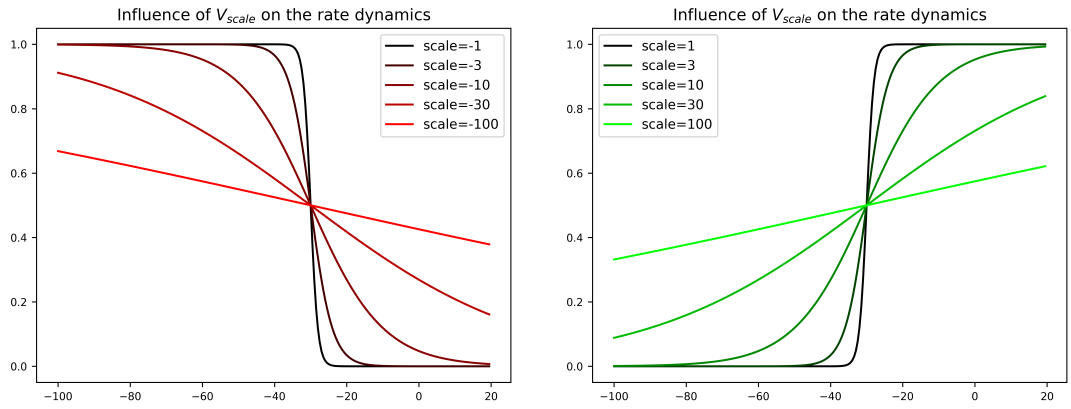
4. Boyle calcium channel :

$$I_{ca}(V,[Ca^{2+}],t) = e(t,V)^2 f(t,V) h([Ca^{2+}]) g_{Ca} \cdot (V - E_{Ca}) \tag{3.8}$$

The parameters n, p, q, e, f are voltage and time dependent activation rates (h varies with the calcium concentration), they represent the activation and opening of ion channels. At the exception of h, they behave following equations (3.9) and (3.10).
At a given voltage, one can compute their steady state in the form :

$$x_\infty = \frac{1}{1 + e^{-\frac{V - V_{mid}}{V_{scale}}}} \tag{3.9}$$

where the parameters V_{mid} and V_{scale} are called respectively the midpoint and the scale. The steady state is thus a sigmoid centered on V_{mid}, where V_{scale} controls the slope of the curve. The midpoint is the voltage at which a subunit is half open. The scale controls the sensitivity as shown in figure 3.5, a positive scale leads to a subunit opening as the



(a) Negative values

(b) Positive values

Figure 3.5: Influence of the scale on rate steady states. Examples with a midpoint of -30 mV and different values for the scale.

voltage increases, a negative one induces the opposite behavior. A high value for the scale gives raise to a very progressive slope, while a small one results in a fast and steep evolution around the midpoint.

The rates tend to their steady state following :

$$\frac{dx}{dt} = \frac{x - x_\infty}{\tau_x} \tag{3.10}$$

where τ_x is their time constant.

Thus, for each of those 5 rates, three parameters have to be defined.

The variable h follows a different law (Eq. (3.11) and (3.12)). It depends on the sole calcium concentration :

$$q = \frac{1}{1 + e^{-\frac{[Ca^{2+}] - [Ca^{2+}]_{mid}}{[Ca^{2+}]_{scale}}}} \tag{3.11}$$

$$h = 1 + (q - 1)\alpha \tag{3.12}$$

Here we have again three parameters : $[Ca^{2+}]_{mid}$, $[Ca^{2+}]_{scale}$ and α

In order to fit calcium imaging data, the calcium concentration is modeled by a simple pump system :

$$\frac{d[Ca^{2+}]}{dt} = I_{Ca} * \rho_{Ca} - \frac{[Ca^{2+}]}{\tau_{Ca}} \tag{3.13}$$

with parameters ρ_{Ca} and τ_{Ca}.

In total, with the 3 parameters of each rate, the reversal potential of the 3 dynamic channels, the maximum conductance of the 4 channels, the calcium pump and the membrane capacitance, this model contains 28 parameters.

Simple Hodgkin Huxley

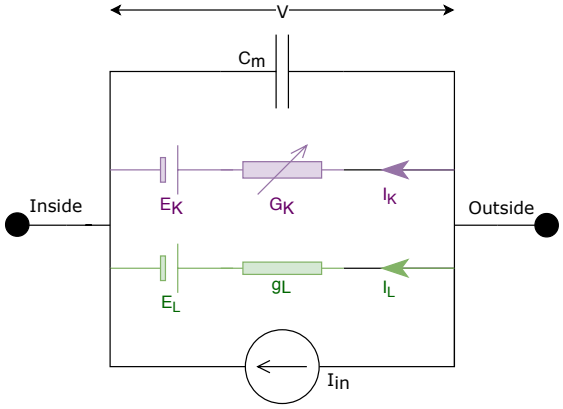


Figure 3.6: Simple Hodgkin Huxley model

In order to study the impact of the model complexity, we select an arbitrary Hodgkin Huxley model with a single dynamic channel and no concentration model :

$$C_m \frac{dV}{dt} = I_{in}(t) + a^3 b g_K \cdot (V - E_K) \tag{3.14}$$

where a and b follow the same voltage dependent behavior described in (3.9) and (3.10). This model is represented in figure 3.6 and possesses 11 free parameters.

Leaky Integrate

Finally, we use a simple Leaky Integrate model, as its number of free parameter is small : the membrane capacitance C_m, conductance g_L and reversal potential E_L sum up to 3 tunable parameters per neuron. This is the simplest model we could use.

Long Short-term Memory [26]

We also study the possibility of using artificial Recurrent Neural Networks (RNN) to mimic biophysical neural behavior (i.e., using artificial neural networks for modeling single biological neurons). LSTM, standing for Long Short Term Memory [26], are gated RNNs that have shown tremendous performance in sequence modeling [57][18]. It might be of interest to replace a neuron or a subpart of a circuit by such "black boxes". Indeed,

they have proven efficient computation and powerful diversity of possible behaviors. LSTM cells have been designed to allow long-term dependency and to tackle vanishing

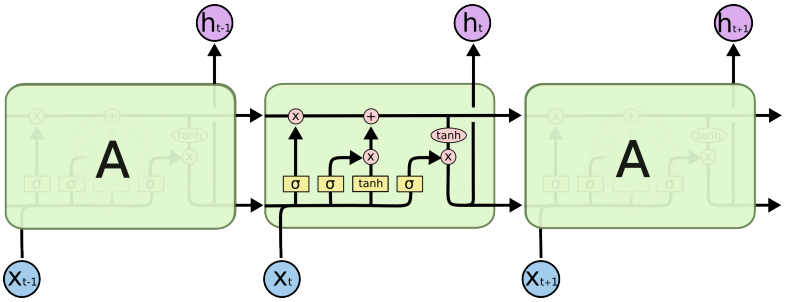


Figure 3.7: Internal working of an LSTM cell. (image source [3])

gradient issues of RNNs and might be easier to optimize than biophysical models.

As shown on figure 3.7, an LSTM cell maintains, for each time step t, an internal state c_t (top horizontal line) and an output h_t (bottom horizontal line), computed based on an input x_t, and previous state c_{t-1} and output h_{t-1}. The cell is composed of a forget gate (first σ on the left), multiplying each part of the state by a number between 0 and 1. Another gate (second σ and $tanh$) is used to add new information to the state. Finally, a last gate (last σ) decides which part of the state should be used for the output.

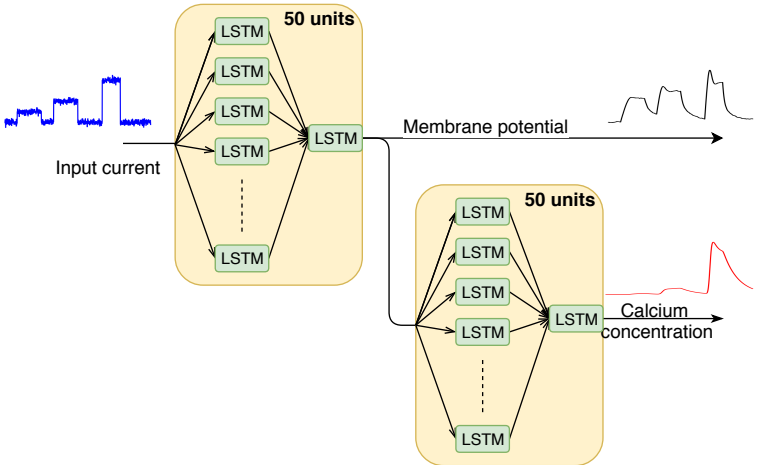


Figure 3.8: LSTM network modeling a biological neuron

To model a single biological neuron, we use a two layers LSTM network, depicted in figure 3.8. Each layer is composed of 50 LSTM cells. This model is meant to be equivalent to the *C. Elegans* Hodgkin Huxley model we describe in 3.2.2. We interpret the output of the first layer as the membrane potential, and the second one as the calcium concentration, since in reality the calcium concentration depends on the membrane potential. In both cases, because the output from an LSTM layer is contained in [-1,1], we scale its output

in the following way :

$$V = 100 * out_1 - 60 \tag{3.15}$$

$$[Ca^{2+}] = 1000 * out_2 \tag{3.16}$$

This way, we obtain a voltage range of $[-160, 40]$ mV and $[-1000, 1000]$ for the calcium concentration. We thus have a resting value of -60 mV and 0 Mol, as in the biophysical model.

3.2.3 Synapse models

Within a neural network, information propagates between neurons via chemical and electrical synapses. We explain here how to model such transmission.

Chemical synapse

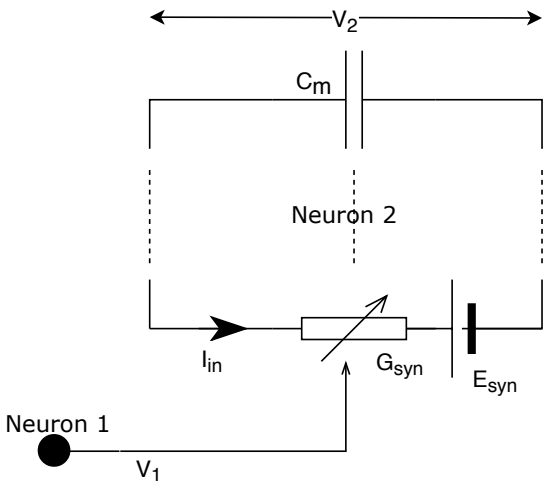


Figure 3.9: Chemical synapse model

At a chemical synapse, two neurons interact with each other by the release of neurotransmitters, from the pre-synaptic to the post-synaptic neuron. The synaptic current can be represented as [33]:

$$I_{i,j} = G_{syn}(V_j) \cdot (E_{rev} - V_i) \tag{3.17}$$

where E_{rev} is the resting potential of the synapse which determines its polarity, and $G_{syn}(V_j)$ the dynamic conductance of the synapse.

As shown in figure 3.9, the chemical synapse is modeled by an additional ion channel in the post-synaptic neuron, which conductance $G_{syn}(V_j)$ varies with the pre-synaptic neuron's potential, following the same law as in (3.9) [33]:

$$G_{syn}(V_j) = \frac{g_{i,j}}{1 + e^{\frac{V_j - V_{mid}}{V_{scale}}}} \tag{3.18}$$

where $g_{i,j}$ is the maximum conductance of the synapse between neurons j and i.

Such a model has 4 free parameters : $w_{i,j}$, V_{mid}, V_{scale} and E_{rev}.

Electrical synapse (gap junction)

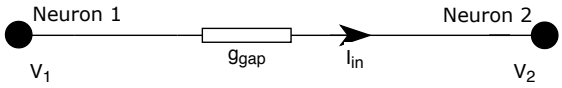


Figure 3.10: Electrical synapse model

A gap junction between neuron j and i is simply modeled as a static conductance (figure 3.10), hence using Ohm's law, where $g_{i,j}$ is the conductance of the synapse :

$$I_{i,j} = g_{i,j} \cdot (V_j - V_i) \tag{3.19}$$

Thus, it only needs the definition of one free parameter $g_{i,j}$.

3.3 Optimization

Once we modeled a neuronal circuit with neural and synaptic models, we need to assign values to the models' parameters. We want to find the values so that the model reproduces the behavior of the original circuit. For this, we use measurements that the model's simulation will have to fit.

Because the change of any single parameter can have a high impact on the final outcome, we need an optimization technique. Several methods exist and have been applied to neural models, such as genetic algorithms [30] [60] [6]. Although they have expressed reasonable performance in small circuits with few neurons and synapses, they have shown difficulty to scale, as the number of parameters increase. For this matter, we propose to use stochastic gradient descent optimization. These algorithms are now widely used in the field of Artificial Intelligence, in particular for Deep Learning. Their main advantage is a low computational cost, at the price of a higher risk to get stuck in a local minima, and the necessity to compute a gradient. To overcome these drawbacks, we use a hybrid technique by associating random search with gradient descent.

3.3.1 Hybrid optimization

As described by algorithm 3.1, we opted for a hybrid optimization strategy. After defining models and a circuit architecture, we initialize several sets of the circuit's parameters with random values. We then apply gradient descent to each of those sets in parallel. While this parallelization requires a larger amount of memory, this allows to increase the speed of computation and chances to find better sets of parameters. At every step, the cost function (defined in 3.3.4) is evaluated and its gradient is computed via differentiation. The cost function represents the distance towards the objective and should be minimized.

Algorithm 3.1: Hybrid Optimization

Input: $f(\theta)$: a cost function with parameter θ
C : a set of constraints for parameter θ
Output: Sets of parameters Θ minimizing the cost function

$\Theta \leftarrow$ random population of parameters for f of size n;
while not *Maximum-Iterations reached* **do**
  for $i \leftarrow 1$ **to** n **do**
    $loss_i \leftarrow$ f($\Theta[i]$) ; /* eq (3.20) */
    $grad_i \leftarrow$ compute_gradient(f,$\Theta[i]$) ; /* cf. algorithm 3.2 */
    $grad_i \leftarrow$ clip_gradient($grad_i$);
    $\Theta[i] \leftarrow$ apply_gradient($grad_i$, $\Theta[i]$) ; /* cf. algorithm 3.2 */
    $\Theta[i] \leftarrow$ apply_constraints(C, $\Theta[i]$);
  end
end
return Θ;

We limit the norm of the gradient to a maximum value to avoid brutal jumps in the parameter space. Using this gradient, the parameters are updated following Adam algorithm [31] (see 3.3.5). Finally, we apply constraints to our parameters. For this purpose, we implement hard constraints as explained in 3.3.2

3.3.2 Parameters constraints

Hard constraints forbid parameters to take values outside of defined ranges. They are appropriate to prevent values leading to unexpected or exploding outcomes in the simulation. Hence, too small absolute values for parameters such as time constants τ_i, scales V^i_{scale} and membrane capacitance C_m lead to instability since they are used as denominators in the models' equations. Furthermore, some parameters like C_m or any conductance g_i have to be positive. Hard constraints also permit keeping the parameters in a restricted range when knowledge is available on the given variables. For this last purpose, soft constraints might perform better, as hard constraints can prevent the gradient to keep its global direction. Indeed, soft constraints allow their violation. However, they add a penalty in the cost function, thus making it more complex, and adding parameters to define. Furthermore, with gradient descent and high dimensional problem, it is not guaranteed that the final outcome does not violate such constraints.

3.3.3 Random initialization

Each parameter starts with a value drawn from a uniform distribution. The bounds will be displayed in tables for each experiment. We select large ranges containing the values used by OpenWorm [59] and respecting the defined constraints. We select those high

ranges in order to evaluate our methods' performance when few or no insight is available on the biophysical properties.

3.3.4 Cost function

For evaluating the quality of a model, one needs a function defining a loss, i.e. how distant this model is from its objective. The goal of the optimization will be to minimize it. Different kinds of cost functions have been used for biological neuron parameters optimization. Mainly, the overlap in $\frac{dV}{dt}$ versus V phase plane [6] ignores all time information and feature extraction [11] is problematic since it is not differentiable. We hence use a weighted Mean-Squared Error. This function is simple, differentiable and fast to calculate. We add a possibility of weighting the neurons and the different measurements (voltage and calcium concentration in our case).

$$Cost(\theta) = \frac{1}{NT} \sum_{m} w_m \sum_{n=0}^{N} w_n \sum_{t} d_{m,n}(t,\theta)^2 \tag{3.20}$$

where N is the number of neurons and T the number of time steps. w_m represents the weights associated to the different measurements (e.g. voltage, calcium concentration), w_n the weight of each neuron, and $d_{m,n}(t)$ is the difference between the measurement m for the neuron n at time t of our current model and the objective trace.

3.3.5 Adam

Adam (for Adaptive Moment Estimation) [31] is a stochastic gradient descent algorithm. It stores an approximation of the mean and uncentered variance of the past gradients, respectively m_t and v_t in algorithm 3.2. This results in an adaptive learning rate for each parameter. It performs well in high dimensional problems, is well suited for sparse gradients, and compares favorably to other adaptive learning-method algorithms [51]. This is the algorithm we choose for our gradient descent strategy.

3.3.6 Tensorflow

We deployed our gradient-based optimization algorithms and our networks in Tensorflow [4].

3.4 Simulation

For the computation of the cost function, one requires simulation of neural circuits. Such simulations exist [19][24], nevertheless, a gradient needs to be computed for algorithm 3.1. One could estimate it with sampling [11], yet this implies running numerous simulations for each parameter, thus dramatically slowing down the process. Hence, the most efficient way is to compute the gradient in an analytical manner, i.e. by differentiating the cost

Algorithm 3.2: Adam optimizer

Input: α : Learning rate
β_1, $\beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates]
$f(\theta)$: Stochastic objective function with parameters θ
θ_0: Initial parameter vector
Output: Parameter vector θ minimizing the cost function

$m_0 \leftarrow 0$;
$v_0 \leftarrow 0$;
$t \leftarrow 0$;
while θ_t *not converged* **do**
  $t \leftarrow t + 1$;
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$;
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$;
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$;
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$;
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$;
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$;
end
return θ_t;

function. Once a network is defined as a Tensorflow computational graph, we are enabled to perform such differentiations.

3.4.1 Neuron simulation

In order to simulate a neuron, the differential equations introduced in 3.2 have to be numerically solved. Because of the sigmoid functions in Eq. (3.18) and (3.9) and the equations' interdependence, they are nonlinear and thus not explicitly solvable.
With the approximation

$$\frac{dV}{dt} \approx \frac{V(t + \Delta t) - V(t)}{\Delta t}, \tag{3.21}$$

for a small time step Δt, we use implicit Euler solver for the calcium concentration and rate dynamics and hybrid (implicit-explicit) solver for the voltage. By inserting Eq. (3.21), we get :

$$V(t + \Delta t) = \frac{V(t)\frac{C_m}{\Delta t} + I_{in}(t) + \sum_c g_c \sum_{x_c} x_c^{\alpha_{x_c}} E_c}{\frac{C_m}{\Delta t} + \sum_c g_c \sum_{x_c} x_c^{\alpha_{x_c}}} \tag{3.22}$$

for Eq. 3.3 and 3.4,where for each channel c, g_c is its conductance, x_c a rate with its associated exponent α^{x_c}, and E_c the reversal potential.

$$[Ca^{2+}](t + \Delta t) = \frac{\tau_{Ca}}{\tau_{Ca} + \Delta t} \cdot ([Ca^{2+}](t) - I_{Ca}(t)\rho_{Ca}\Delta t) \tag{3.23}$$

for Eq. 3.13

$$x(t+\Delta t) = \frac{\Delta t \tau_x}{\Delta t + \tau_x}\left(\frac{x(t)}{\Delta t} + \frac{x_\infty(V) + \Delta t}{\tau_x}\right) \tag{3.24}$$

for Eq. 3.10

In this way we obtain incremental equations that we update after each time step Δt. This leads to algorithm 3.3 for the step update of a neuron and 3.4 for a circuit.

Algorithm 3.3: Neuron step

Input: $[v,X,IC]$: Previous state of the neuron containing the voltage v, rates X and ion concentrations IC
I_{in} : Input current
Output: Updated state $[v',X',IC']$

for *c in Channels* **do**
  $I_c \leftarrow$ compute-current(v, $X[c]$);
end
$v' \leftarrow$ update-voltage(v, $\sum I_c$, I_{in}); /* eq (3.22) */
for *ic in IC* **do**
  $ic' \leftarrow$ update-concentration(ic, I_{ic}) ; /* eq (3.23) */
end
for *x in X* **do**
  $x_\infty \leftarrow$ compute-steady-state(x, v) ; /* eq (3.9) */
  $x' \leftarrow$ update-rate(x, x_∞) ; /* eq (3.24) */
end
return *[v',X',IC'];*

Algorithm 3.4: Circuit step

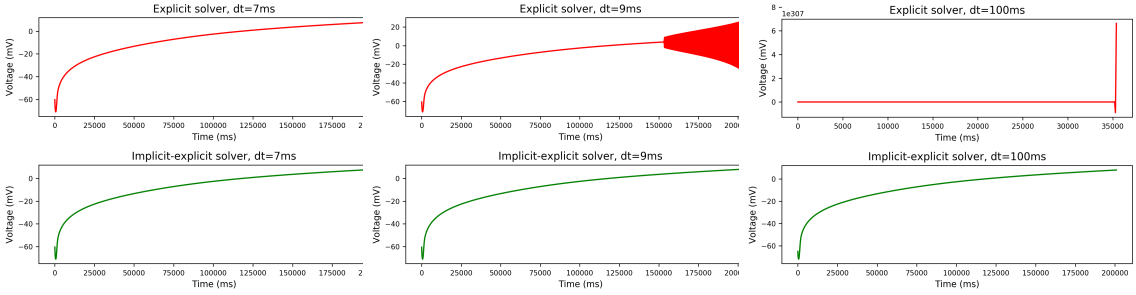
Input: S: Previous states of the neurons,
I_{in} : Input currents
Output: Updated states S'

for $g_{i,j}$ *in GapJunctions* **do**
  $I_{in}[i] \leftarrow I_{in}[i] +$ gap-current$_{i,j}(S[i], S[j])$; /* eq (3.19) */
end
for $s_{i,j}$ *in Synapses* **do**
  $I_{in}[i] \leftarrow I_{in}[i] +$ syn-current$_{i,j}(S[i], S[j])$; /* eq (3.17) */
end
for *n in Neurons* **do**
  $S'[n] \leftarrow$ neuron-step$_n(S[n], I_{in}[n])$; /* alg 3.3 */
end
return S';

The use of an implicit solver is important in case of stiff differential equations. It allows an increased stability and the use of larger time steps. In figure 3.11, we compare our equation (3.22) with equation (3.25) obtained with an explicit solver strategy for the membrane potential :

$$V(t+\Delta t) = V(t) + \frac{I_{in}(t) + \sum_c I_c(t)}{C_m}\Delta t \tag{3.25}$$

As we can observe, given the same input current $I_{in}(t)$, both solvers are sensibly equal for small time steps. Nevertheless, the explicit solver quickly becomes unstable, for $\Delta t = 9$ ms here, and explodes when the time step increases ($\Delta t = 100$ms). On the other side, the implicit-explicit solver stays stable.



(a) $\Delta t = 7$ ms (b) $\Delta t = 9$ ms (c) $\Delta t = 100$ ms

Figure 3.11: Influence of the time step Δt on the voltage simulation for our hybrid solver (Eq. (3.22)) in green and an explicit solver (Eq. (3.25)) in red

3.5 ODYNN API

In this section we present the *ODYNN* API - standing for Optimization of Dynamic Neural Networks - which we developed and used for our experiments. It aims at providing an optimization suite for biological neural circuits. *ODYNN* allows implementation of different neuron models, definition of circuit architectures, simulation and optimization of such circuits, and evaluation of the results. It can be downloaded via github (`https://github.com/MarcusJP/ODYNN`), and a documentation and tutorials are available under `odynn.readthedocs.io`.
In this section we describe its main principles.

3.5.1 Development

Our API has been deployed in Python3 and Tensorflow 1.8, following the best practices for developing a Python package [49]. It has been developed incrementally, fully tested via unit-test and evaluated with continuous integration using Travis [2] and Codecov [1]. It is available on github : `https://github.com/MarcusJP/ODYNN` as a python

package and can be used under MIT license. The documentation was made with Sphinx [12] and can be found at `odynn.readthedocs.io` together with tutorials in Jupyter Notebook [32].

3.5.2 ODYNN's Structure

Figure 3.12 represents the Unified Modeling Language (UML) class diagram of our API. We describe here the main characteristics of its class structure.

We opted for object oriented programming to allow the use and comparison of different models and methods. A first abstract class *Optimizer* (center-right) possesses a method *optimize* (algorithm 3.1) allowing to optimize the parameters contained in an attribute object which class implements the interface *Optimized*. For instance, classes to be optimized have to define an access to their variable (attribute *variables*) and apply their constraints in a method *apply_constraints*. This allows a common representation of all classes vowed to be optimized, thus bringing a proper encapsulation between optimizing and optimized objects. The classes *NeuronOpt* and *CircuitOpt* (below) implement *Optimizer* by defining an appropriate loss (see Eq. (3.20)). As the name indicates, *NeuronOpt* is made for

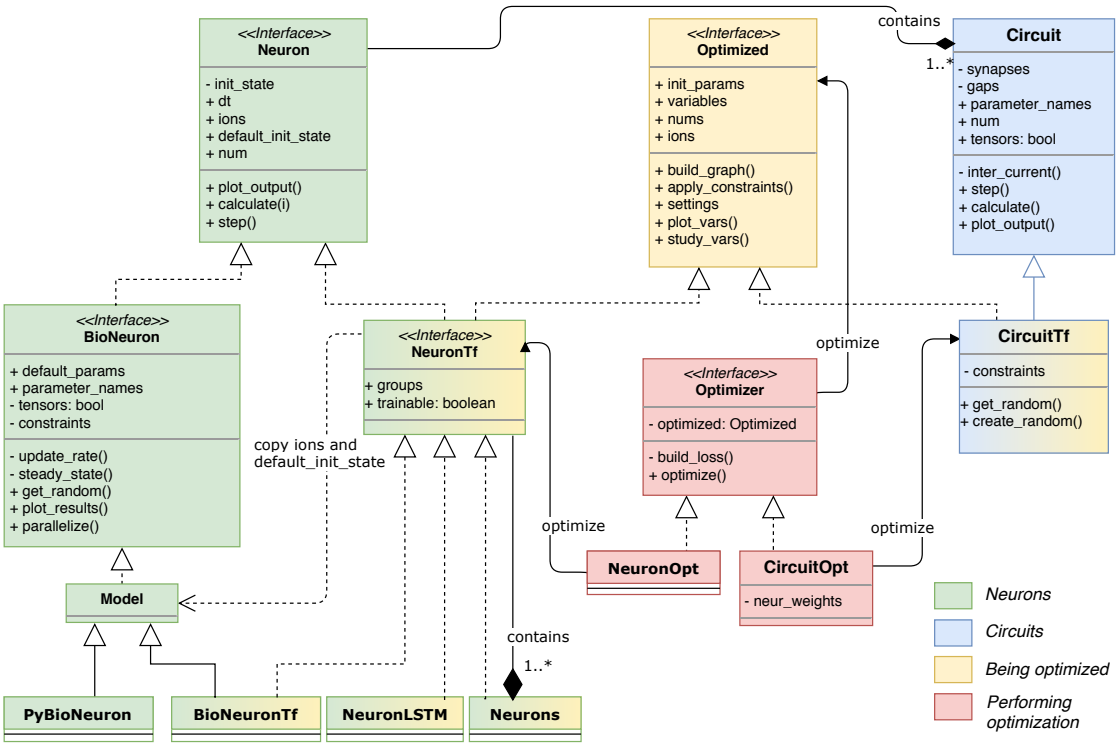


Figure 3.12: ODYNN UML class diagram.

optimizing classes representing neurons.

The left part of the diagram describes the hierarchy of our neural models.

The top left class *Neuron* defines the general structure a neuron class should follow in order to be used in a circuit, mainly defining an initial state (attribute *init_state*), declaring the ions which concentrations are modeled (attribute *ions*), the number of neurons modeled by an instance (attribute *num*, for optimization or usage in a circuit) or implementing algorithm 3.3 in a method *step*. The class *Neuron* also defines a general way to plot outputs of an experiment. Furthermore, the class *BioNeuron* creates a way of implementing a biological model (as presented in subsection 3.2.2). From there, we define classes for each model, here represented as *Model*. A simple configuration file defines which model is used, and two other classes, *PyBioNeuron* and *BioNeuronTf*, specialize this model. The difference between them is that the first is implemented in Python, while the second is fully implemented in Tensorflow, and implements *NeuronTf* (center-left). *PyBioNeuron* exists only for simplicity and simulation, and doesn't allow optimization. The class *NeuronTf* is there to hide the nature of a neuron during optimization, by providing a common interface. At the moment, three different classes can represent neurons and be optimized :

- *BioNeuronTf* for biological models
- *NeuronLSTM* allows the abstraction of a neuron using LSTM cells, presented in subsection 3.2.2
- *Neurons* is used when studying circuits and combines biological and artificial models in a common architecture.

On the right side, the classes *Circuit* and *CircuitTf* are mostly the equivalent to *PyBioNeuron* and *BioNeuronTf*. Their structure is more simple but is probably going to gain in complexity in order to allow the use of different synaptic models, similarly to what is done for neurons.

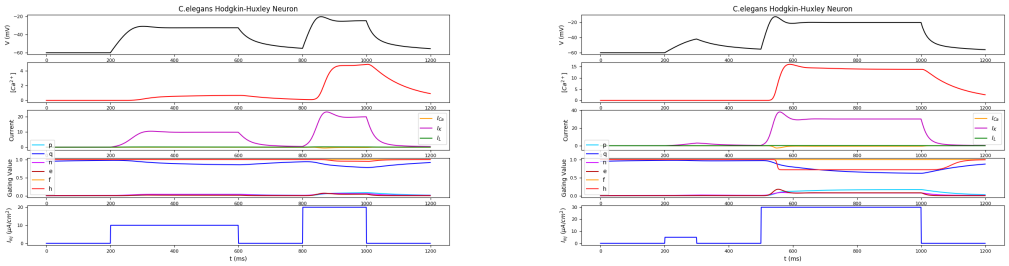
The main strength of our API is its strong encapsulation. The latter gives the advantage of being able to define a new model in a fast way. After creating a new class extending the interface *BioNeuron*, namely defining the *step()* and *get_random()* (for random parameters) methods, the default parameters and initial state, the whole API can be used in the same way, and one can use the exact same code for simulation and optimization amongst different models.

The main challenge of this architecture is the possibility of using different models in a same circuit. This is due to the optimization limitations imposed by the gradient methods. A pure search based optimization algorithm would be a possible solution for this with the further computational costs. Moreover, One could define a general adaptable model to be reduced to the variety of sub-models (e.g. setting all dynamic conductances of a Hodgkin Huxley model to zeros leads to a simple Leaky Integrate), this is however not very elegant nor efficient. This challenge will be the focus of our continued effort.

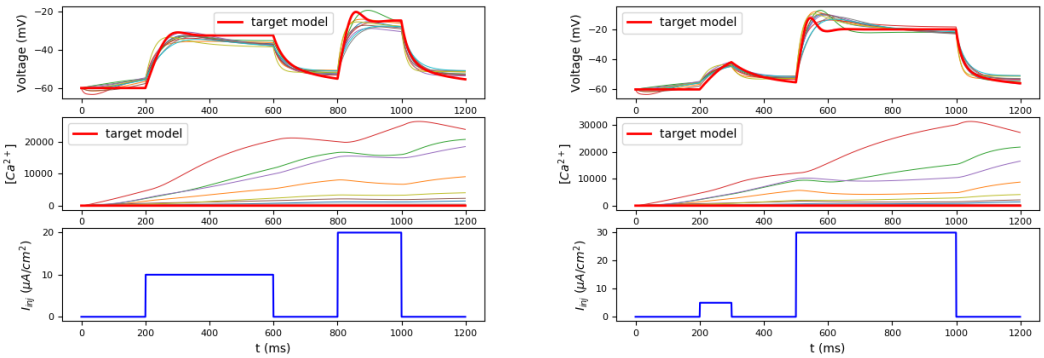
3.5.3 How to perform optimization with ODYNN

In this section, We illustrate a simple example on how to perform an optimization run using *ODYNN*. The code below performs the optimization of a single neuron. It generates a voltage traces from a simulation and optimize 10 neural models to fit these traces. This code plots the simulation features displayed on figure 3.13 and save them in the folder defined by the *folder* variable. Among others, the evolution of the loss, parameters, and curve comparisons will be saved and displayed.
This will allow analysis and reproduction of the results after the end of the optimization.

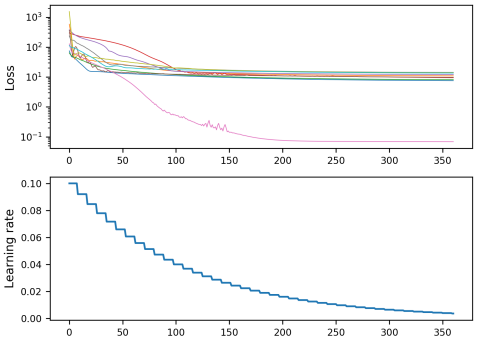
```
import numpy as np
from odynn import utils, nsimul, neuron, noptim
#This file defines the model we will use
from odynn.models import cfg_model

dt = 1.
folder = 'Example'

# Function to call to set the target directories for plots and saved files
dir = utils.set_dir(folder)

#Definition of time and 2 input currents
t = np.arange(0., 1200., dt)
i_inj1 = 10. * ((t>200) & (t<600)) + 20. * ((t>800) & (t<1000))
i_inj2 = 5. * ((t>200) & (t<300)) + 30. * ((t>500) & (t<1000))
i_injs = np.stack([i_inj1, i_inj2], axis=-1)

#10 random initial parameters
params = [cfg_model.NEURON_MODEL.get_random() for _ in range(10)]
neuron = neuron.BioNeuronTf(params, dt=dt)

#This function will take the default parameters of the used model if none is
                                        given
train = nsimul.simul(t=t, i_inj=i_injs, show=True)

#Optimization
optimizer = noptim.NeuronOpt(neuron)
optimizer.optimize(dir=dir, train=train)
```



(a) Plots generated by the simulation. From the top are displayed the voltage (black), the calcium concentration (red), the different ion flows, the activations rates and the input current in blue.



(b) Plots generated for each step during the optimization, comparing the trained models with the target data in red. From the top : voltage, calcium concentration and input current.



(c) Loss evolution of all trained models (top) and learning rate (down)

Figure 3.13: Various plots generated by a python script using ODYNN.

CHAPTER 4

Results

In this section we present the results of our detailed experimentations realized by ODYNN. Unless stated otherwise, optimizations were done with 700 iterations. The other settings, such as the random initialization and constraint bounds, learning rate, time steps, number of models and duration of the optimization will be indicated in tables in annex A. We defined large ranges for constraints and initial parameter values to observe our approach performance when no knowledge is available on the internal neural dynamics.

In every experiment, we used the same initial states for all neurons, described in table 4.1:

Table 4.1: Initial parameter values

V	a	b	e	f	h	p	q	n	$[Ca^{2+}]$
-60.	0	1	0	1	0	0	1	0	1e-7

For commodity, we will refer to our developed models detailed in 3.2.2 as LI for Leaky Integrate (3.2.2), SHH for the simple arbitrary Hodgkin Huxley model (3.2.2) and CHH for the model corresponding to C. Elegans' neurons (3.2.2).

4.1 Fitting single neurons on calcium imaging data

In this experiment, we fit a biophysical model as well as an LSTM network with calcium imaging data from the brain of *C. elegans*.

Calcium Imaging Dataset

We used calcium imaging recordings from [29]. The data contains several minutes of recording from *C. elegans*' interneuron AVA. This recording represents the the variation

of a fluorescent indicator sensitive to the inner calcium concentration. The values are derived from a Hill equation :

$$\frac{\Delta F}{F} \propto \frac{[Ca^{2+}]^n}{[Ca^{2+}]^n + k_d} \tag{4.1}$$

where $\frac{\Delta F}{F}$ is the increase in fluorescence, which is the data we have, and $[Ca^{2+}]$ is the inner calcium concentration. The fluorescent indicator in the measured dataset at hand was GCaMP5K. Its Hill coefficient n is 3.8, and the half-activation k_d is 0.189 mM^n [7]. In order to perform our experiment, we used directly the indicator values, representative for the intracellular calcium concentration. Indeed, as illustrated in figure 4.1, the shape is very similar with different values of α. The main change is the amplitude, and our simple calcium model (Eq. (3.13)) allows any scaling by multiplication of ρ_{Ca} and dividing τ_{Ca} with the same number.

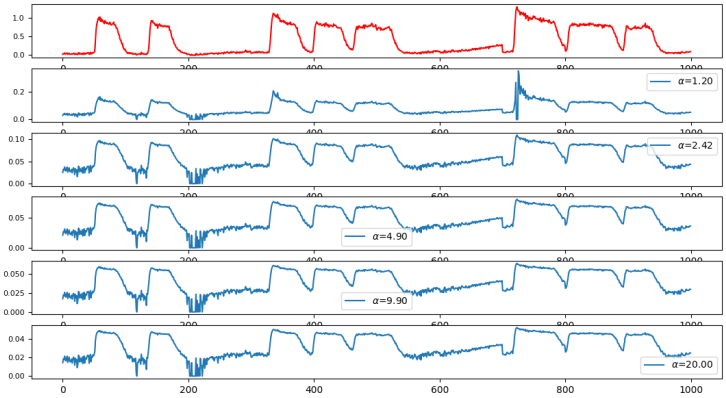


Figure 4.1: The fluorescent indicator (red) compared to the calcium concentration when using different proportion factors (blue)

In order to fit this curve, an input current needs to be defined. As explained by Fuchs et al. [17], in calcium imaging experiments [29][39] and electrophsiological measurements [50], the AVA neurons activity depicts a switching behavior between two states. When applied a ramp shaped input, the neuron produces a graded response [40]. This suggests that the neurons receives a box shape input. Hence, we defined an 'on/off' input current based on the calcium imaging data, which was later convoluted with a Gaussian curve, as in [17]. Since the value of the 'on' current is still unknown, we conducted 5 experiments with values in the range [0.5, 10] $\mu A/cm^2$ which all fitted the curve in a similar way, we depict here one of them.

4.1.1 Fitting calcium imaging data with a C. Elegans Hodgkin Huxley model

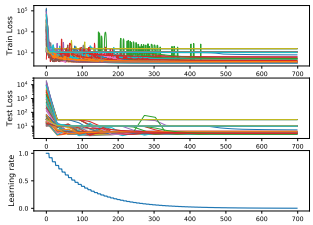
We trained 100 different CHH models in parallel. As explained in the methods, each variable was drawn from a uniform distribution. The boundaries are indicated in table A.1 (Min init and Max init). The time step Δ_t=340 ms corresponds to the time between two measurements. Interestingly, the optimization could handle such a large value. We used a decreasing learning rate as follows:

$$l = 0.92^{\lfloor n/9 \rfloor} \tag{4.2}$$

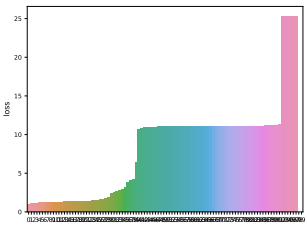
where n is the iteration index. A decreasing learning rate allows fast evolution at the beginning of the optimization, when the behavior is most often completely distinct from the fitted curves. With iterations, the trained model's traces evolves towards its target, and a smaller learning rate creates smaller change in the parameters, hence enabling a more detailed fining.

We selected around 30 minutes of recordings and used one half for training our models and the second half for testing.
Hard constraints were applied to each variable, which bounds are detailed in table A.1 (LB and UB).

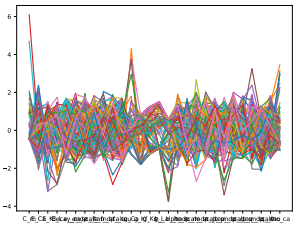
The optimization lasted 32 minutes, although as can be observed in figure 4.2a, most of the models already converged after 300 iterations. As shown in figure 4.2b, they are divided in three categories depending on their final loss. Only the ones before the plateau at around 21 could fit our target. The best model had a cost value of 1.03.



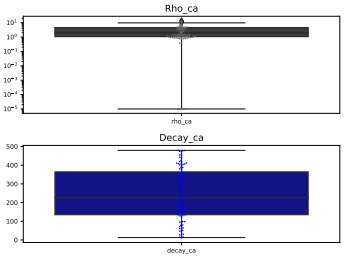
(a) Evolution of the train (top) and test loss (middle), and learning rate (down) with iterations



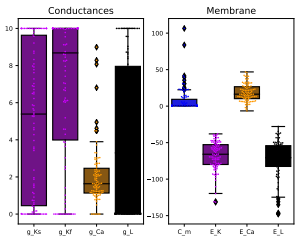
(b) Final loss values for the 100 models



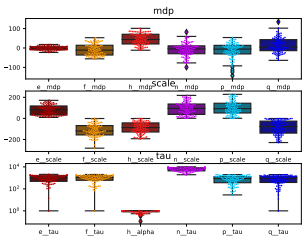
(c) Comparison of the normalized parameters of the best models.



(d) Calcium pump parameters



(e) Membrane parameters



(f) Rates parameters

Figure 4.2: Loss and parameter distribution for fitting calcium imaging data with C. Elegans Hodgkin Huxley.

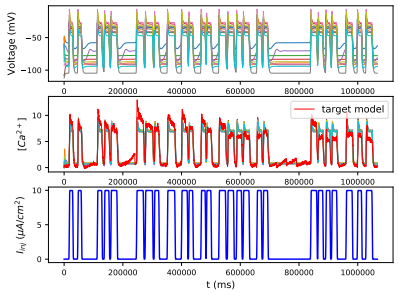
We selected the 41 models of the first category (before the plateau) and compared their parameters in figure 4.2c. All parameters have been normalized by subtracting the mean and dividing by the standard deviation :

$$x_{norm} = \frac{x - \bar{x}}{\sigma_x} \tag{4.3}$$

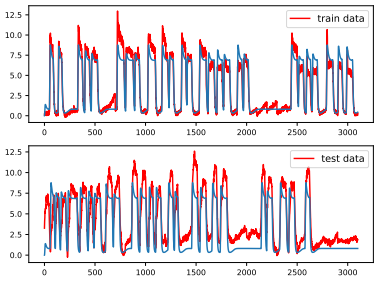
Note that no global structure can be recognized among the different solutions. We display the distribution of the different parameters values in figures 4.2d, 4.2e and 4.2f. Most variables display a great variation, in particular the channel conductances. The calcium channels depicts a lower deviation, since we are fitting the calcium concentration. The 41 best models traces are compared with the calcium imaging data in figure 4.3a and we observe here the consequences of this variation, as they exhibit very different scales related to their membrane potentials. Hence, fitting calcium imaging data is not enough to extract the physiological parameters of a neuron.

In figure 4.3b we show one of the best results compared to the target curve. It was able to capture parts of the phenomenon creating a bigger response when the neuron hasn't been simulated in a long time. However, we were unable to catch a proper long term

potentiation (elevation of the concentration without stimulation). This is presumably due to the simplicity of our model when modeling calcium ions dynamics.



(a) Comparison of the 41 best models with the target (middle, calcium concentration), Voltage (top) and input currents (down) are displayed



(b) Comparison of one of the best models (red) with the target measurement (blue)

Figure 4.3: Comparison of the results from the optimized biophysical model and the fitted calcium imaging data

We performed the same experiment but changed only the initial maximal bound for the time constants (τ's), dividing them by 10. The models were then unable to exhibit a greater response after a long resting time. This demonstrates the sensitivity of gradient descent to initial parameters.

4.1.2 Fitting calcium imaging data with LSTM networks

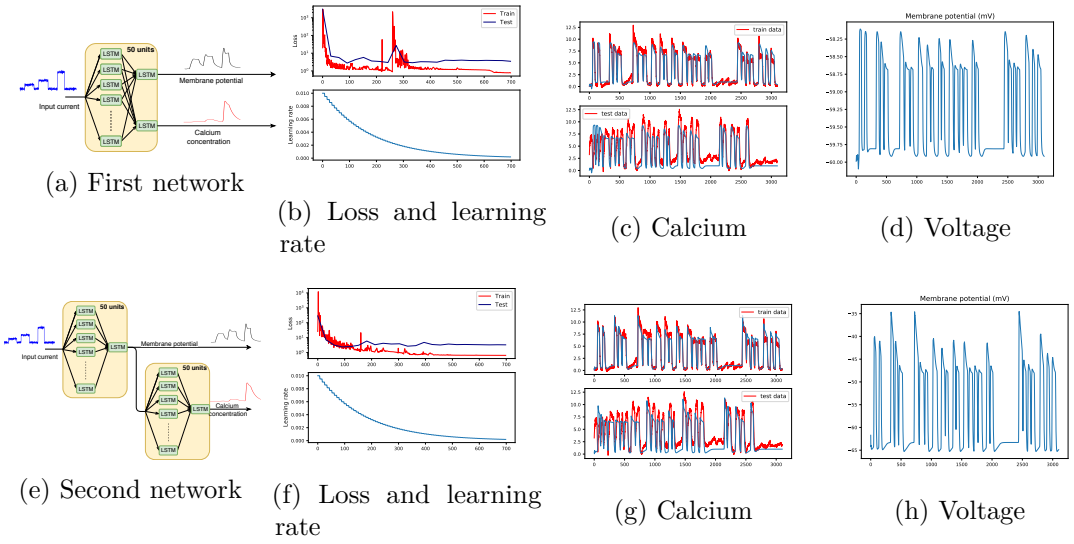
We used here the same data but utilized LSTM networks for fitting. We compared the performances of two different architectures. The first one (figure 4.4a) is made of a single layer of 50 LSTM cells, while the second one (figure 4.4e) is the one we present in 3.2.2, possessing two layers of 50 units.

We used a smaller learning rate of $0.01 \cdot 0.95^{\lfloor n/9 \rfloor}$ than for the biophysical models. In fact, unlike biophysical models, all their parameters are bound between 0 and 1. Furthermore, greater learning rates resulted in chaotic behaviors, where they produced high frequency oscillation and couldn't evolve anymore from this state.

We trained a single model at a time, since the tensorflow implementations didn't allow parallelization. Single cells or layers can be defined and support batch training, however it is currently impossible to add dimensions to the parameters matrices, which would allow parallel training.

Both architecture fitted well both the train and test data (figures 4.4b, 4.4f, 4.4c and 4.4g), and obtained a final MSE of respectively 0.79 and 0.64. Hence, they performed better than the biophysical models. However, their membrane potentials had shapes

very similar to the calcium concentration curve. This observation is obvious for the first architecture, while it is more interesting in the second case. The main difference between them was the range of the potential values, the second architecture depicts a more realistic one with voltage in [-65, -38] mV against [-60, -58] mV for the first one.



(a) First network (b) Loss and learning rate (c) Calcium (d) Voltage

(e) Second network (f) Loss and learning rate (g) Calcium (h) Voltage

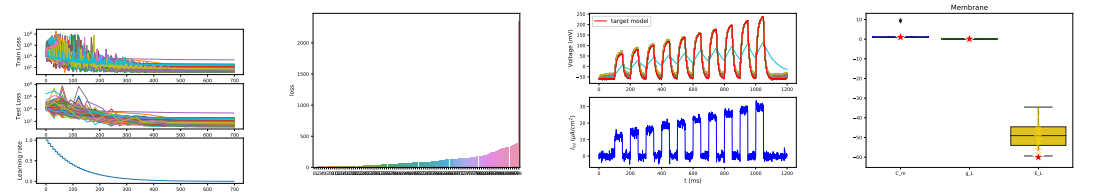
Figure 4.4: Comparison of the two LSTM network architectures (two rows) for fitting calcium imaging data.

4.2 Fitting single neurons on simulated data

In another experiment, we explored the possibilities of our algorithm to find parameter sets by fitting outputs generated by these same sets.
We first hand tuned biophysical parameters for each model until their simulations exhibited realistic or interesting behaviors. We then defined 10 different input currents lasting for 1.2 s each, and separated them into train (5) and test (5) data. They can be visualized in figure 4.6 as the blue curves. We ran a simulation using those 10 scenarios on our target parameters, recorded the corresponding membrane potentials (and calcium concentrations for the *C. elegans* model) and used them as target for our optimization. Before the optimization, we added random noise on the records and input currents in order to evaluate the robustness of our approach. For the latter, we used a Gaussian distribution with standard deviation 1.5.
We performed this for each of the 3 biophysical models presented in chapter 3, each time training 100 randomly initialized parameter sets.
The parameter used for the initial simulation are presented in tables in annex A.2 with column 'Target' and the parameters giving the best fit in column 'Best fit'.

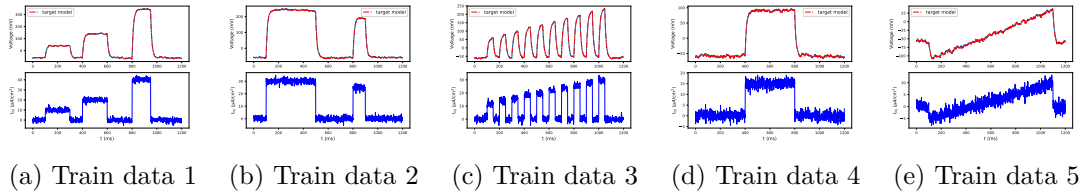
4.2.1 Fitting simulated data with Leaky Integrate

Our first optimization experiment was using the LI model. All solutions except one exhibited a precise fitting (figure 4.5) and were able to find the initial parameters reasonably accurate, at the exception of the reversal potential E_L which shows a greater variation. Indeed, the membrane capacitance C_m and the leakage conductance g_L control the reactivity and amount of current, and thus the speed at which the membrane potential varies, which is robust to noise. E_L only controls the resting state and can be blurred with noise.
The best model had a final loss of 12.46 mV^2 and its parameters written in table A.4 are very close to the targets.
In figure 4.6, we represent a comparison of this best fit with the behavior of the target model on the training and testing currents. Overall, the algorithm could fit the desired curve and recover the initial parameters.

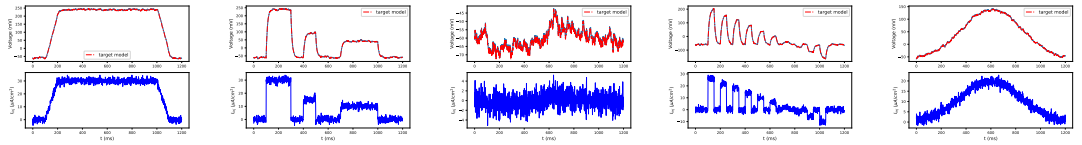


(a) Evolution of the loss (b) Final cost function values (c) 100 models with the target (d) Distribution of the 100 models parameters

Figure 4.5: Results for fitting simulated data with Leaky Integrate. The red stars over the box plots indicate the value of the parameters producing the target data



(a) Train data 1 (b) Train data 2 (c) Train data 3 (d) Train data 4 (e) Train data 5

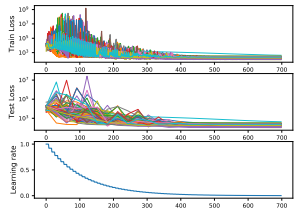


(f) Test data 1 (g) Test data 2 (h) Test data 3 (i) Test data 4 (j) Test data 5

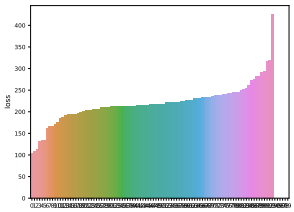
Figure 4.6: Comparison of the membrane potential of the best (blue) and target (red) model with the different input currents (dark blue) when fitting simulated data with Leaky Integrate

4.2.2 Fitting simulated data with a simple Hodgkin Huxley model

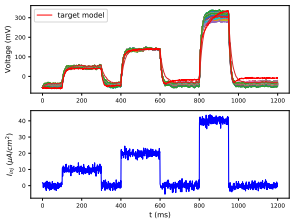
Our second experimentation studied the SHH model. Again, all models demonstrated reasonably good fit to the target waveform (figure 4.7) and were able to find the proper C_m and channel conductances values, while the reversal potentials were more varied (figure 4.7d). Mainly, they all show the exact same shape, however slightly shifted in the direction of the evolution, hence being more sensitive to inputs. The best parameter set displayed in table A.5 is now far from the target model, although their behaviors are almost identical, as highlighted in figure 4.8. Its final cost value was 104 mV^2.
The different rate parameters values (figure 4.7e) are spread. This demonstrates that the Simple Hodgkin Huxley model is already complex enough to exhibit the same behavior in different regions of the parameter space, and a good fit does not imply similar parameters.



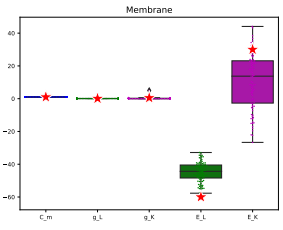
(a) Evolution of the loss



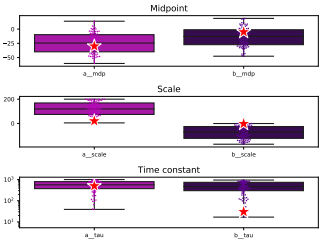
(b) Final cost function values



(c) 100 models with the target

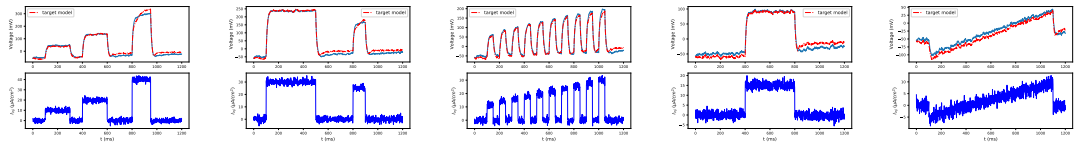


(d) Distribution of the membrane parameters

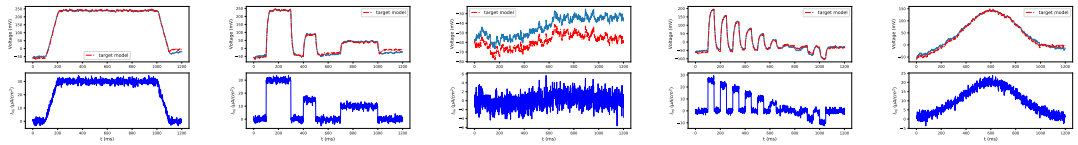


(e) Distribution of the rate parameters

Figure 4.7: Results for fitting simulated data with simple Hodgkin Huxley. The red stars over the box plots indicate the value of the parameters producing the target data



(a) Train data 1 (b) Train data 2 (c) Train data 3 (d) Train data 4 (e) Train data 5

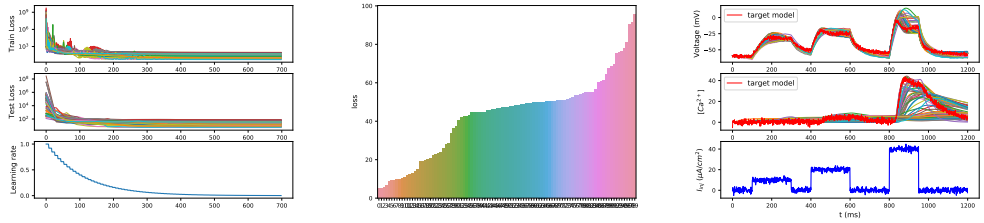


(f) Test data 1 (g) Test data 2 (h) Test data 3 (i) Test data 4 (j) Test data 5

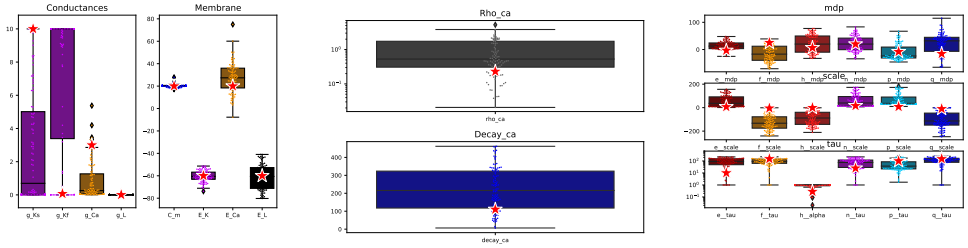
Figure 4.8: Comparison of the membrane potential of the best (blue) and target (red) model with the different input currents (dark blue) when fitting simulated data with simple Hodgkin Huxley

4.2.3 Fitting simulated data with *C. elegans* Hodgkin Huxley

The third experiment was conducted using the *C. elegans* Hodgkin Huxley model (CHH), we fitted the voltage and calcium concentration, simultaneously . This time, all models exhibited a reasonably accurate fit for the voltage, but a greater difficulty for the calcium concentration (figure 4.9c). In fact, in this precise case, the values range for the potential were higher and thus had a greater weight in the cost function. Once a model found a good location in the parameter space to mimic the voltage, it could face an impossibility to match the calcium dynamic without losing its fit on the potential. As a matter of fact, gradient descent only allows temporary greater values for the loss.
All models were able to find the values for C_m and g_L 4.9d). The other conductances were most of the time completely distinct from the target ones, especially for the two potassium channel which show inversion of conductances values. Because of the equal constraints applied on the rate parameters and conductances, they were fully free and found various settings of compensating each other to deliver similar behaviors, this is particularly true for the two potassium channels using the same reversal potential E_K.
The best parameters written in table A.6 are now far from the target model, despite their very close behaviors highlighted in figure 4.10 and final loss of 5.17.
Hence, a greater complexity in the model leads to numerous possibilities leading to similar reactions.

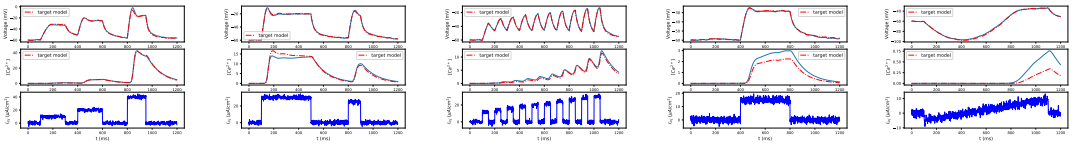


(a) Evolution of the loss (b) Final cost function values (c) 100 models with the target

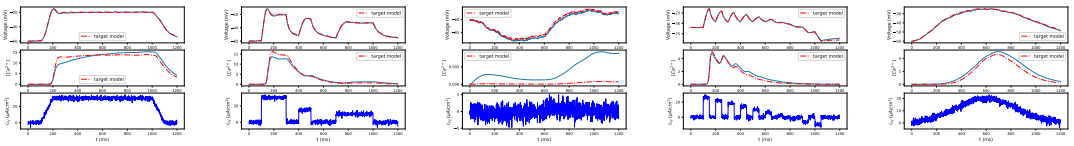


(d) Distribution of the membrane parameters (e) Calcium pump parameters (f) Distribution of the rate parameters

Figure 4.9: Results for fitting simulated data with C. Elegans Hodgkin Huxley. The red stars over the box plots indicate the value of the parameters producing the target data



(a) Train data 1 (b) Train data 2 (c) Train data 3 (d) Train data 4 (e) Train data 5



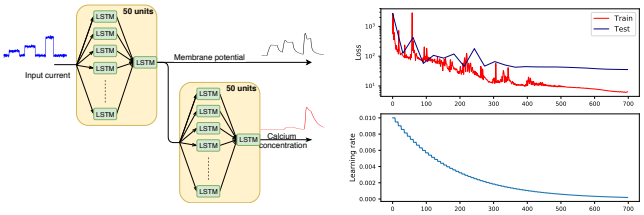
(f) Test data 1 (g) Test data 2 (h) Test data 3 (i) Test data 4 (j) Test data 5

Figure 4.10: Comparison of the membrane potential (top) and calcium concentration (middle) of the best (blue) and target (red) model with the different input currents (dark blue) when fitting simulated data with C. Elegans Hodgkin Huxley

4.2.4 Fitting simulated data from *C. elegans* neural model with LSTM

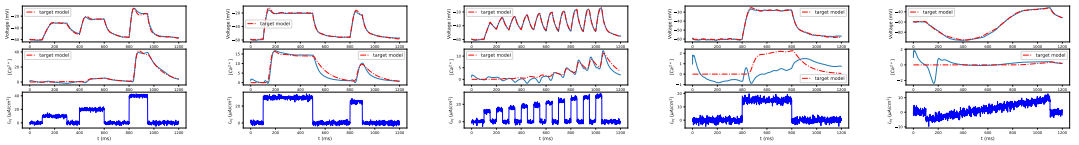
In another experiment, we tested the ability of the LSTM networks to mimic the same simulated data from CHH. We used the two layers network described in the methods

and trained it on the data from the previous experiment (4.2.3).
As we can see in figures 4.11b and 4.12, it nicely fitted both the train and test data, in particular the voltage traces and obtained a loss of 6.14, similar to the biophysical model's score on the same data (5.17). The calcium concentration was not as precise because the noise had more impact on it, since its range values were smaller but were applied the same noise.
However, we can say that the LSTM network showed robustness to noise and could surprisingly capture the behavior of our target model.

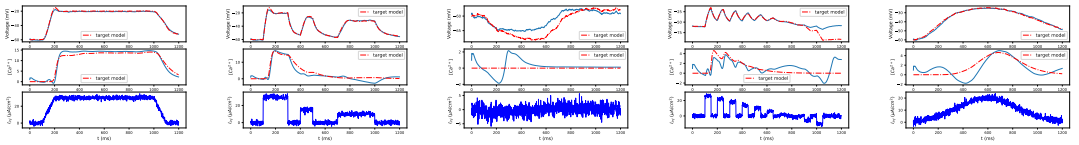


(a) Network architecture

(b) Loss evolution



(a) Train data 1 (b) Train data 2 (c) Train data 3 (d) Train data 5 (e) Train data 5



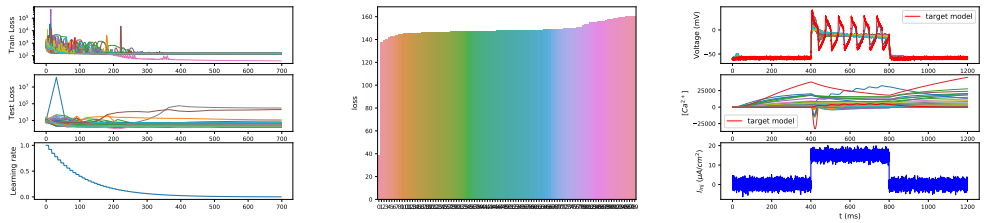
(f) Test data 1 (g) Test data 2 (h) Test data 3 (i) Test data 4 (j) Test data 5

Figure 4.12: Comparison of the membrane potential of the best (blue) and target (red) model with the different input currents (dark blue) when fitting simulated data from C. Elegans Hodgkin Huxley with LSTM networks

4.2.5 Fitting oscillatory behavior with *C. elegans* neural model

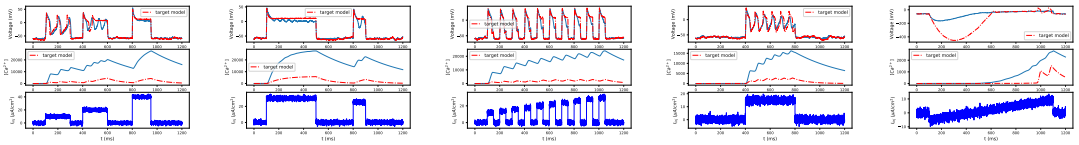
In the past experiments, our methods demonstrated strength in fitting noised simulated data. However, one of our parameter sets for the CHH model exhibited an oscillatory activity on some input current ranges only. We encountered high difficulty to match this behavior. We had to restrict our cost function for voltage only and use a single train data to succeed. Still, as we can see in figures 4.13a and 4.13b, out of the 100 models, only one could oscillate properly. All the other models simply averaged the potential

during the oscillations (figure 4.13c.

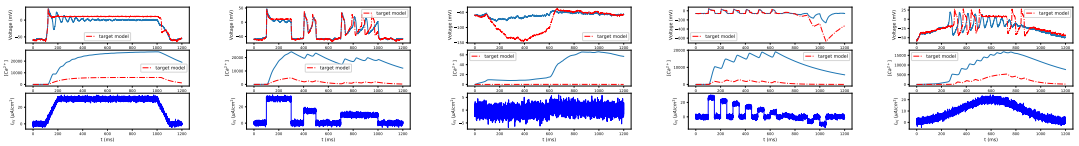


(a) Evolution of the loss (b) Final cost function values (c) 100 models with the target

Figure 4.13: Loss and curves comparison for the oscillatory behavior



(a) Train data 1 (b) Train data 2 (c) Train data 3 (d) Train data 4 (e) Train data 5



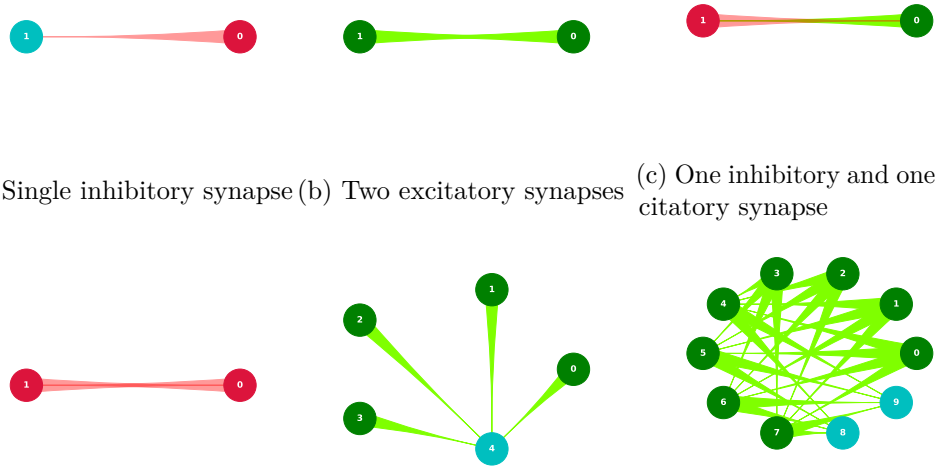
(f) Test data 1 (g) Test data 2 (h) Test data 3 (i) Test data 4 (j) Test data 5

Figure 4.14: Comparison of the membrane potential of the best (blue) and target (red) model with the different input currents (dark blue) when fitting oscillating simulated data with C. Elegans Hodgkin Huxley

4.3 Optimizing simple circuits with simulated data

We were able to successfully optimize the circuits shown in figure 4.15, i.e. both the neural and synaptic parameters. We go through the details of the optimization pipeline on a more challenging architecture in the next chapter.

Our procedure was similar to the previous section, we used the output of a simulation as target for several models trained in parallel. The cost functions were calculated based on the potential of a subset of neurons, respectively (for each architecture in figure 4.15), neuron 1 for the first 4 architectures, neuron 4 for the fifth and and neurons 8 and 9 for the last one.



(a) Single inhibitory synapse (b) Two excitatory synapses (c) One inhibitory and one excitatory synapse

(d) Two inhibitory synapses (e) 4 neurons with excitatory synapses towards a fifth one (f) Fully connected circuit (of excitatory synapses) with 3 layers : 4, 4 and 2 neurons

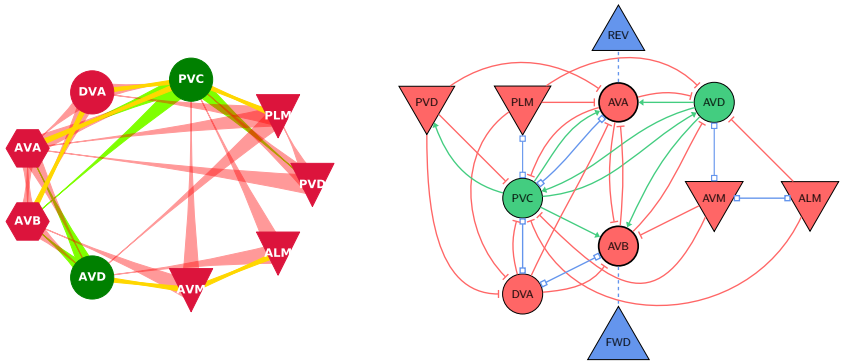
Figure 4.15: Simple architectures ODYNN could optimize. Each disk is a neuron. Synapses are represented by triangles which bases are on the pre-synaptic neurons and last angle on the post-synaptic one. Green is for excitatory and red for inhibitory.

In several experiments, we evaluated the possibilities of replacing some neurons by LSTM networks for optimization. However, none of them were successful. We hold the opinion that this happened because of both a vanishing gradient and too different form of gradients between the biophysical and artificial parameters.

4.4 Optimizing the Tap Withdrawal circuit

The tap-withdrawal circuit is the part of *C. elegans*' nervous system responsible for the reflexive response of the animal to external mechanical stimulus. It has been well studied, for instance for studying habituation [48]. More recently, it has been used for deep reinforcement learning for robot control [34], where neurons were modeled with Leaky Integrate.
We performed an experiment on this circuit to test the ability of our algorithm for scaling. The network is illustrated in figure 4.16. It contains 9 neurons, 23 synapses and 6 gap junctions, resulting in 350 free parameters with the CHH model.

As previously, we first selected arbitrary parameters and input currents applied to the sensory neurons. We then ran our simulation and saved the membrane potentials of the two command neurons AVA and AVB. Finally, we draw all parameters from uniform

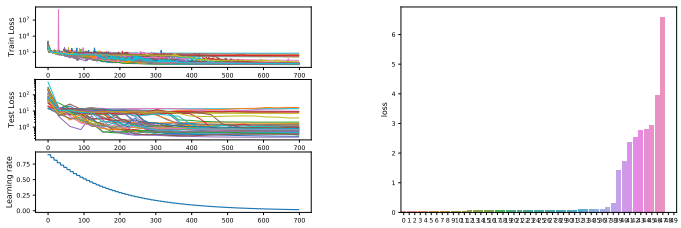


(a) Automatized circuit representation from ODYNN

(b) Circuit representation taken from Wicks et al. [63]

Figure 4.16: Two alternative representations of the tap withdrawal circuit
Triangles are for sensory neurons, circles are inter neurons and hexagons (left only) are for command neurons. Red and green stands for respectively inhibitory and excitatory synapses. Yellow (left) or blue (right) lines model gap junctions.

distributions as usual and optimized them in order to fit the membrane potentials of AVA and AVB. The parameters were constrained and initialized as in the previous experiments for neurons, and as shown in table A.8 for synapses. We applied the same to all neurons and synapses, except for parameter E_{syn} which depended on whether a synapse was excitatory or inhibitory. We trained 50 models in parallel with a time step of 0.5 ms. The dataset consisted of 3 train and test datas of 1.2 seconds each.



(a) Evolution of the loss

(b) Final cost function values

Figure 4.17: Loss evolution and final values when optimizing the Tap Withdrawal circuit.

Total training time was 2 hours and a half. The evolution of the loss and the final values are displayed in figure 4.17. As we can see, most models fit both to the train and test data with a high precision. Out of 50, 38 have a final loss of less than 0.5 mV^2.
We present on figure 4.18 a comparison of the best result and the initial simulation on each neuron of the circuit. As we can see, although the fitted neurons AVA and AVB match their target, the other neurons show a different behavior than in the initial simulation. This shows freedom in the parameter space for producing a given behavior,

in the sense that several solutions exist resulting in the wanted target waveforms for AVA and AVB.

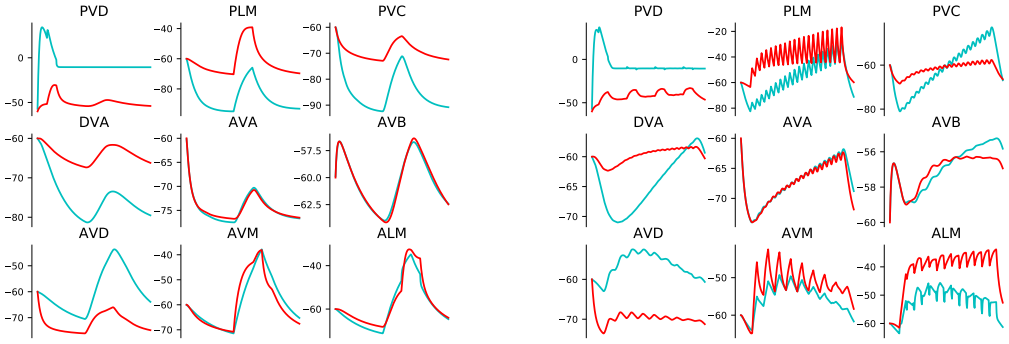


Figure 4.18: Comparison of the initial simulation (red) and best result (cyan) on each neuron with two different test input currents. The values are in mV.

We performed 5 experiments with time steps in the range [0.1, 1] ms, various parameter values producing the simulated data, and different random seeds on this same circuit. All of them had similar outcomes.

Hence for optimizing neuronal circuits deployed by a *C. elegans* Hodgkin Huxley model, the ODYNN optimization suite is performant. However, this optimization only took into account a subset of neurons and fitted arbitrary data, we perform in the next section an optimization of all neurons of a larger circuit.

As previously, the optimization couldn't handle the replacement of any subset of biophysical models by LSTM networks.

4.5 Optimizing the Forward Locomotion circuit

The Forward Locomotion Circuit (FLC) is a large subpart of *C. elegans*' nervous system, originating the worm's forward crawling [36]. The circuit contains 114 chemical synapses, 82 electrical synapses and 39 neurons, summing up to 1630 free parameters in a *C. elegans* Hodgkin Huxley setting, and thus forms an interesting optimization challenge. In [36], authors managed to obtain a crawling behavior of the worm by performing a hand-tuning and genetic algorithms. The crawling behavior requires successive activation and inactivation of neurons in a specific order. To reach this objective, they used a single set of parameters shared among all neural cells and used a different chemical synaptic model [47] than the one we designed, as they used a single time-constant synapse model, resulting in a more progressive evolution of the synaptic conductance.

We performed 8 different experiments performing 1400 iterations with a learning rate $l_r = 0.4 \cdot 0.95^{\lfloor \frac{n}{9} \rfloor}$. We defined a smaller initial value than previously since the large number

of parameters induces a higher sensitivity to changes. Indeed, because of the multiple interconnections between neurons, each parameter change results in consequences all over the circuit. The decreasing is also smoother for allowing a longer tuning, again because of the high dimensionality. The constraints are displayed in table A.9. The only change for this experiment is the lower upper bounds for the synaptic conductances (from 2 to 0.5), because higher upper bounds were leading to exploding behaviors. We opted for a time step of 0.2 ms for the same reasons, and optimized each time 10 models in parallel.

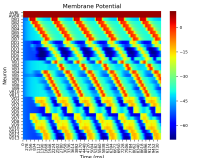
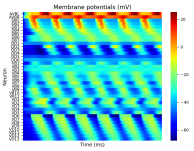
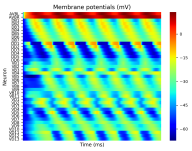
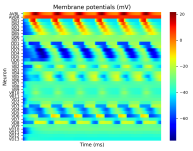
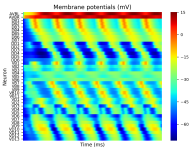
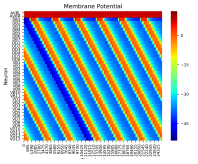
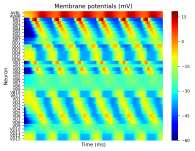
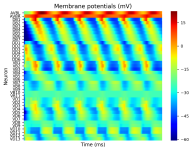
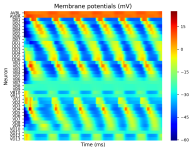
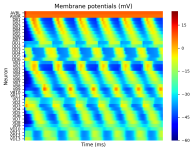
Regarding the target data, we first collected the membrane potentials exhibited by the 39 neurons with the circuit parameters obtained by Lung et al. [36]. The data is displayed in the first row of table 4.2. Moreover, we defined a more regular target by selecting the membrane potential record of the first oscillating neuron DB1, and defining for each neuron the target of its predecessor shifted by 64 ms. The curves were then applied a gaussian noise of standard deviation 1 mV. We obtained in this way a more regular shape, but with the same time shifting between consecutive neurons as in the original data. This second target is shown in the second row of table 4.2. The reason for using this redefined objective is that when optimizing a complex circuit towards an expected behavior, one can not anticipate all precise variations of every neuron, and would most likely define such a simple target at first.

For each target data, we compared the outcomes of the LI and CHH model. Furthermore, in each setting, we conducted one full optimization, i.e. optimizing each parameters separately, and a 'grouped' one, where we used a single set of biophysical parameters shared among all neurons. The latter allows to considerably reduce the dimensionality, and is moreover the way OpenWorm [59], used by Lung in its experiment, perform simulation. Namely, they ignore the particularity of each cell by using a single set of parameters for every neuron.

Interestingly, grouping neurons did not have any impact on the duration of experiments. This is because the computed gradients do not change, only parameters were being updated by a combination of them. The optimization times are displayed in table A.4, and were approximately 16 hours when adopting Leaky Integrate, and 29 hours for the more complex model.

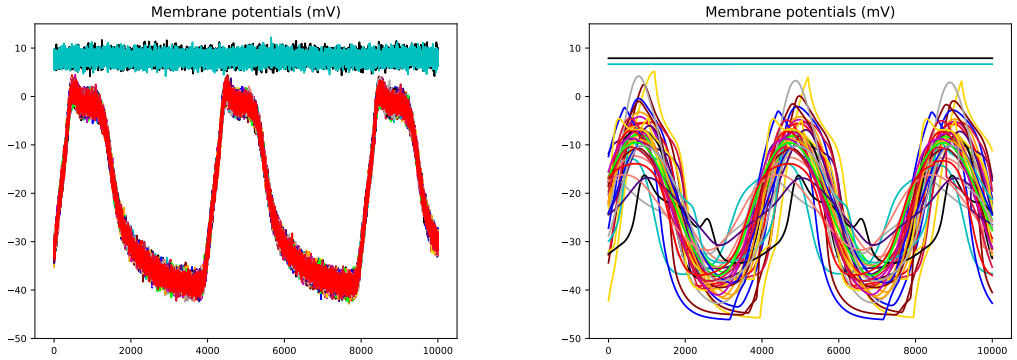
The best results of each experiment are compared graphically in table 4.2 together with their final losses. In all settings, the models were able to reproduce the traveling waves for the majority of neurons. The waves are more clearly defined when using the redefined target, presumably because the output from [36] contains additional unexpected behaviors induced by the successive hand tuning and optimizations applied. Reducing the parameter space by using a single set of parameters showed better results for the first target. Surprisingly however, this led to higher losses with the redefined target. As a matter of fact, each neuron possessing different connections, it is presumably harder to obtain a identical behavior for all of them with a single model. Overall, despite the higher complexity, the CHH exhibited a higher capability of fitting over LI. Hence, although it had 9 times more trainable variables per neuron, the gradient descent optimization was able to take advantage of the more powerful expressiveness of this model. In general,

Table 4.2: Comparison of the best results for each of the 8 experimental optimization of the FLC. Each figure displays the evolution of the 39 neurons' membrane potentials with time. Each row corresponds to a neuron. Warmer colors encode higher values.

Target curves	Leaky Integrate		C. Eleg. Hodgkin Huxley	
	Full optim.	Grouped	Full optim.	Grouped
from [36]	loss=103	loss : 93.3	loss : 117	loss : 79.4
				
redefined	loss : 119	loss : 124	loss : 90.4	loss : 99.2
				

we obtained the best outcome using a CHH model and grouping neurons. Although the distance with the objective was not the lowest for the redefined data, this setting provided the best defined traveling waves.

We compare this last outcome in a different manner in figure 4.19, where we 'unshifted' the membrane potentials of the neurons. As we can see, the result displays the expected period. However, some neurons exhibit different shapes and amplitudes, because of their different connections.



(a) Target curves

(b) Result

Figure 4.19: Comparison of the target data with the result when 'unshifting' the traveling waves. Each line represent the membrane potential of a neuron.

CHAPTER 5

Conclusion

5.1 Summary

In this study, we introduced ODYNN, a novel optimization scheme for biological neural circuits that takes advantage of both search and gradient based optimization methods. It allowed us to perform numerous experiments with variety of neuronal models and circuit architectures.

ODYNN was able to handle parallel optimization of large and complex neural circuits, with large parameter spaces, i.e. up to 1630 parameters, and provide promising results. Taking advantage of the computational graph of Tensorflow, it demonstrated the capability of performing such tasks successfully and in a reasonable amount of time. The hybrid solver strategy for solving the biophysical models' dynamics exhibited stability with large time steps, up to 340 ms for a single neuron.

Adam optimization performed well in fitting measurements for single neurons and different circuit architectures. We showed that our approach is resistant to noise and that the number of solutions for fitting membrane potentials recordings increases proportionally to the complexity of the used model. Hence, it is necessary to perform more precise physical measurements in order to catch the biophysical properties of a neuron. In the same manner, circuits possess a high degree of freedom for producing a target behavior for a subset of nodes. However, we demonstrated the possibility to fully optimize the Forward Locomotion circuit of C. Elegans, consisting in 39 neurons, 114 chemical synapse and 82 gap junctions.

Additionally, we analyzed the ability of LSTM networks to capture neuronal dynamics. Those networks were able to mimic calcium imaging data, as well as measurements from biophysical model simulations. They demonstrated capability of generalization by adopting the appropriate behavior on unseen data.

5.2 Open challenges

The purpose of this section is to explain the main limitations we encountered and introduce open challenges and some directions to overcome them.

5.2.1 Finding biophysical parameters

As we explained, despite succeeding in fitting traces, stochastic gradient descent was unable to find the biophysical parameters used to produce the target curves of the optimization when dealing with Hodgkin Huxley models. In order to solve this problem, evolutionary algorithms could be used, since they usually perform better in finding global optima. However, the parameter space may simply contain several global optima. In the latter case, we should study if some particular applied currents can differentiate them, as well as a global way to distinguish similar sets of parameters. Another possibility would be to perform more precise measurements on neural membrane to be able to fix some parameters.

5.2.2 Fitting oscillatory behavior

The hybrid optimization we used showed an increased difficulty for mimicking oscillatory behaviors. The reasons for this difficulty should be further investigated. Integrating frequency in the cost function could help for this matter. The cost function should, however, stay differentiable. Again, evolutionary algorithms might perform better in this particular case.

5.3 Future work

For future work, we intend to test the presented algorithms on patch clamp data (voltage and current recordings) for single neurons. We should also gather recordings on a circuit scale to evaluate the capabilities of circuit tuning on real data and larger circuits.

Regarding ODYNN, we aim to develop a more complex structure for circuits in order to incorporate different synaptic models. We are also considering to integrate other optimization techniques in order to compare in a more detailed way their speed and efficiency.

Implementing our own LSTM cell would allow us to parallelize the training of LSTM networks, as we did for biophysical models, and to study their limiting behavior in circuit tuning.

APPENDIX A

Experimental settings

A.1 Fitting single neurons on calcium imaging data

A.1.1 Fitting calcium imaging data with a Hodgkin Huxley model

Table A.1: Experimental settings for fitting calcium imaging data with a Hodgkin Huxley model

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
100	$0.92^{\lfloor n/9 \rfloor}$	340ms	1	1	33min	0.53 h

Parameter	Min init	Max init	LB	UB	Unit
C_m	0.5	40	0.5	∞	μF/cm^2
E_{Ca}	0	40	-∞	∞	mV
E_K	-80	-40	-∞	∞	mV
E_L	-80	-40	-∞	∞	mV
τ_{Ca}	10	500	0.1	∞	none
V^e_{mdp}	-30	0	-∞	∞	mV
V^e_{scale}	1.0	200.0	1.0	∞	mV
τ^e	1.0	2000.0	1.0	∞	ms
V^f_{mdp}	-50	50	-∞	∞	mV
V^f_{scale}	1.0	200.0	-∞	-1.0	mV
τ^f	1.0	2000.0	1.0	∞	ms
g_{Ca}	0.1	10	1e-05	10.0	mS/cm^2
g_{Kf}	0.1	10	1e-05	10.0	mS/cm^2
g_{Ks}	0.1	10	1e-05	10.0	mS/cm^2
g_L	0.0001	0.5	1e-05	10.0	mS/cm^2
α^h	0.1	0.9	0	1	none
V^h_{mdp}	1	100	-∞	∞	mV
V^h_{scale}	1.0	200.0	-∞	-1.0	mV
V^n_{mdp}	-50	50	-∞	∞	mV
V^n_{scale}	1.0	200.0	1.0	∞	mV
τ^n	2000.0	10000.0	1.0	∞	ms
V^p_{mdp}	-50	50	-∞	∞	mV
V^p_{scale}	1.0	200.0	1.0	∞	mV
τ^p	1.0	2000.0	1.0	∞	ms
V^q_{mdp}	-50	50	-∞	∞	mV
V^q_{scale}	1.0	200.0	-∞	-1.0	mV
τ^q	1.0	2000.0	1.0	∞	ms
ρ_{Ca}	1e-5	10	1e-05	10	none

A.1.2 Fitting calcium imaging data with LSTM networks

Table A.2: Experimental settings for fitting calcium imaging data with a single layer LSTM network (figure 4.4a)

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
1	$0.01 \cdot 0.95^{\lfloor n/9 \rfloor}$	340ms	1	1	33min	0.3 h

Table A.3: Experimental settings for fitting calcium imaging data with a 2 layers LSTM network (figure 4.4e)

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
1	$0.01 \cdot 0.95^{\lfloor n/9 \rfloor}$	340ms	1	1	33min	25min

A.2 Fitting single neurons on simulated data

A.2.1 Fitting simulated data with Leaky Integrate

Table A.4: Experimental settings for fitting simulated data with Leaky Integrate

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
100	$0.92^{\lfloor n/9 \rfloor}$	1ms	5	5	1.2s	0.5 h

Parameter	Min init	Max init	LB	UB	Target	Best fit	Unit
C_m	0.5	40	0.5	∞	1	1.0201392	μF/cm^2
g_L	1e-5	10	1e-9	10	0.1	0.099840015	mV
E_L	-70	-45	-∞	∞	-60.0	-59.39891	mV

A.2.2 Fitting simulated data with a simple Hodgkin Huxley model

Table A.5: Experimental settings for fitting simulated data with a simple Hodgkin Huxley model

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
100	$0.92^{\lfloor n/9 \rfloor}$	1ms	5	5	1.2s	0.6 h

Parameter	Min init	Max init	LB	UB	Target	Best fit	Unit
C_m	0.5	40.0	0.5	40.0	1.0	1.0462663	μF/cm^2
E_K	-40	30	-∞	∞	30.0	28.679049	mV
E_L	-70	-45	-∞	∞	-60.0	-49.92975	mV
V^a_{mdp}	-50	0	-∞	∞	-30.0	-57.70183	mV
V^a_{scale}	1.0	200.0	1.0	200.0	20.0	50.879784	mV
τ^a	1.0	1000.0	1.0	1000.0	500.0	733.7421	ms
V^b_{mdp}	-30	20	-∞	∞	-5.0	-34.603146	mV
V^b_{scale}	-200.0	-1.0	-200.0	-1.0	-3.0	-48.160458	mV
τ^b	1.0	1000.0	1.0	1000.0	30.0	101.292244	ms
g_K	1e-5	10	1e-09	10.0	0.5	0.5530106	mS/cm^2
g_L	1e-5	10	1e-09	10.0	0.1	0.103595845	mS/cm^2

A.2.3 Fitting simulated data with *C. elegans* Hodgkin Huxley

Table A.6: Experimental settings for fitting simulated data with *C. elegans* Hodgkin Huxley

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
100	$0.92^{\lfloor n/9 \rfloor}$	1ms	5	5	1.2s	1 h

Parameter	Min init	Max init	LB	UB	Target	Best fit	Unit
C_m	0.5	40	0.5	∞	20.0	20.335058	μF/cm^2
E_{Ca}	0	40	-∞	∞	20.0	10.355236	mV
E_K	-80	-40	-∞	∞	-60.0	-60.515057	mV
E_L	-80	-40	-∞	∞	-60.0	-72.45788	mV
τ_{Ca}	10	500	0.1	∞	110.0	120.05607	none
V^e_{mdp}	-30	0	-∞	∞	-3.36	9.678116	mV
V^e_{scale}	1.0	200.0	1.0	∞	6.75	8.948739	mV
τ^e	1.0	200.0	1.0	∞	10.0	5.938851	ms
V^f_{mdp}	-50	50	-∞	∞	25.2	22.144915	mV
V^f_{scale}	1.0	200.0	-∞	-1.0	-5.03	-21.890793	mV
τ^f	1.0	200.0	1.0	∞	151.0	77.04556	ms
g_{Ca}	0.1	10	1e-05	10.0	3.0	2.4085476	mS/cm^2
g_{Kf}	0.1	10	1e-05	10.0	0.07	9.705711	mS/cm^2
g_{Ks}	0.1	10	1e-05	10.0	10.0	0.32655165	mS/cm^2
g_L	0.0001	0.5	1e-05	10.0	0.005	1e-05	mS/cm^2
α^h	0.1	0.9	0	1	0.282	0.8330167	none
V^h_{mdp}	1	100	-∞	∞	6.42	71.03009	mV
V^h_{scale}	1.0	200.0	-∞	-1.0	-1.0	-76.96787	mV
V^n_{mdp}	-50	50	-∞	∞	19.9	14.857346	mV
V^n_{scale}	1.0	200.0	1.0	∞	15.9	35.30616	mV
τ^n	200.0	1000.0	1.0	∞	25.0	217.60043	ms
V^p_{mdp}	-50	50	-∞	∞	-8.05	-28.896307	mV
V^p_{scale}	1.0	200.0	1.0	∞	7.43	31.390028	mV
τ^p	1.0	200.0	1.0	∞	100.0	22.043167	ms
V^q_{mdp}	-50	50	-∞	∞	-15.6	36.714428	mV
V^q_{scale}	1.0	200.0	-∞	-1.0	-9.97	-125.18112	mV
τ^q	1.0	200.0	1.0	∞	150.0	207.72289	ms
ρ_{Ca}	1e-5	10	1e-05	10	0.23	2.0701041	none

A.2.4 Fitting simulated data from *C. elegans* neural model with LSTM

Table A.7: Experimental settings for fitting simulated data from *C. elegans* neural model with LSTM

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
1	$0.01 \cdot 0.95^{\lfloor n/9 \rfloor}$	1ms	5	5	1.2s	0.42 h

A.3 Optimizing The Tap Withdrawal circuit

Table A.8: Experimental settings for the tap withdrawal circuit optimization.

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
50	$0.92^{\lfloor n/9 \rfloor}$	0.5ms	3	3	1.2s	2.5 h

Parameter	Min init	Max init	LB	UB	Unit
G_{gap}	1e-7	2	1e-7	2	mS/cm^2
G_{syn}	1e-7	2	1e-7	2	mS/cm^2
V_{mdp}^{syn}	-50	60	$-\infty$	∞	mV
V_{scale}^{syn}	0.1	100	0.1	∞	mV
$E_{syn}^{inhibitory}$	-120	-60	-120	-60	mV
$E_{syn}^{excitatory}$	-60	50	-60	50	mV

A.4 Optimizing the Forward Locomotion circuit

Table A.9: Constraints and initialization for the FLC optimization.

Parameter	Min init	Max init	LB	UB	Unit
G_{gap}	1e-7	0.5	1e-7	0.5	mS/cm^2
G_{syn}	1e-7	0.5	1e-7	0.5	mS/cm^2
V_{mdp}^{syn}	-50	60	$-\infty$	∞	mV
V_{scale}^{syn}	0.1	100	0.1	∞	mV
$E_{syn}^{inhibitory}$	-120	-60	-120	-60	mV
$E_{syn}^{excitatory}$	-60	50	-60	50	mV

Table A.10: Experimental settings for fully optimizing the FLC with Lung's output, Leaky Integrate

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	16.1 h

Table A.11: Experimental settings for fully optimizing the FLC with redefined output, Leaky Integrate

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	16.3 h

Table A.12: Experimental settings for optimizing the FLC grouping neurons with Lung's output, Leaky Integrate

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	16.4 h

Table A.13: Experimental settings for optimizing the FLC grouping neurons with redefined output, Leaky Integrate

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	16.6 h

Table A.14: Experimental settings for fully optimizing the FLC with Lung's output, C. Elegans Hodgkin Huxley

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	28.8 h

Table A.15: Experimental settings for fully optimizing the FLC with redefined output, C. Elegans Hodgkin Huxley

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	24.7 h

Table A.16: Experimental settings for optimizing the FLC grouping neurons with Lung's output, C. Elegans Hodgkin Huxley

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	28.8 h

Table A.17: Experimental settings for optimizing the FLC grouping neurons with redefined output, C. Elegans Hodgkin Huxley

#Models	Learn. rate	Δ_t	#Train	#Test	Data length	Duration
10	$0.4 \cdot 0.95^{\lfloor n/9 \rfloor}$	0.2ms	1	0	5s	28.8 h

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] codecov.io. https://codecov.io/gh.

[2] Travis ci. https://travis-ci.org/,. Accessed: 2018-08-01.

[3] Understanding lstm networks - post on colah's blog, 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/,. Accessed: 2018-06-13.

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[5] L. F. Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304, 1999.

[6] P. Achard and E. De Schutter. Complex parameter landscape for a complex neuron model. *PLoS computational biology*, 2(7):e94, 2006.

[7] J. Akerboom, T.-W. Chen, T. J. Wardill, L. Tian, J. S. Marvin, S. Mutlu, N. C. Calderón, F. Esposti, B. G. Borghuis, X. R. Sun, et al. Optimization of a gcamp calcium indicator for neural activity imaging. *Journal of Neuroscience*, 32(40):13819–13840, 2012.

[8] E. L. Ardiel and C. H. Rankin. An elegant mind: learning and memory in caenorhabditis elegans. *Learning & Memory*, 17(4):191–201, 2010.

[9] R. D. Beer. Parameter space structure of continuous-time recurrent neural networks. *Neural Computation*, 18(12):3009–3051, 2006.

[10] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[11] U. S. Bhalla and J. M. Bower. Exploring parameter space in detailed single neuron models: simulations of the mitral and granule cells of the olfactory bulb. *Journal of neurophysiology*, 69(6):1948–1965, 1993.

[12] G. Brandl. Sphinx documentation. *URL http://sphinx-doc. org/sphinx. pdf*, 2010.

[13] M. Chalfie, J. E. Sulston, J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. The neural circuit for touch sensitivity in caenorhabditis elegans. *Journal of Neuroscience*, 5(4):956–964, 1985.

[14] S. Druckmann, 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.

[15] R. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal*, 1(6):445–466, 1961.

[16] W. Foster, L. Ungar, and J. Schwaber. Significance of conductances in hodgkin-huxley models. *Journal of neurophysiology*, 70(6):2502–2518, 1993.

[17] M. Fuchs, M. Zimmer, R. Grosu, and R. M. Hasani. Searching for biophysically realistic parameters for dynamic neuron models by genetic algorithms from calcium imaging recording. *arXiv preprint arXiv:1711.01436*, 2017.

[18] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[19] M.-O. Gewaltig and M. Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[20] M. S. Goldman, J. Golowasch, E. Marder, and L. Abbott. Global structure, robustness, and modulation of neuronal models. *Journal of Neuroscience*, 21(14):5229–5238, 2001.

[21] D. Goodman and R. Brette. The brian simulator. *Frontiers in Neuroscience*, 3:26, 2009.

[22] M. B. Goodman, D. H. Hall, L. Avery, and S. R. Lockery. Active currents regulate sensitivity and dynamic range in c. elegans neurons. *Neuron*, 20(4):763–772, 1998.

[23] J. Guckenheimer, S. Gueron, and R. M. Harris-Warrick. Mapping the dynamics of a bursting neuron. *Phil. Trans. R. Soc. Lond. B*, 341(1298):345–359, 1993.

[24] M. Hines. The neurosimulator neuron. *Methods in neuronal modeling*, pages 129–136, 1998.

[25] M. L. Hines and N. T. Carnevale. The NEURON simulation environment. *Neural Computation*, 9(6):1179–1209, Aug 1997.

[26] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] A. L. Hodgkin 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):500–544, 1952.

[28] D. B. James M. Bower. The book of genesis: Exploring realistic neural models with the general neural simulation system. *Springer*, 1997.

[29] S. Kato, H. S. Kaplan, T. Schrodel, S. Skora, T. H. Lindsay, E. Yemini, S. Lockery, and M. Zimmer. Global brain dynamics embed the motor command sequence of Caenorhabditis elegans. *Cell*, 163(3):656–669, 2015.

[30] N. Keren, N. Peled, and A. Korngreen. Constraining compartmental models using multiple voltage recordings and genetic algorithms. *Journal of neurophysiology*, 94(6):3730–3742, 2005.

[31] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[32] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[33] C. Koch and I. Segev. *Methods in neuronal modeling: from ions to networks*. MIT press, 1998.

[34] M. Lechner, R. M. Hasani, and R. Grosu. Neuronal circuit policies. *arXiv preprint arXiv:1803.08554*, 2018.

[35] K. Leung, A. Mohammadi, W. S. Ryu, and I. Nemenman. Stereotypical escape behavior in caenorhabditis elegans allows quantification of effective heat stimulus level. *PLoS computational biology*, 12(12):e1005262, 2016.

[36] D. Lung. A simplified cell network for the simulation of c. elegans' forward crawling. *Workshop on Worm's Neural Information Processing at the 31st Neural Information Processing Systems (NIPS) Conference, Long Beach, CA, USA, 2017.*, 2017.

[37] R. M. Hasani, V. Beneder, M. Fuchs, D. Lung, and R. Grosu. SIM-CE: An advanced simulink platform for studying the brain of Caenorhabditis elegans. *arXiv preprint arXiv:1703.06270*, 2017.

[38] H. Mania, A. Guy, and B. Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[39] J. S. Marvin, B. G. Borghuis, L. Tian, J. Cichon, M. T. Harnett, J. Akerboom, A. Gordus, S. L. Renninger, T.-W. Chen, C. I. Bargmann, et al. An optimized fluorescent probe for visualizing glutamate neurotransmission. *Nature methods*, 10(2):162, 2013.

[40] J. E. Mellem, P. J. Brockie, D. M. Madsen, and A. V. Maricq. Action potentials contribute to neuronal signaling in c. elegans. *Nature neuroscience*, 11(8):865, 2008.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[42] C. Morris and H. Lecar. Voltage oscillations in the barnacle giant muscle fiber. *Biophysical journal*, 35(1):193–213, 1981.

[43] F. Nadim, Ø. H. Olsen, E. De Schutter, and R. L. Calabrese. Modeling the leech heartbeat elemental oscillator i. interactions of intrinsic and synaptic currents. *Journal of computational neuroscience*, 2(3):215–235, 1995.

[44] Ø. H. Olsen, F. Nadim, and R. L. Calabrese. Modeling the leech heartbeat elemental oscillator ii. exploring the parameter space. *Journal of computational neuroscience*, 2(3):237–257, 1995.

[45] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[46] A. A. Prinz, C. P. Billimoria, and E. Marder. Alternative to hand-tuning conductance-based models: construction and analysis of databases of model neurons. *Journal of neurophysiology*, 90(6):3998–4015, 2003.

[47] A. A. Prinz, D. Bucher, and E. Marder. Similar network activity from disparate circuit parameters. *Nature neuroscience*, 7(12):1345, 2004.

[48] C. H. Rankin, T. Abrams, R. J. Barry, S. Bhatnagar, D. F. Clayton, J. Colombo, G. Coppola, M. A. Geyer, D. L. Glanzman, S. Marsland, et al. Habituation revisited: an updated and revised description of the behavioral characteristics of habituation. *Neurobiology of learning and memory*, 92(2):135–138, 2009.

[49] K. Reitz and T. Schlusser. *The Hitchhiker's Guide to Python: Best Practices for Development*. " O'Reilly Media, Inc.", 2016.

[50] W. M. Roberts, S. B. Augustine, K. J. Lawton, T. H. Lindsay, T. R. Thiele, E. J. Izquierdo, S. Faumont, R. A. Lindsay, M. C. Britton, N. Pokala, et al. A stochastic neuronal model predicts random search behaviors at multiple spatial scales in C. elegans. *Elife*, 5:e12572, 2016.

[51] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[52] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[53] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[54] B. B. Shtonda and L. Avery. Dietary choice behavior in caenorhabditis elegans. *Journal of experimental biology*, 209(1):89–102, 2006.

[55] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[56] C. Soto-Trevino, P. Rabbah, E. Marder, and F. Nadim. Computational model of electrically coupled, intrinsically distinct pacemaker neurons. *Journal of neurophysiology*, 94(1):590–604, 2005.

[57] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[58] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[59] B. Szigeti, P. Gleeson, M. Vella, S. Khayrulin, A. Palyanov, J. Hokanson, M. Currie, M. Cantarelli, G. Idili, and S. Larson. OpenWorm: an open-science approach to modeling Caenorhabditis elegans. *Frontiers in computational neuroscience*, 8:137, 2014.

[60] A. M. Taylor and R. M. Enoka. Optimization of input patterns and neuronal properties to evoke motor neuron synchronization. *Journal of computational neuroscience*, 16(2):139–157, 2004.

[61] L. R. Varshney, B. L. Chen, E. Paniagua, D. H. Hall, and D. B. Chklovskii. Structural properties of the caenorhabditis elegans neuronal network. *PLoS computational biology*, 7(2):e1001066, 2011.

[62] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philos Trans R Soc Lond B Biol Sci*, 314(1165):1–340, 1986.

[63] S. R. Wicks, C. J. Roehrig, and C. H. Rankin. A dynamic network simulation of the nematode tap withdrawal circuit: predictions concerning synaptic function using behavioral criteria. *Journal of Neuroscience*, 16(12):4017–4031, 1996.

[64] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.