Università degli Studi di Milano - Bicocca
Department of Psychology
Psychological Sciences

Autonomously Learning Parameters to Simulate Biologically Plausible Portions of the Brain

Thesis supervisor
Prof. Alice Mado Proverbio

Author
Stefano Bettani
Badge number: 828033
N. of characters: 40000

Academic year 2019-2020

Contents

1 Abstract

Brain simulations are often used in neuroscience to test theoretical models of computation. Understanding the low-level dynamics of the brain is fundamental to develop better artificial intelligence algorithms, and in turn those algorithms provide more sophisticated methods to crack the brain code. Still, building simulations of sections of the brain is complex, and finding the parameters of the neurons and synapses so that the behavior of the artificial network matches the biological one requires a lot of time and resources. Here it is developed a simple framework to automate the search for the best parameters for the simulation, and is provided a use case with a model of a hypercolumn and a Reinforcement Learning algorithm to optimize the parameters so that the firing rate of the excitatory populations in every layer matches the activity recorded in vivo. The results show that Reinforcement Learning obtains results that exceed in accuracy the referenced papers.

2 Theoretical Introduction

> "Computational Neuroscience is an approach to understanding the information content of neural signals by modeling the nervous system at many different structural scales, including the biophysical, the circuit, and the systems levels. Computer simulations of neurons and neural networks are complementary to traditional techniques in neuroscience."
>
> (Dayan & Abbott, 2005, Series Foreword)

Understanding how the brain works is one of the hardest challenges the humanity has ever faced, and it is crucial in the progress of Science: understanding intelligence, the learning process and consciousness would be groundbreaking for the developing of new technologies.

One of the main issues in the study of the brain right now is retrieving the data: we have enough computational power to process signals from billions of neurons at the same time, but we can't just put an electrode in everyone of them, although some are going in that direction (Musk, 2019).

There are other widely used technologies that are able to records brain metabolism or electromagnetic fields and currents over the scalp, and they can help to figure out correlation between high level cognitive functions and active brain areas, but they don't offer insights on low level computations. Cellular biology is very interested in synaptic transmission, and their research is inspiring Deep Learning researchers to try understand how the brain uses action potentials to carry information, and how the computations occur from a algorithmic point of view.

Deep Learning and Neuroscience

Research in neuroscience has historically been driven by a biological approach: it has been focused on how the cells were structured, on how the genome influenced the production on neurotransmitters and how those neurotransmitters could shape behavior.

Then, a movement, mostly inspired by physics, started interesting on circuits: how does the electromagnetic waves of lights become perception? How does we decide and control our movement in the space? Many instrumentation was developed to study the brain while on task, like EEG, MEG, PET and fMRI, but none of them could give us the same insight as the one given by invasive techniques such as in cell recordings.

Lastly, around ten years ago (LeCun, Bengio, & Hinton, 2015), Deep Learning was rediscovered, thanks to the development of high parallel computing power (GPUs), and the real potential of artificial neural networks came to light.

Deep Learning started having a profound impact on Neuroscience, not only thanks to the new methods that it carried for analyzing data and comparing circuits and their behaviours, but also for the change in objective: Cognitive and Computational Neuroscience was aimed to understand the brain and how it works, but the most prominent use cases were clinical. With Deep Learning, discovering a more efficient learning algorithm is worth billions of dollars and, like it or not, this influences research heavily.

Right now, understanding how the brain works, how the mind works, has the potential to exhibit new ways to make machines learn, to shed light on the most efficient ways to program an algorithm able to do what only living beings seem to do well: perceiving, orienteering, understanding language, producing language and ultimately reasoning.

The underlying hypotheses that sustain the research are a) that understanding how the brain solves computational challenges can help to build better algorithms and b) that better algorithms hopefully allows for a deeper comprehension of the brain. The developed algorithms will enable more efficient methods to study many areas, like computational biology (Angermueller, Pärnamaa, Parts, & Stegle, 2016), traffic flow prediction (Koesdwiady, Soua, & Karray, 2016), improvement of existing techniques of brain imaging (Milletari et al., 2017), financial market prediction (Fischer & Krauss, 2018) and many more.

Although the perspective of a machine able to "think" is still very far away, research is obtaining incredible results with deep learning, and it isn't showing signs of a slow.

What is missing though, is a deeper connection between Neuroscience research and AI research. A recent paper (Richards et al., 2019) promotes this link, proposing to apply the AI framework to the study of Neuroscience, and pointing out that we absolutely need advances in the latter to proceed with the first.

In particular, understanding how brain connectivity is able to determine certain kinds of computations, is a topic of major interest in the field of Deep Learning, specifically to be able to design network's architectures that obtain high performance on the set of

tasks that living beings excel at.

A use case is to compare an algorithm for a given task, like visual recognition, to the brain circuit that executes that task, in this case the ventral system of vision, to understand if there are similarities in the behavior and if they correlate with higher performance.

To do so it is necessary to record and understand how the system of interest works, how it is connected, and how it reacts to a stimulus. This can be very hard, because with the technologies available in this time this usually requires an invasive approach, so it can't be done on many humans. This is where simulations come into play: a simulation can help infer the internal dynamics of the given system without requiring full knowledge of it.

Still, inference is usually computationally complex and non-linear, so finding the exact parameters for every neuron and every connection in a big model, while respecting the imposed thresholds and without full knowledge of the system is very challenging for an ordinary algorithm.

What is Deep Learning

Deep Learning is a machine learning method, based on an extension of the *perceptron* (Rosenblatt, 1958), and can be used as classifier or to approximate functions. A Deep Learning architecture, a Deep Neural Network, is composed of nodes and links, where the former are characterized by an *activation function*, and the latter by a weight, that usually is random first and is changed to optimize the outcome thanks to Stochastic Gradient Descent (Robbins & Monro, 1951) or an evolution of it. The nodes are organized in layers, and they can be fully connected or have convolutional or recursive features. The first layer is called "input layer", the last one "output layer" and the ones in between them are called "hidden layers". The input layer is a vector of nodes, and has the same shape of the data. The hidden layers can be of any size, and there can be as many as is wished. The size of the output layer is equal to the number of the classes, in the case of a classifier model, or equal to the output size of the function that it is intended to approximate.

The activation function is a nonlinear function, usually a Rectified Linear function for the nodes in the hidden layers and a specific function in the output layer, depending on the context. There are many parameters to be defined in a Deep Neural Network, but the more prominent are the learning rate, the depth and the activation functions.

Simulations

A less invasive solution is to use our knowledge of the brain to build complex, biologically plausible simulations of the brain, or parts of the brain, and then retrieve from them all the data that is needed. This data can then be used to predict behaviors of real networks, to understand non-linear dynamics, or to test whether the understanding of a phenomena

is accurate enough to provide all the information needed to reproduce it.

However, this approach, deeply analyzed in the past (Smolensky, 1988), doesn't offer assurance that the simulation and the real process could actually be interchangeable. In the present, many of the challenges of that time were overcome, in particular computational power (Djurfeldt et al., 2008), and simulations are used to confirm theoretical models of the brain.

The Framework

Here is proposed a simple framework to guide the search for the parameters of a model of the brain, in an autonomous fashion and with high accuracy results.

The elements of the framework are just the simulation model and the optimization algorithm, and they are characterized by the fixed and the free parameters for the simulation, while for the algorithm is featured by its loss function, so which output of the simulation is evaluated and how, its k parameter, representing the scale of the output, and its structure.

In this study the simulation model, that will be described in more detail later, is an hypothetical column of the sensory cortex of a mammal, developed by Potjans (Potjans & Diesmann, 2014), the free parameters are a subset of the ones they found to minimize the difference of the firing rate of the excitatory neurons in four different layers of the simulation between the firing rates of the respective population and layers in the in vivo situation. So the firing rate is the output of the simulation that is evaluated. The k parameters will vary between different simulation to find the most adeguate for each case. Finally, the structure of the optimization algorithm will be derived from a Reinforcement Learning Algorithm, namely Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), whose functioning will be explained in the next section.

2.1 Reinforcement Learning

Reinforcement Learning is an area of machine learning which considers a setup consisting of an agent interacting with an environment through actions. The agent receives an input from the environment, chooses an action and receives a reward, based on the state of the environment after the action.

For example, the environment could be a model of a neuron connected to a spike detector, and the action at disposal of the agent could be to change the constant background current of the neuron. If its objective is to have the neuron spike with a specific rate, like 4 spikes per second, then the reward could be the absolute difference between the firing rate of the previous second and the one desired.

The agent would be rewarded less when it achieves a firing rate very different from 4 spikes per second, and more when it is close, so it will gradually learn to approach the

right value.

This way of modeling the problem of learning is called *Markov Decision Process*, and it provides a mathematical framework that let artificial agents make optimal decisions in a noisy environment where the agent has missing information about its internal dynamics.

Reinforcement Learning has shown the potential to understand highly complex observation spaces, as in the game of go (Silver et al., 2017), in videogame (Vinyals et al., 2019), (Berner et al., 2019), robotic control (Andrychowicz et al., 2020) and autonomous vehicles systems (Sallab, Abdou, Perot, & Yogamani, 2017), but most importantly, it has been showed in some domains, such as predicting protein structure (Senior et al., 2020), that Deep Reinforcement Learning is well suited to learn complex features of models and act to modify them in a desired direction.

The actor-critic

DDPG belongs to the family of actor-critic algorithms, which use an action-value function (called the critic) to process the distribution of rewards over actions, and a policy (called the actor) to find the best possible action.

Actor-critic algorithms were developed to overcome the limitations of "actor-only" and "critic-only" algorithms, as stated in the paper that introduced them:

> - Actor-only methods work with a parameterized family of policies. The gradient of the performance, with respect to the actor parameters, is directly estimated by simulation, and the parameters are updated in a direction of improvement. A possible drawback of such methods is that the gradient estimators may have a large variance. Furthermore, as the policy changes, a new gradient is estimated independently of past estimates. Hence, there is no "learning," in the sense of accumulation and consolidation of older information.
> - Critic-only methods rely exclusively on value function approximation and aim at learning an approximate solution to the Bellman equation, which will then hopefully prescribe a near-optimal policy. Such methods are indirect in the sense that they do not try to optimize directly over a policy space. A method of this type may succeed in constructing a "good" approximation of the value function, yet lack reliable guarantees in terms of near-optimality of the resulting policy.
>
> (Konda & Tsitsiklis, 2000)

The critic takes the value of an action and the reward obtained executing it and tries to shape a function that assigns to every action a value, called Q-value, that represent

the expected reward after taking the action. The policy (the actor) defines the agent behavior, and its job is to take the present state as input and to choose an action.

To train the actor, DDPG uses the critic: in the previous example, the actor training process would be to take a set of actions in some random states, and then the critic would evaluate them, and the negative mean loss of the critic would be the loss of the actor. The actor and the critic are function approximation algorithms, in this case Artificial Neural Networks (ANNs).

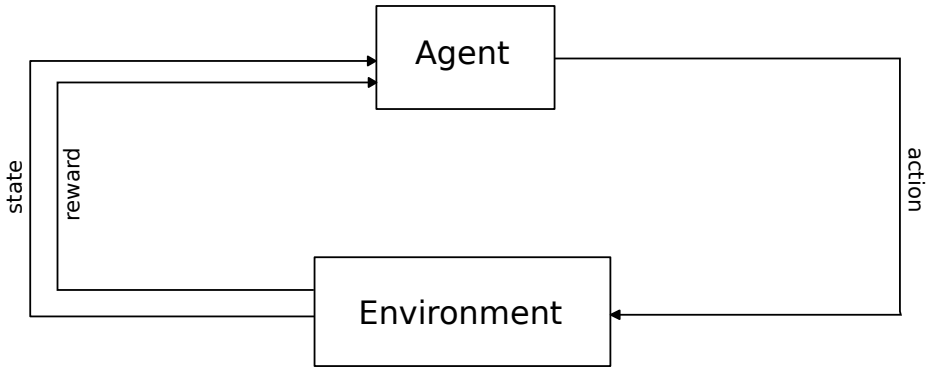

Figure 1: A Generic Reinforcement Learning setup

Multi-armed bandit

The setting for this study has one difference with the one of the example above. While in that case every action produces another state, and the agent will have to take another action on that state, in this study the environment reaches the terminal state after one single transition, and the agent has to choose a value for many different actions in a continuous space.

This kind of setup is called multi-armed bandit (Auer, Cesa-Bianchi, Freund, & Schapire, 2002), and the main problem to solve is the balance between exploration and exploitation: we want the agent to take the best possible action, but that could prevent him to explore the environment and find even better actions.

One solution is to add simple Gaussian noise to the actions, that decays slowly as the training goes on. This way the agent is able to point in one direction, but can choose a precise value only after a high number of episodes, when he has explored the environment enough.

There are more sophisticated way to solve this problem, for example computing an estimate of the confidence on the reward obtained by taking some combination of the actions, and imposing a loss function that minimizes its values for lower values of confidence, so that the agent is "motivated" to explore all the action space. This kind of solution can be useful in a setup where the environment changes over time, as it is in many robotic tasks, but it could end up slowing down the training process in the simplified case of the multi-armed bandit (Wawrzynski, 2015).

2.2 The simulation model

Cortical columns

The vast majority of cortical cortex is neocortex, the part of our brain that has developed most recently. The building blocks of neocortex are the cortical columns, defined as portions of the brain that develop perpendicular to the surface of the brain, and every column has a different receptive field from the others.

Cortical columns themselves are organized in 6 layers, that differ for cell type composition and connections with other layers and areas of the brain. Cells belongs to excitatory or inhibitory populations, and the pattern of activity that the cortical columns show, consistent between different areas and mammals (de Kock & Sakmann, 2009), is the result of the dynamical interaction of neurons in the network.

The layer 1 of the cortex is not included in the model, due to the low number of neurons contained in it, and the second and third layers are considered together, as the differences in neuron types are not considered.

Integrate and fire neurons

Neurons are not easy to simulate, and there are models that tries to comprehend all their complexity, like Hodgkin-Huxley's, but the expense in terms of parameterization and computational weight are steep. Furthermore, in many cases, simpler neuron models like integrate and fire provide a good approximation of more complex models (Bernander, Douglas, Martin, & Koch, 1991). Here there is an exhaustive explanation of the integrate and fire model, which will be used in this study:

> "The integrate-and-fire neuron model is one of the most widely used models for analyzing the behavior of neural systems. It describes the membrane potential of a neuron in terms of the synaptic inputs and the injected current that it receives. An action potential (spike) is generated when the membrane potential reaches a threshold, but the actual changes associated with the membrane voltage and conductances driving the action potential do not form part of the model. The synaptic inputs to the neuron are considered to be stochastic and are described as a temporally homogeneous Poisson process."
>
> (Burkitt, 2006)

3 Material and Methods

3.1 The objective

This study aims to demonstrate the effectiveness of Reinforcement Learning algorithms in optimizing parameters for network-scale brain simulation. Furthermore, it is a mean to test Potjans and Diesmann's thesis, sustaining that the connectivity of a biological network is the main factor that contributes in the shaping of firing rates. It could be seen as an extension to Potjans and Diesmann's research, attempting to leave the task of finding the right parameters to an algorithm, in a sense reverse engineering the recorded activity rather than reaching it through induction.

3.2 The methodology

A modified version of the DDPG algorithm was trained on a simulation of a brain column, where it was able to modify different parameters in different settings, explained in detail later. Its objective was to minimize the difference between the firing rates of the simulation and the ones recorded from in vivo settings. Specifically, the activity it was trying to match was the one provided from the study of De Kock et al. (de Kock & Sakmann, 2009), that has also been used from Potjans and Diesmann to compare their result. This way a comparison between the three results will be coherent.

The data (de Kock & Sakmann, 2009) comes from rats in resting state, with the head fixed in position, and has been chosen over other recordings because it has been found to be the more complete and reliable, even if it only includes the activity of excitatory populations.

This experiment provides both an instrument for future research to exploit breakthroughs in close fields, like machine learning, and the opportunity to understand which how changing parameters shapes the output of a network.

In order to achieve that understanding, the experiment was repeated 4 times, each of which has different parameters modified:

1. Neuron parameters
2. Synapses parameters
3. Connection probabilities
4. All at once

Refer to appendix A.1 for a more accurate description of what was changed in each setting.

3.3 The environment

The brain column

The environment in which the agent acted was a scaled version of the Potjans and Diesmann column, where the number of neurons is $8 \cdot 10^3$ instead of $8 \cdot 10^4$. That was necessary to reduce computational time. The neurons belong to excitatory and inhibitory populations, and are organized on four different layers, that represent the layers 2/3, 4, 5 and 6 in the sensory cortex of a rat.

For the simulated column to be a suitable environment for training a DDPG agent, it needs to provide both feedback on its activity and a way to change it.

The reward

The only input to the agent was the reward, calculated as the negative squared root of the absolute difference between the firing rates of the excitatory neurons in the layers 2/3, 4, 5 and 6 recorded from the somatosensory cortex of a rat (de Kock & Sakmann, 2009), and the firing rates of the excitatory neurons in the same layers. Computing the reward in this way is not optimal, because the agent doesn't know in which layer the error was, but right now it doesn't seem to exist a better method.

The parameters

All the starting parameters are the ones provided by Potjans and Diesmann, in particular the ones regarding synapses and connections, while neuron parameters are also the same of the random balanced network. The agent directly changes them, as will be described later. Parameters aren't constrained in any way, other than trying to reduce their change and not adding connections between populations that were not already linked, for example when connection probability equals 0.

Potjans and Diesmann's Model

The starting point of this study is the model developed by Potjans and Diesmann (Potjans & Diesmann, 2014), as it was for many others (Wagatsuma, Potjans, Diesmann, & Fukai, 2011), (Lindén et al., 2011), (Shimoura et al., 2018).

The authors (Potjans & Diesmann, 2014) expanded the random balanced network (Brunel, 2000) to a multi-layered case, comprising integrate-and-fire neurons, exponential currents and following the parameterization of the starting model. In particular, to achieve biologically plausible behaviors of the network, the synaptic relative strength of inhibitory connections is multiplied by a factor of 4.

The model of cortical column considered is still assumed to be anatomically uniform, with only 2 populations of neuron, distributed in different proportion between 4 layers. In reality there are many different kind of neurons and other cells, but trying to simulate

those is both very computationally expensive and not that determinant in terms of the considered outcome.

Their simulation aims to be the minimal extension of the mono-layered model of the random balanced network. It only integrates static connections between neurons, and the set of cell types is the smallest typically distinguished in experiments.

The focus of this research was over the role of connectivity between populations of neurons in shaping firing rates. To achieve a proper connectivity map to emulate the activity of biological network, the authors (Potjans & Diesmann, 2014) combined data from previous physiological and anatomical studies on mammal's neocortex, through an algorithmic procedure and following constraints to adhere with biological data.

The connectivity maps belonged to different mammals and different areas, because the coherent data from the same area and animal was not available. Furthermore, anatomical and physiological recordings produce different biases, that was taken into account in the study. Nevertheless, the connection probabilities originate from this data will only be applicable over a theoretical generic column, not a specific one.

The parameters were chosen in a way such that excitatory postsynaptic potentials has an average amplitude of 0.15 mV with a rise time of 1.6 ms, imitating the in vivo situation. The delays in the network are independent of the layer, with delays of excitatory populations on average twice as long as delays of inhibitory populations, based on in vivo data (Thomson & Bannister, 2003). Excitatory connections from layer 4 to layer 2/3 are doubled, because the data was on them was inconclusive, but they affect only marginally ongoing activity.

The firing rates were compared with the values recorded on other studies (de Kock & Sakmann, 2009) and the trend showed in both cases was similar, so that the lowest rates were shown in layers 2/3 and 6, and the highest in layer 5 (Figure 2), yet it wasn't accurately matched.

3.4 The agent

The algorithm

As already said, the model deployed in this study is a modified version of DDPG. The choice of the algorithm was based on many factors, but the most important feature was that it had to be able to change the parameters of the simulation, that are continuous values, so it had to be able to compute continues actions. This is best achieved by algorithms that are intended to be used in the robotic context, exactly like DDPG.

Adaptation to the bandit problem

Being the environment shaped as a multi-armed bandit problem, the agent find itself always in the terminal state, so it doesn't have to compute state Q-values for concatenated

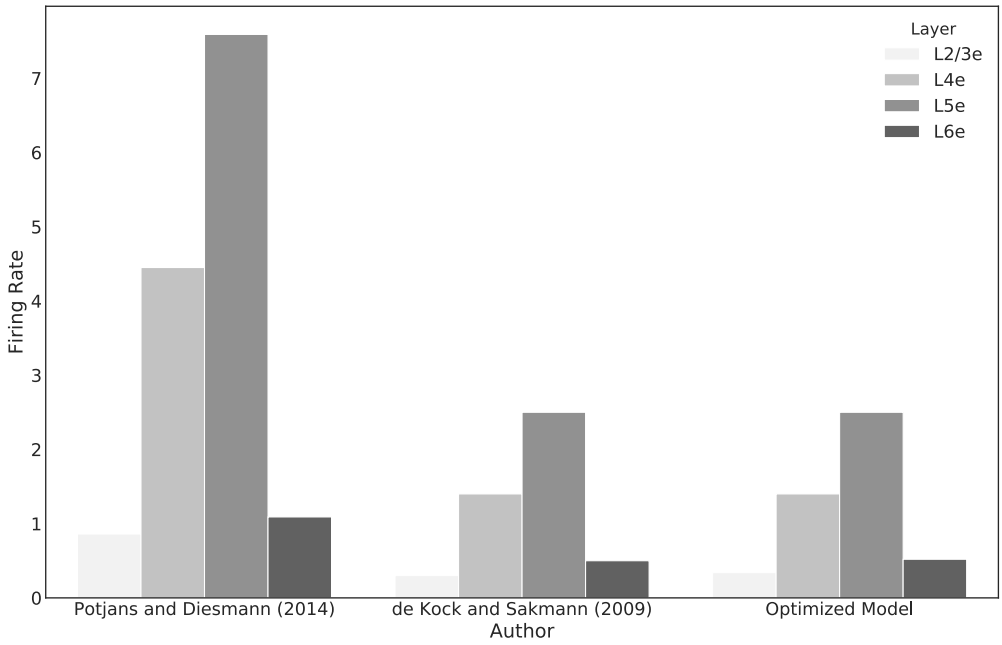

Figure 2: Comparison of firing rates from the different studies

observations, and the DDPG model could be simplified, removing the target networks.

Parameters update

The output layer of the actor model has a hyperbolic tangent as activation function, that bounds the output values between 0 and 1. This leaves the possibility to modulate the effect that each action has on the parameters. Specifically, if the p is the vector of parameters to change, the actions are the vector a of values $a_1, a_2 \dots a_n$ between 0 and 1 with n as the number of parameters to change, and $k > 1$ a constant that constrains the action, the update was done by the equation:

$$p_i^{new} = p_i + \frac{a_i \cdot p_i}{k} \tag{1}$$

That is important to reduce the space where the agent looks for the values of the parameters, but has to be manually tuned.

3.5 The whole model

Having described how the main parts of the model work, here it is an overview of the whole training process.

Generating the agent

Following the original paper (Lillicrap et al., 2015) the agent is created with the weights of both the actor and the critic initialized in such a way so that the first actions that it takes are close to zero, that helps a stable and reproducible training: in fact it could happen that a particular set of starting weights produce actions such that the model is already close to the solution, but that would be just random.

The noise

Then each of the first actions of the agent will be just the noise generated by a Gaussian distribution with 0 as mean and standard deviation equal to 1. The noise is maximum in the start so that the agent is forced to explore a relatively large space before deciding were to focus. After around 7000 iterations the noise will be halved (its half-life is $7 \cdot 10^4$).

The interaction with the environment

When the agent takes an action what it really happens is that:

1. The actor generates a vector of actions
2. The agent adds the noise to the vector
3. The vector get fed in the environment and divided by k
4. The vector is turned into a sort of dictionary, where every value is associated with its key
5. The brain column is built retrieving the values using the keys
6. The brain column is simulated for 1000 ms, and the spikes of 500 ms are counted and averaged with respect to their population and layer
7. The resulting fire rates are returned to the program

Training the agent

The agent is trained on a random batch of past episodes, using the recorded data. The randomness helps with stabilizing the training, and training on a batch is much more efficient for neural networks on GPU architectures, thank to parallelization.

Usually, after the first few thousands of training steps, the model converges to low values of errors, and slowly approaches the 0 error. This deceleration could be due to the problem mentioned above of optimizing multiple target values at the same time. The noise is critical in this period, because without it the agent would just repeat the action he experienced to yield the best results, and the learning process would stop.

4 Results

4.1 Changing neuron parameters

As Potjans and Diesmann suggested (2014), neuron parameters do not play a big role in shaping the network behavior. This obviously doesn't mean that neurons don't play a fundamental role in the simulation, but it could hint that their approximation is already good enough to provide plausible results.

In fact, in almost 40000 episodes the agent's best result was ≈ 2.20 mean absolute error. Although it is an improvement of the original model, where the error was ≈ 9.26, it is yet relatively far from the biological data.

Nevertheless, we have to consider that the agent was able to change the parameters for all neurons at once, differentiating only between excitatory and inhibitory populations. The k factor dividing the actions was 20 during the first run, but the results looked capped at low values, so it was lowered to 5 in a subsequent run, to leave the agent a wider range of action.

4.2 Changing synapses parameters

Synapses parameters control the amplitude of postsynaptic potential, the delays of the connections and the relative strength of inhibitory synaptic connections. Potjans and Diesmann (2014) called this value g and they imposed $g = 4$, to reproduce the dynamics exhibited by the random balanced network through the dominance of inhibitory connection over the excitatory ones.

In this setup the agent found that the value of g that optimized the output was ≈ 4.4, which is a further step in the direction of strong inhibitory synapses. Moreover, analyzing the data from the training steps, a it is shown a negative correlation between error magnitude and the mean delay of inhibitory connections, while when the delay of excitatory connections increased, the error shrank. Still, the parameter that looked more influential was the g value, while the one with lowest correlation with the reward was the standard deviation of the amplitude of excitatory post-synaptic potentials.

Synapses parameters' optimization was only slightly more effective than neuron's, with an ≈ 1.96 mean absolute error with $k = 5$. This probably was because considered synapses parameters, as neuron's, mostly regarded universal thresholds. Beside that, synapse parameters was only 6, against the 26 of neurons and 64 of structure. That certainly reduced the power of the agent on the simulated brain column.

One possibility at this time could be to reduce the k factor to let the agent have wider actions, but, the best results weren't achieved by extreme actions, so it wouldn't have been meaningful to do that.

4.3 Changing structure parameters

The scenario where the model was trained to change the structure parameters was the most successful between the first three, and it achieved an absolute error of ≈ 0.031 in around 40000 episodes.

Every action was divided by a factor of 20, to restrict the action range. That was necessary considered that the structure parameters were 64 values, making the problem highly complex to solve.

	Layer 2/3	Layer 4	Layer 5	Layer 6
Baseline	0.751	4.29	6.816	1.126
Results	0.29	1.40	2.52	0.50
Target	0.30	1.40	2.50	0.50

Table 1: Comparison between models

The model was tested every 200 episodes without noise to monitor its learning curve. As you can see in the picture 3, after the first 20000 episodes the model converges slowly almost to the 0.

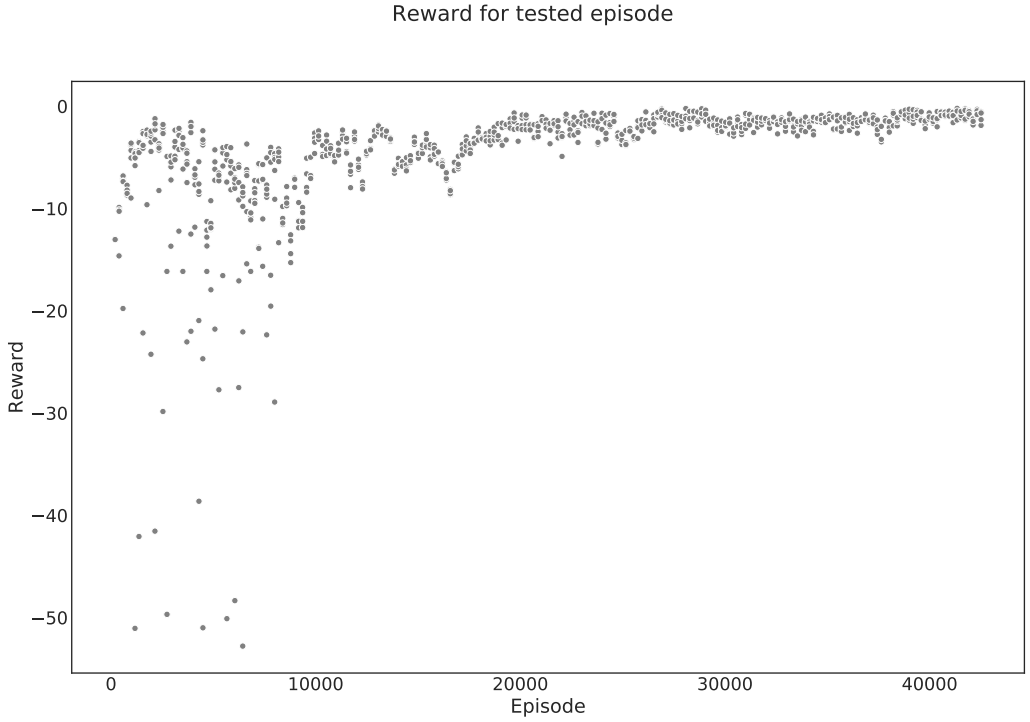

Figure 3: Absolute sum of errors

4.4 Changing all parameters

Interestingly, this case did not reach the results of the structure's scenario, only by less then ≈ 0.01 absolute error, but to real difference was in the actions that it took to reach that error: while in the "pure" model the actions' absolute values were relatively small, in this case almost every action was at his limit, -1 or 1.

A possible interpretation can be that changing the synapses', neurons' and structure's parameters altogether, the agent actually optimizes the structure's parameters in a non-optimal state of the other parameters. In this case the agent would only need more training time, to be able to explore a larger share of its action space.

5 Conclusions

The objective of this research is not to discover better parameters for the Potjans and Diesmann model, yet it was intended as an example aimed to prove the potential of the application of Deep Reinforcement Learning on brain simulations.

In this particular case, the target for the agent was to match the firing rates of the excitatory populations in the different layers, and it is highly promising not only the accuracy with which the values matched, but also the fact that the inhibitory firing rates were adjusted properly, maintaining the biological plausibility.

In the future, this kind of approach could be applied to both conduct and endorse the research of biological parameters of the brain, such as population numerosity, connectivity, as in this case, but also to reverse-engineer the effects of diseases on specific areas of the brain.

Obviously, this approach would be extremely useful in a whole-brain simulation, but since this is not feasible in a convenient way, it could help to make realistic approximations of it, shaping parameters in way that makes the simulation behave properly.

Improvement of the model

A further improvement could be reached applying different layers of optimizations one after another, so that for example neuron's parameters are optimized first, then synapses' and lastly structure's. To do so without "over-shooting", it would be possible to impose thresholds to the first sets of parameters.

Another approach that would minimize the chance of finding non-realistic parameters could be to change the loss function of the actor, adding a constraint on the size of actions, so that larger actions produce a higher loss. With this device, the agent would be inclined to take the lowest possible actions that achieve to get the higher rewards. The right balance between them could be obtained tweaking a constant that multiplies the constraint.

Expanding the scope

This kind of framework could obtain notable results modifying the setup so that there are different sets of target values, obtained in different situations, for example in resting state and while executing a task. This way the agent would learn to choose the parameters in a coherent way, so that the simulation will be more functionally realistic.

Deep Learning Integration

The framework here provided hopes to be part of the process of connection between Deep Learning and Neuroscience research. The scientific community could greatly benefit this conjunction, and if there is a path that both the Artificial Intelligence and the

Neuroscience current can walk together, then it should be done.

Our brain is able to solve the problem of general intelligence in a way that no algorithm can even dream at this time. This problem arises from the *No Free Lunch Theorem* (Wolpert & Macready, 1997), that states that every algorithm has the same expected performance over all tasks.

This is mathematically proven, and what it means is that it doesn't exist a "better algorithm" in itself, but only a more fit one. Now, despite the Artificial Intelligence research has always aimed to build a "general intelligence", it never intended to build an intelligence better at *every* task: that would mean to be able to solve problems that never arises in our physical world, or dynamics that cannot exist on the Earth, so that would be pointless. Nonetheless, the No Free Lunch Theorem clarified that the research effort shouldn't focus blindly on building something more powerful, like a huge calculator that can solve any computation in a matter of nanoseconds, but it should aim to intentionally excluding some kind of problems from the reach of the algorithm, including only the meaningful tasks.

This is elaborated also in recent works, (Richards et al., 2019), where the authors refer to this selection as an "inductive bias". For example, a Deep Neural Network has a bias on hierarchically shaped problems, because of the way the computations are carried on in its layers. This is yielding great results in the set of tasks that humans and animals are good at because our world is often shaped hierarchically: for example our language is structured in texts, composed with sentences, built with words that many times has prefixes or suffixes that bear their own meaning.

The brain is very good at it, in fact Deep Neural Networks are derived from its structure and the way it works. This promises that understanding other mechanisms too, like how to adjust the trajectory of a movement, or coping with partial information, would help obtain excellent results if applied through artificial algorithms.

Controversies - Why do we need an Artificial General Intelligence

All the effort in the direction of building a "super intelligence" looks natural: it could lead to the automation of nearly every boring process humanity needs, and it would be error-free and more efficient at the same time. It could help the world to deal with the huge issues we are facing, like global warming, poverty and inequality.

This looks all very promising, but it is only speculation: reality is that we don't know what would really happen if a super-human intelligence will be created. It is theoretically simply unimaginable for human beings, but maybe there is a profound reason that drives researchers and funders to undertake this grand effort: the need for guidance.

For as long as the humanity existed, people always looked for a leader, able to offer them safety in many areas such as physical, moral and economical security. This has been translated in the political structure of the monarchy and in religious institutions, but has

returned too many disappointments: there are few people able to cover a position of power without taking advantage of it, and the last century has shown us that the leaders that make the most success are often the worst. An impartial, superior intelligence with no concept of personal interest would solve this problem.

Still, those are all conjectures, and all it is known is that this stream of research is incredibile fruitful, and even if we don't know yet if the fruits will be pleasant or not, the promises are so appetizing that the question is not if, but when they will be achieved.

A Appendix

A.1 Environment Model

The code from NEST was modified to make possible for the agent to act on it. To do so, the network's parameters were divided in 3 groups: neuron, synapses and structure parameters.

Synapses parameters

- Mean amplitude and standard deviation of excitatory postsynaptic potential
- Relative inhibitory synaptic strength
- Mean delay of excitatory and inhibitory connections
- Relative standard deviation of the delay of excitatory and inhibitory connections

Neurons parameters

- Membrane potential average and standard deviation for every population and layer
- Reset membrane potential
- Membrane potential after a spike
- Membrane capacitance
- Membrane time constant
- Time constant of postsynaptic excitatory and inhibitory currents
- Time constant of external postsynaptic excitatory current
- Refractory period after a spike

Structure parameters

The connections probabilities between the layers and the populations are defined in an 8x8 matrix, and the agent could modify every value, resulting in an action vector of 64 values. The total number of neurons wasn't changed.

A.2 DDPG model

The DDPG model is programmed with PyTorch (Paszke et al., 2019) following the suggestions of the original paper (Lillicrap et al., 2015), so the neural networks are consisting of input size equals to 1, because there was no need to feed any state to the model, so the input was equal to one. The critic model had a secondary input, the action values, in the first hidden layer. Both models had 2 hidden layer including respectively 300 and 200 nodes, with a Rectified Linear Unit as activation function. The output layer of the critic network was linear and of size one, while the output layer of the actor was a Hyperbolic Tangent and had the same size of the action vector.

The learning rates were 10^{-3} and 10^{-4} for respectively the critic and the actor network, and the batch size was 64. The noise was just a vector of the same length of the action vector, with random values drawn from a Gaussian distribution. The noise was subjected to a multiplicative decay factor of 0.9999.

References

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., ... others (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, *39*(1), 3–20.

Angermueller, C., Pärnamaa, T., Parts, L., & Stegle, O. (2016). Deep learning for computational biology. *Molecular systems biology*, *12*(7), 878.

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (2002). The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, *32*(1), 48–77.

Bernander, O., Douglas, R. J., Martin, K., & Koch, C. (1991). Synaptic background activity influences spatiotemporal integration in single pyramidal cells. *Proceedings of the National Academy of Sciences*, *88*(24), 11569–11573.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... others (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of computational neuroscience*, *8*(3), 183–208.

Burkitt, A. N. (2006). A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, *95*(1), 1–19.

Dayan, P., & Abbott, L. F. (2005). *Theoretical neuroscience: Computational and mathematical modeling of neural systems*. The MIT Press.

de Kock, C. P., & Sakmann, B. (2009). Spiking in primary somatosensory cortex during natural whisking in awake head-restrained rats is cell-type specific. *Proceedings of the National Academy of Sciences*, *106*(38), 16446–16450.

Djurfeldt, M., Lundqvist, M., Johansson, C., Rehn, M., Ekeberg, O., & Lansner, A. (2008). Brain-scale simulation of the neocortex on the ibm blue gene/l supercomputer. *IBM Journal of Research and Development*, *52*(1.2), 31-41.

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669.

Koesdwiady, A., Soua, R., & Karray, F. (2016). Improving traffic flow prediction with weather information in connected cars: A deep learning approach. *IEEE Transactions on Vehicular Technology*, *65*(12), 9508–9517.

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008–1014).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lindén, H., Tetzlaff, T., Potjans, T. C., Pettersen, K. H., Grün, S., Diesmann, M., & Einevoll, G. T. (2011). Modeling the spatial reach of the lfp. *Neuron*, *72*(5), 859–872.

Milletari, F., Ahmadi, S.-A., Kroll, C., Plate, A., Rozanski, V., Maiostre, J., ... others (2017). Hough-cnn: deep learning for segmentation of deep brain regions in mri and ultrasound. *Computer Vision and Image Understanding*, *164*, 92–102.

Musk, E. (2019). An integrated brain-machine interface platform with thousands of channels. *bioRxiv*. Retrieved from `https://www.biorxiv.org/content/early/2019/08/02/703801` doi: 10.1101/703801

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. , 8024–8035. Retrieved from `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`

Potjans, T. C., & Diesmann, M. (2014). The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. *Cerebral cortex*, *24*(3), 785–806.

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., ... others (2019). A deep learning framework for neuroscience. *Nature neuroscience*, *22*(11), 1761–1770.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Sallab, A. E., Abdou, M., Perot, E., & Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, *2017*(19), 70–76.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., ... others (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, 1–5.

Shimoura, R. O., Kamiji, N. L., de Oliveira Pena, R. F., Cordeiro, V. L., Ceballos, C. C., Romaro, C., & Roque, A. C. (2018). Reimplementation of the potjans-diesmann cortical microcircuit model: from nest to brian. *bioRxiv*, 248401.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... others (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioral and brain sciences*, *11*(1), 1–23.

Thomson, A. M., & Bannister, A. P. (2003). Interlaminar connections in the neocortex. *Cerebral cortex*, *13*(1), 5–14.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M.,

... others (2019). Alphastar: Mastering the real-time strategy game starcraft ii. *DeepMind blog*, 2.

Wagatsuma, N., Potjans, T. C., Diesmann, M., & Fukai, T. (2011). Layer-dependent attentional processing by top-down signals in a visual cortical microcircuit model. *Frontiers in Computational Neuroscience*, *5*, 31.

Wawrzynski, P. (2015). Control policy with autocorrelated noise in reinforcement learning for robotics. *International Journal of Machine Learning and Computing*, *5*(2), 91.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, *1*(1), 67–82.