Adaptive Control of Epileptic Seizures using Reinforcement Learning

Arthur Guez

Master of Science

School of Computer Science

McGill University

Montreal, Quebec

2010-08-05

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science

DEDICATION

To my family and Andrée-Anne.

ACKNOWLEDGEMENTS

First and foremost, I thank my supervisor, Joelle Pineau, for the invaluable guidance, support, and insightful feedback she has provided me for the last three years, but also for trusting me to work on such a fascinating project. Additionally, I benefited greatly from being part of the Reasoning and Learning lab; every member, either through talks, suggestions, or discussions, has had an influence on my work and has helped me grow into a better scientist. For that reason, and for the fun and sane working atmosphere they contributed to, I am obliged to every member of the Reasoning and Learning lab, past and present, that I had the chance to interact with—this includes, among others, Doina Precup, Prakash Panangaden, Robert Kaplow, Cosmin Paduraru, Robert D. Vincent, Amin Atrash, Keith Bush, Jordan Frank, Mahdi Milani Fard, Stéphane Ross, Robert West, Julien Villemure, Laura Lewis, Yuri Grinberg, Gheorge Comanici, Monica Dinculescu, Kamal Al-Marhoobi, Dorna Kashef, Guillaume Saulnier, Ryan Faulkner, Fabian Kaelin, Shao-wei Png, Phil Bachman, Pablo Samuel Castro, and Julieta Jakubowicz. I am equally thankful for useful discussions with fellow graduate students Mazen Al Borno and Ivan Savov.

Moreover, the help from Dr. Avoli's lab, at the Montreal Neurological Institute, has been essential to the realization of my thesis. In particular, I want to thank Gabriella Panuccio for conducting the electrophysiological experiments presented in this thesis.

I am also grateful for the financial support I received from both the Natural Sciences and Engineering Research Council of Canada and the Fonds Québécois de la Recherche sur la Nature et les Technologies.

ABSTRACT

This thesis presents a new methodology for automatically learning an optimal neurostimulation strategy for the treatment of epilepsy. The technical challenge is to automatically modulate neurostimulation parameters, as a function of the observed field potential recording, so as to minimize the frequency and duration of seizures. The methodology leverages recent techniques from the machine learning literature, in particular the reinforcement learning paradigm, to formalize this optimization problem. We present an algorithm which is able to learn an adaptive neurostimulation strategy directly from labeled training data acquired from animal brain tissues. Our results suggest that this methodology can be used to automatically find a stimulation strategy which effectively reduces the incidence of seizures, while also minimizing the amount of stimulation applied. This work highlights the crucial role that modern machine learning techniques can play in the optimization of treatment strategies for patients with chronic disorders such as epilepsy.

ABRÉGÉ

Cette thèse présente une nouvelle méthodologie pour apprendre, de façon automatique, une stratégie optimale de neurostimulation pour le traitement de l'épilepsie. Le défi technique est de moduler automatiquement les paramètres de stimulation, en fonction de l'enregistrement de potentiel de champ observé, afin de minimiser la fréquence et la durée des crises d'épilepsie. Cette méthodologie fait appel à des techniques récentes développées dans le domaine de l'apprentissage machine, en particulier le paradigme d'apprentissage par renforcement, pour formaliser ce problème d'optimisation. Nous présentons un algorithme qui est capable d'apprendre une stratégie adaptative de neurostimulation, et ce directement à partir de données d'apprentissage, étiquetées, acquises depuis des tissus de cerveaux d'animaux. Nos résultats suggèrent que cette méthodologie peut être utiliser pour trouver, automatiquement, une stratégie de stimulation qui réduit efficacement l'indicence des crises d'épilepsie tout en minimisant le nombre de stimulations appliquées. Ce travail met en évidence le rôle crucial que les techniques modernes d'apprentissage machine peuvent jouer dans l'optimisation de stratégies de traitements pour des patients souffrant de maladies chroniques telle l'épilepsie.

TABLE OF CONTENTS

LIST OF FIGURES

List of Algorithms

Nomenclature

4AP	4-aminopyridine
EC	Entorhinal cortex
Extra Tree	Extremely Randomized Tree
FQI	Fitted Q iteration
FVI	Fitted value iteration
RL	Reinforcement learning

CHAPTER 1
Introduction

Clinicians treating individuals with chronic disorders—e.g., epilepsy, mental illness, HIV infection—often prescribe a series of treatments in order to maximize favorable outcome for the patient. This generally requires modifying the duration, dose or type of treatment over time. Selecting the best sequence of treatments for an individual presents significant challenges, due to the heterogeneity in response to treatment, as well as the potential for relapse or side-effects. Clinicians frequently rely on clinical judgement and instinct, rather than formal evidence-based processes to optimize sequence of treatments.

Reinforcement learning (RL)[1] is a well-known framework for optimizing sequences of actions in an evolving, time-varying system [65]. Recent advances in reinforcement learning techniques make possible its application to real-world domains in which the state space is continuous and high-dimensional. When applied in the context of treatment design, reinforcement learning provides the means to evaluate the long-term effect of a given treatment, and thus optimize *sequences of treatments* for a given objective.

[1] Throughout this thesis, reinforcement learning refers to the computer science framework named as such, rather than the—related—concept of reinforcement in animal learning.

The idea of applying reinforcement learning to optimize treatment strategies is relatively novel both in the medical and machine learning communities. We attribute this in large part to a lack of appropriate sequential data (or alternately a generative model), which is a key requirement for applying reinforcement learning. This situation is rapidly changing: the medical community has a strong interest in designing studies with multiple sequential, randomized trials [48]. In addition, ongoing clinical trials are evaluating the usefulness of treatment strategies that rely on automated prediction methods to trigger treatment [36], and significant attention is being devoted to developing high fidelity *in silico* models of chronic diseases [73].

The growing availability of medical data and the rapid development of machine learning techniques, such as reinforcement learning, suggest a space for investigating the automated learning of sequential treatment strategies from data. In this thesis, we focus our attention on learning a treatment strategy for epilepsy by optimizing neurostimulations.

1.1 Problem statement

Epilepsy is a highly-prevalent and severe neurological disorder, affecting around 1% of the world population [38]. It is characterized by recurrent unprovoked seizures—synchronous abnormal neuronal firings in the brain which can cause involuntary changes in behavior and sensations. Up to 30% of epilepsy patients suffer from refractory epilepsy [39], meaning that anti-epileptic medications are not effective on those patients. A common treatment option for drug-resistant patients is to undergo resective surgery, but this comes at the risk of physical and cognitive impairments.

An emerging treatment alternative for drug-resistant patients is Deep Brain Stimulation (DBS). In DBS, electrical stimulation is applied near the epileptogenic zone, with the goal of modulating the electrical hyperactivity, and thus preventing seizure symptoms and therefore improving the poor quality of life of those drug-resistant patients.

Recent studies evaluated the effectiveness of DBS strategies in drug-resistant human patients [11, 21, 30, 43]. However, conclusive evidence in humans is difficult to achieve. There is large variance in the disease and symptoms between patients, and it can be difficult to choose an adequate stimulation protocol for each patient. Most results to date come from uncontrolled studies, or (controlled) studies including only a small number of patients. There are many parameters to select when applying DBS, including the target area for stimulation, as well as the frequency, intensity and pattern of the electrical signal. These can be specified either through an open-loop paradigm, or as a closed-loop control system. An open-loop strategy uses preset parameters to deliver stimulation, without monitoring electrical cortical activity. Examples of open-loop strategies include fixed-frequency (periodic pacing) stimulation, as well as stimulation strategies based on a fixed random process (e.g., gaussian noise generator or Poisson impulse trains). In a closed-loop strategy, the stimulation parameters are dynamically changed in response to sensor readings of brain activity; this is illustrated in Figure 1–1. This can be achieved by using software to automatically detect an impending seizure and administering a fixed stimulation protocol designed to terminate the seizure [16, 20, 41, 70, 37, 25, 33]. The stimulation parameters can also be dynamically changed in response to sensor readings through

more sophisticated feedback control methods; the purpose of this thesis is to explore these more elaborated closed-loop control mechanisms by using the reinforcement learning framework.

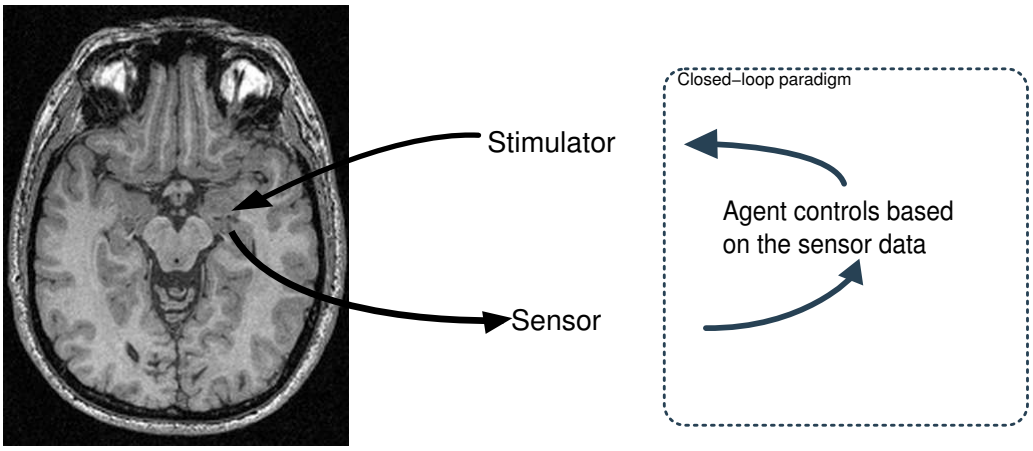

Figure 1–1: *In closed-loop DBS, the stimulation is dynamically adapted based on sensor readings from the brain.*

More substantial evidence of DBS successfully reducing epileptic symptoms has been produced using animal models of epilepsy. A number of studies focused on finding the appropriate stimulation site and frequency of stimulation for open-loop stimulation strategies [21, 61, 17] as well as for closed-loop strategies [60, 49, 20, 61, 14].

Findings using an *in vitro* model have consistently demonstrated that repetitive low-frequency stimulation, delivered in the hippocampus at frequencies similar to those of naturally occurring discharges, provides a reduction of epileptiform synchronization [17]. In particular, the 1.0 Hz frequency was found to have maximal efficacy in depressing ictogenesis in rodent brain slices.

In order to test whether we can find an RL neurostimulation strategy that provides an improvement over open-loop strategies, we explore the avenue of directly optimizing stimulation patterns in an *in vitro* model of epilepsy. Some experimental variables, such as the location of the stimulating and recording electrodes, are more easily controlled when using *in vitro* models than when dealing with living animals. In addition, the mean interval of time between seizures is shorter *in vitro*—by two orders of magnitude. This allows us to concentrate more attention on computational issues than on the management of long and complex biological experimentation, while still maintaining a challenging domain that resembles the human epileptogenicity. Experimentation is also cheaper *in vitro* and, as outlined above, good open-loop neurostimulation strategies already exist for those models, providing a—challenging—comparison baseline. For the reasons mentioned, it should be clear that experimenting in an *in vitro* model of epilepsy is a logical first step towards adaptive seizure control in human patients.

Informally, the problem can be formulated as follows: at every moment in time, given some information about what happened to the signal previously, we need to decide *which stimulation action* we should choose (if any) so as to minimize seizures and, to a lesser extent, stimulations *now and in the future*. The advantage of minimizing the number of stimulations is twofold; first, it can help save the battery life of the embedded stimulation device and, secondly, it relieves some of the stress which is applied to the brain tissue when it is stimulated.

1.2 Computational Approach

Reinforcement learning is one of the leading techniques in computer science and robotics for automatically learning optimal control strategies in dynamical systems. The technique was originally inspired by the trial-and-error learning studied in the psychology of animal learning (thus the term 'learning'). In this setting, good actions by the animal are positively reinforced and poor actions are negatively reinforced (thus the term 'reinforcement').[2] Reinforcement learning was formalized in computer science and operations research by researchers interested in sequential decision-making for artificial intelligence and robotics, where there is a need to estimate the usefulness of taking sequences of actions in evolving, time varying system [35, 65]. It is especially useful in situations in which the agent's environment is stochastic, and for poorly-modeled problem domains in which the optimal control strategy is not obvious.

We frame the problem of learning neurostimulation strategies in the reinforcement learning framework. The goal of the agent is to learn a policy of stimulation that minimizes the duration of seizures while minimizing the number of stimulations. The policy sequentially decides which frequency of stimulation to use for a small amount of time until the next decision point.

[2] It was later discovered that there is a deeper link between reinforcement learning algorithms and the brain mechanisms underlying reinforcement and learning in animals, it is now an active area of research [19].

The recording data available to the reinforcement learning agent is continuous, high-dimensional, and substantially noisy; for that reason we require a robust technique to be able to learn anything from it. In particular, we need to generalize over unseen states using *function approximation*. Though employing function approximation in RL for large continuous problems with unknown dynamics is not a solved issue, some domain knowledge can help but it is not trivial to incorporate in the solution. In Chapter 3, we raise the problem of dealing with sparse data and address it with a domain specific solution to improve the policy of the RL agent.

1.3 Results

We demonstrate the feasibility of using machine learning techniques to learn adaptive treatment strategies by providing evidence that we can learn, from field-potential recordings, a neurostimulation controller that suppresses epileptic seizures.

A first set of evidence is provided *off line* by evaluating our learned reinforcement learning policy, using hold-out datasets, against other stimulation strategies. A second set of evidence is obtained by evaluating our closed-loop strategy *in vitro* using a rodent-brain slice model of epilepsy. We show that our learned controller is able to perform as well as a 1.0 Hz open-loop strategy, while reducing the total amount of stimulation delivered in most slices. Overall, the findings support the view that our computational methodology, such as the use of reinforcement learning, is a viable approach to optimize adaptive treatment strategies.

1.4 Overview

The rest of this thesis is organized as follows. Chapter 2 provides the computational technical background necessary to understand the methodology and experiments in the thesis. Chapter 3 highlights the advantages of approximating across actions in our domain. Then, Chapter 4 describes the experiments that were run and the results, obtained *in silico* and *in vitro*, are presented. Part of the results in Chapter 4 are already published in [58]. Finally, Chapter 5 concludes and discusses the work presented in this thesis before outlining avenues for future work.

CHAPTER 2
COMPUTATIONAL METHODS

This chapter presents the computational technical background pertinent to the work presented in this thesis. In Section 2.1, we introduce *reinforcement learning* and, in Section 2.2, we look at the problem of *function approximation* in reinforcement learning. We then present a particular reinforcement learning method, called *fitted Q iteration*, in Section 2.3. Finally, in Section 2.4, we briefly examine *regression* in order to introduce a particular regression method called *extremely randomized trees*.

2.1 Reinforcement Learning

Reinforcement learning is a problem in which an *agent* learns to make decisions optimally in a given environment by exploring possible actions and receiving scalar rewards for those actions. It is especially useful in situations in which the agent's environment is stochastic, and for poorly-modeled problem domains in which the optimal decision-making policy is not obvious [8, 32, 35, 65, 67].

Formally, the problem is modeled as a Markov decision process (MDP) [32] represented as 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$ consisting of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$ available to the agent, along with a probabilistic transition function T, and a reward function R. Time is modeled as a series of discrete steps with $0 \leq t \leq \infty$. On performing an action $a \in \mathcal{A}$ in state s, the agent receives a scalar reward $r = R(s, a)$ and the environment moves to a new state s' according to the conditional probability distribution $T(s, a, s') = P(s'|s, a)$. The state is assumed to be a sufficient statistic

for the past sensor observations, this is known as the Markov assumption. In an MDP, a *policy* is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that dictates the behavior of an agent (i.e., it defines which action an agent chooses to execute for every state of $\mathcal{S}$). In a reinforcement learning scenario, the transition and reward functions are unknown and the agent learns about the MDP by interacting with the environment; the classical interaction, called *online* RL, is presented in Figure 2–1.

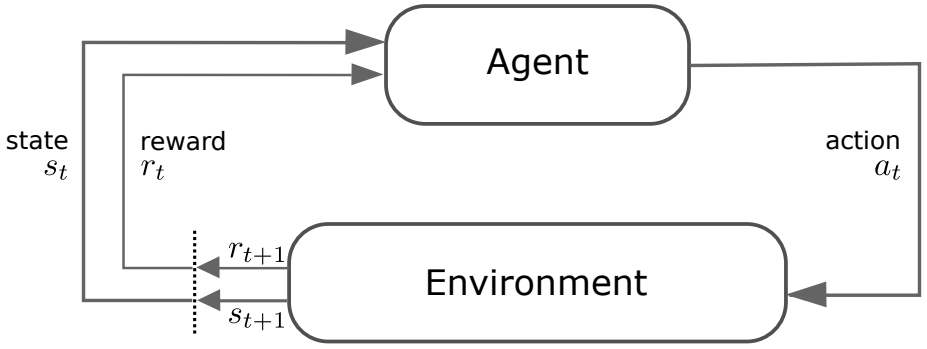

Figure 2–1: *The classical agent-environment interaction in reinforcement learning, adapted from [65].*

The discounted total reward, or *return*, of following a policy π over some infinite time horizon, denoted $\mathcal{R}$ is:

$$\mathcal{R} = \sum_{t=0}^{\infty} \gamma^t R_t. \tag{2.1}$$

Here $\gamma \in [0, 1)$ is a discount factor for future rewards, it implies that rewards far in the future will be worth exponentially less than immediate rewards.[1] The sequence of random variables R_t in Equation 2.1 is a random process that depends on a

[1] It can be thought of as the agent's probability of surviving to the next time step.

starting state s_0 and that defines possible state–action–reward sequences in the MDP according to the transition function T, the reward function R, and the policy π.

Given this formulation, the *value function* $V^\pi : \mathcal{S} \to \mathbb{R}$, designating the value of each state for an agent following a fixed policy π, can be defined as:

$$V^\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s \right] \quad \forall s \in \mathcal{S}. \tag{2.2}$$

In other words, the value V^π of a state s is the *expected* return of following the policy π when starting in state s. By expanding Equation 2.2, the value function can be expressed as a linear fixed-point equation:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s') \tag{2.3}$$

$$= (B^\pi V^\pi)(s) \quad \forall s \in \mathcal{S}, \tag{2.4}$$

where $B^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is the *Bellman operator* for policy π defined as

$$(B^\pi V)(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V(s'). \tag{2.5}$$

We define the *optimal* value V^* for a state s to be:

$$V^*(s) = \max_\pi E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s \right], \tag{2.6}$$

which we can expand to the non-linear fixed-point equation:

$$V^*(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^*(s') \right) \tag{2.7}$$

$$= (B^* V^*)(s) \quad \forall s \in \mathcal{S}, \tag{2.8}$$

where $B^* : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is the *Bellman optimality operator* defined as

$$(B^*V)(s) = \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V(s'). \tag{2.9}$$

Therefore, the optimal value of a state is the maximum[2] expected return that it is possible to achieve in that state. The *optimal policy* π^* can then be constructed from V^* as follows:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V^*(s') \right) \quad \forall s \in \mathcal{S}. \tag{2.10}$$

It is also sometimes useful to express the optimal value of a *state–action* pair, which we formulate as:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s',a'). \tag{2.11}$$

Similarly, the value of a state–action pair, when following a policy π, is defined as the following fixed-point equation:[3]

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)Q^\pi(s',\pi(s')). \tag{2.12}$$

[2] If the action set is not finite, then the maximum operator should be replaced by the *supremum* operator.

[3] Bellman operators for action–value functions can be defined, *mutatis mutandis*, like in Equations 2.5, 2.9.

It is easy to see the relation between the value function and the action–value function:

$$V^{\pi}(s) = \max_{a} Q^{\pi}(s, a), \tag{2.13}$$

$$V^{*}(s) = \max_{a} Q^{*}(s, a). \tag{2.14}$$

Using the action–value function and combining Equations 2.10, 2.11, and 2.14, the optimal policy can now be simply expressed as:

$$\pi^{*}(s) = \operatorname*{argmax}_{a} Q^{*}(s, a). \tag{2.15}$$

In a policy evaluation (or value prediction) scenario, the goal of the task is to evaluate a given policy π by finding V^{π} (or Q^{π}). This is useful to compare the performance of different agents or to predict the result of applying a policy π. In an optimal control scenario, the agent's goal is to find the optimal policy π^{*}. The latter scenario is more pertinent to the work in this thesis; however, both are tightly related. As a matter of fact, some methods employ algorithms designed to solve the first scenario as a component to finding the optimal policy; this is known as *policy iteration*. Therefore, we first review some basic algorithms that carry out policy evaluation in Section 2.1.1. Then, in Section 2.1.2, we discuss algorithms that achieve optimal control. Note that all algorithms presented in Sections 2.1.1 and 2.1.2 are tabular algorithms that assume a finite—and generally small in the sense that it can be compactly enumerated—state space; Section 2.2 discusses the application of reinforcement learning for infinite—or simply large—state spaces.

2.1.1 Value Prediction

Let us first consider the simple case in which we know the MDP model a priori, in other words $\mathcal{S}$, $\mathcal{A}$, T, and R are known. Then, we can solve the fixed-point Equation 2.12 by iterating $\hat{Q}_N$ over N:

$$\hat{Q}^{\pi}_N(s,a) \leftarrow B^{\pi}(\hat{Q}^{\pi}_{N-1}) \tag{2.16}$$

$$= R(s,a) + \gamma E\left[\hat{Q}^{\pi}_{N-1}(s', \pi(s'))\right] \tag{2.17}$$

$$= R(s,a) + \gamma \sum_{s'} P(s'|s,a)\hat{Q}^{\pi}_{N-1}(s', \pi(s')), \quad \forall N > 0 \tag{2.18}$$

with $\hat{Q}^{\pi}_0 \equiv 0$. This is an instance of a dynamic programming solution and is known as the value iteration algorithm [8]. B^{π} is a contraction mapping in the infinity norm, for $\gamma < 1$, since $\|B^{\pi}Q - B^{\pi}Q'\|_{\infty} \leq \gamma\|Q - Q'\|_{\infty}$ *for any* Q and Q'; therefore, by the Banach fixed-point theorem [6], the sequence converges to a unique fixed point—Q^{π}. V^{π} can also be obtained in a similar fashion.

When the model is not available, there are two main approaches to estimate V^{π}. The agent can first estimate T and R from samples in order to get approximations $\hat{T}$ and $\hat{R}$,[4] and then compute the (action-)value function using the value iteration algorithm described above with a model composed of $\hat{T}$ and $\hat{R}$. If $\hat{T}$ and $\hat{R}$ are sufficiently accurate, then the value function obtained will be close to the real V^{π} [10]. This is called a *model-based* approach.[5] An agent can also estimate V^{π} directly

[4] This can be done, for example, using maximum likelihood estimation.

[5] There exists several model-based approaches, the one outlined is rather elementary.

with so-called *model-free* approaches. A simple model-free technique is to gather return estimates $\mathcal{R}_s$ from a state s when following policy π to build a Monte-Carlo (MC) estimate of $V^\pi(s)$:

$$\hat{V}^\pi(s) \leftarrow \hat{V}^\pi(s) + \alpha_t \left[\mathcal{R}_s - \hat{V}^\pi(s)\right], \tag{2.19}$$

where α_t is the learning rate. A shortcoming of that method is that it requires a full return estimate in order to update $\hat{V}^\pi(s)$, we can alleviate that problem with the so-called *temporal-difference* (TD) learning method [63]; TD is a central concept in reinforcement learning. In TD learning, the current estimate of the (action-)value function is used as part of the learning target to avoid the need for a full return sample:[6]

$$\hat{V}^\pi(s_t) \leftarrow \hat{V}^\pi(s_t) + \alpha_t \left[r_t + \gamma \hat{V}^\pi(s_{t+1}) - \hat{V}^\pi(s_t)\right]. \tag{2.20}$$

Equation 2.20 is the essence of the TD(0) algorithm. The MC method and the TD(0) algorithm can be viewed as the two extrema of a parametrized algorithm TD(λ), where $\lambda = 0$ leads to TD(0) and $\lambda = 1$ leads to a MC method. Note that TD(λ) is known to converge for all λ [18].

2.1.2 Control

In the optimal control scenario, if the model is known, we can also rely on dynamic programming by using a value iteration algorithm, as in Equation 2.18, to learn the optimal action-value function—and therefore the optimal policy. The

[6] For that reason TD algorithms are said to *bootstrap*.

following update equation is needed in place of Equation 2.18 (i.e., we utilize the B^* operator instead of B^π):

$$\hat{Q}^*_N(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a \in \mathcal{A}} \hat{Q}^*_{N-1}(s',a) \;\; \forall s, a \;\; \forall N > 0. \tag{2.21}$$

Again, as in the policy evaluation scenario, model-based approaches can be used to first estimate the MDP model and then derive the (near-)optimal policy from the approximated model. With respect to model-free approaches, the temporal-difference ideas employed in the policy evaluation case can also be leveraged to learn the optimal policy. The Q-learning algorithm [75] is an *online* algorithm to learn the optimal action-value function that relies on a TD update, its main update equation is:

$$\hat{Q}^*(s_t,a_t) \leftarrow \hat{Q}^*(s_t,a_t) + \alpha_t \left[r_t + \gamma \max_{a \in \mathcal{A}} \hat{Q}^*(s_{t+1},a) - \hat{Q}^*(s_t,a_t \right]. \tag{2.22}$$

This algorithm is said to be online because there is an interaction with the environment to decide which action to execute at each step. In order to converge to the optimal action-value function, the agent must not act greedily; that is to say, the agent must occasionally select actions which are not optimal according to its current estimate in order to explore potentially better actions. A simple non-greedy policy, called ϵ-greedy policy, chooses to execute a random action with probability ϵ and acts greedily otherwise. Since the Q-learning algorithm will converge to the optimal action-value function,[7] as opposed to the action-value function of the policy it is

[7] If α_t decreases in an appropriate manner.

executing (i.e., the ϵ-greedy policy or some other exploration policy), we say it is an *off-policy* algorithm. The early theoretical analysis of Q-learning is presented in [71].

Note that the subject of efficient exploration in RL is vast and is still the subject of active investigation. Researchers seek algorithms that achieve faster convergence, or good enough performance with a finite set of samples, instead of simply guaranteeing asymptotic convergence. The case in which we cannot interact with the system is covered in Section 2.3.

2.2 Function Approximation

Standard tabular reinforcement learning algorithms assume a finite number of states—and actions—, yet most real-world problems possess some continuous components that do not have straightforward discretizations. For those problems, the continuity implies an infinite number of states and, therefore, some form of generalization over unseen states is required to be able to learn a policy for all states. If V^π is the true value function of a policy π, then the *function approximation* problem is to find a representable and computable function $V^{\pi'}$ from a defined class of functions, such as to match V^π closely.

A popular method is to represent the value function $V^{\pi'}$ as a smooth differentiable function of a parameter vector θ and to learn the parameters using gradient-descent [63]. In a special case of parametrization, called *linear function approximation*, a vector of features ϕ_s represents a state s such that the value function can be represented as a linear combination of those features: $V^{\pi'}(s) = \theta^\top \phi = \sum_i \theta(i)\phi_s(i)$.

Since being able to approximate the value function is a crucial requirement to solving most real-world problems, we briefly review methods for approximating the value function in the value prediction and control scenarios.

2.2.1 Value Prediction

In a value prediction scenario, the problem is to find the value V^π of a fixed policy π by interacting with the environment. The update for TD(0) when used with linear function approximation is:

$$\theta \leftarrow \theta + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))\nabla_\theta V(s_t) \tag{2.23}$$

$$= \theta + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))\phi_{s_t}. \tag{2.24}$$

Gradient-descent methods for TD(λ) learning with linear function approximation were initially proposed in [63] along with a proof of convergence for the cases TD(0) and TD(1). The algorithm was used successfully to learn an expert backgammon player in [68]. A proof of convergence of linear TD(λ), for general λ, was later published in [72]. However, those results were only valid for on-policy learning (i.e., the states are sampled according to the policy π to be evaluated) which is not applicable if the samples are generated through the execution of a different behavior policy. In [59], an off-policy linear TD(λ) learning algorithm that relies on importance sampling was demonstrated to be convergent, but at the expense of high variance. More recently, a fast convergent off-policy TD(0) algorithm with linear function approximation, called GTD for gradient temporal-difference, was introduced to remove those limitations [66]. This work was later generalized to non-linear function approximation [42].

2.2.2 Control

Function approximation has been employed early on in order to seek (near-) optimal policies of complex problems [64]. Nevertheless, researchers soon realized that using function approximation was not necessarily sound and could lead to divergence even on some simple problems [12].

If the transition model is known, then a classic solution is to run the *fitted value iteration* (FVI) algorithm which iterates the value function as follows: $V_{k+1} = \Pi_{\mathcal{F}} B V_k$, where B is the Bellman operator and $\Pi_{\mathcal{F}}$ is a projection on the space of value functions representable by the class of functions $\mathcal{F}$ employed for the function approximation. FVI is known to be convergent if $\Pi_{\mathcal{F}}$ is a non-expansion, this is satisfied in particular for a class of approximation functions called *averagers* or *kernel-based methods* [27]. Examples of *averagers* are k-nearest-neighbor, state aggregation, tree-based methods, and local weighted averaging.

If the model is unknown, then an approximate version of fitted value iteration, described in more details in Section 2.3, relies on sample trajectories from the MDP to learn a policy and is known to be converge to the optimal value function *asymptotically* under some continuity assumption and for particular classes of approximation function [53]. Finite sample bounds for those methods were obtained in [47]. Other methods are used in practice, such as variants of Q-learning with function approximation, although they do not always provide theoretical guarantees. If state aggregation is used with Q-learning, then convergence to an approximation of Q^*—optimal in the induced MDP—can be obtained [9].

2.3 Fitted Q Iteration

Many traditional reinforcement learning approaches use *on-line* learning, in which the agent interacts with the environment dynamically and updates its policy after taking each action [65]. However, in many practical domains (including instances of medical treatment design), it is not possible to train an agent entirely on-line. Normally, data will be collected in a fixed series of experimental trials and the potentially disruptive effects of an untrained agent may impose an unacceptable risk (e.g., if the policy is applied to a patient as part of a medical treatment).

In cases such as this, it is preferable to take a *batch* mode reinforcement learning approach, in which the agent is trained using a series of previously recorded trajectories containing state, action, and reward information.

The fitted Q iteration (FQI) algorithm [22], which builds on earlier work on fitted value iteration [28, 53], takes as input a set $\mathcal{F}$ of 4-tuples of the form (s_t, a_t, r_t, s_{t+1}), where each tuple is a sample of a one-step transition dynamics of the system. Unlike earlier formulations of batch-mode RL, the FQI algorithm is well suited for problems with continuous state and action spaces. Also, the algorithm has been shown to make efficient use of training data [34], which is especially important in medical applications, where data may be sparse and expensive to collect.

When using FQI, it is assumed that we do not know the transition dynamics or reward function of the MDP. We can still approximate the value iteration algorithm as follows; at each iteration k of the algorithm, we form an estimate $\hat{Q}_k$ of the true Q_k function by iteratively learning the mapping from the sample transitions:

$$\hat{Q}_k(s_t, a_t) = r_t + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_{k-1}(s_{t+1}, a'). \tag{2.25}$$

By using this formulation, the reinforcement learning problem can be cast as a batch supervised learning problem. Thus any regression algorithm can be used to learn the mapping $Q : S \times A \to \mathbb{R}$ at each iteration. The pseudocode for the FQI algorithm is listed in Algorithm 1.

Algorithm 1: Fitted Q iteration algorithm, adapted from [22].

Input: $\mathcal{F} = \left\{\left(s_t^l, a_t^l, r_t^l, s_{t+1}^l\right), l = 1, \ldots, n\right\}$ and a regression algorithm
$r : \{(i, o) | (i, o) \in \mathbb{R}^k \times \mathbb{R}\} \to f : \mathbb{R}^k \to \mathbb{R}$

Output: $\hat{Q}$

$N \leftarrow 0$
$\hat{Q}_N(s, a) \leftarrow 0 \ \ \forall (s, a) \in \mathcal{S} \times \mathcal{A}$
while *Stopping condition is not reached* **do**
    $N \leftarrow N + 1$
    $\mathcal{TS} \leftarrow \left\{\left(i^l, o^l\right), l = 1, \ldots, |\mathcal{F}|\right\}$ where
        $i^l = (s_t^l, a_t^l)$,
        $o^l = r_t^l + \gamma \max_{a \in \mathcal{A}} \hat{Q}_{N-1}(s_{t+1}^l, a)$.
    $\hat{Q}_N \leftarrow r(\mathcal{TS})$
end
$\hat{Q} \leftarrow \hat{Q}_N$

If the transitions are stochastic, then most regression methods ensure that we are implicitly computing an expectation for $\hat{Q}_N(s, a)$, instead of fitting it to a particular realization of the transition kernel.

The stopping condition in Algorithm 1 typically checks if the difference between two consecutive $\hat{Q}_N$ is less than some threshold: $\|\hat{Q}_N - \hat{Q}_{N-1}\|_\infty < \epsilon$. However, the FQI algorithm is only guaranteed to converge under some conditions, for example in

the case of kernel-based methods if the kernels are kept constant across the iterations [22]. Therefore, it is sometimes more appropriate to set a fixed number of iterations, relying on the rate of convergence of the value iteration algorithm as an indicator for the required number of iterations.

In terms of the theoretical quality of the solution obtained by the algorithm, some finite-sample performance bounds for FQI were proven in [2] and [47] for a particular class of regressors.

2.4 Regression

The FQI algorithm requires an appropriate supervised regression algorithm to learn the $\hat{Q}_N$ functions. In this section, we first briefly review the bias/variance decomposition for supervised learning and then introduce a particular tree-based regression algorithm called Extremely Randomized Trees.

2.4.1 Bias/Variance Decomposition

In supervised learning, a statistical model describing the data generation process that we want to model is generally assumed:

$$O = f(I) + \epsilon, \tag{2.26}$$

where O is the output random variable that depends on I, the input variable, and the random variable ϵ models noise or variation with $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma_\epsilon^2$. The random variable I is distributed according to some distribution $\mathcal{D}_I$. The goal of the regression problem is to learn the relation $f(i) = E(O|I = i) \ \ \forall i$ given a set of pairs $\mathcal{TS} = \{(i_l, o_l)|l = 1, \ldots, n\}$ generated according to the (unknown) model. The learned estimate of f is denoted by $\hat{f}$. We can decompose the expected prediction

error of the regression algorithm's estimate for a particular query point i_0 as follows:

$$\begin{aligned}\mathrm{Err}(i_0) &= E[(O - \hat{f}(i_0))^2 | I = i_0] \\ &= \sigma_\epsilon^2 + (E[\hat{f}(i_0)] - f(i_0))^2 + E[(\hat{f}(i_0) - E[\hat{f}(i_0)])^2] \\ &= \sigma_\epsilon^2 + \mathrm{Bias}^2(\hat{f}(i_0)) + \mathrm{Var}(\hat{f}(i_0)). \end{aligned} \tag{2.27}$$

The first term in Equation 2.27 is the irreducible error of this statistical model, it is a lower bound on what is achievable by any learning algorithm. Supervised learning algorithms strive to reduce the two other terms, the bias and variance terms. The squared bias term indicates how far is the average estimate to the true average output, and the variance term expresses how the estimate varies around its mean depending on the learning samples and, potentially, the algorithm's randomness. Note that this is a local estimate of the error which can be globalized by taking into account the distribution of samples $\mathcal{D}_I$.

One possible strategy to reduce the bias and variance terms involves utilizing a combination of randomization and averaging [31]. We review one particular method that performs this, called Extremely Randomized Trees, in the following section.

2.4.2 Extremely Randomized Trees

The Extremely Randomized (Extra) trees [26] is a regressor with attractive properties that has already been employed in the context of a medical application [23].

Unlike classical regression tree algorithms such as CART or Kd-trees, the Extra Tree algorithm builds an ensemble of trees, and the overall value returned by the final classifier is the mean of the values of the individual trees. The Extra Trees

algorithm is inspired by the celebrated Random Forest algorithm [13], the main differences between the two algorithms are that the Extra Trees method does not bootstrap samples from the training set—instead it relies on all the samples for building every tree—and more randomness is introduced in the tree construction process. The first difference is intended to reduce bias and the second to reduce variance—after averaging of the trees in the forest. In a vast empirical study, Extra trees were faster to compute and generally obtained similar or better performance than Random Forests [26].

The Extra Tree algorithm has three parameters: M, the number of trees to create; K, the number of candidate tests at each node; and n_{min}, the minimal number of nodes at each leaf. The algorithm builds each of M trees using the entire training set $\mathcal{TS}$. Each node is constructed by creating K candidate tests consisting of a randomly selected element of the input vector and a random cut point. A score is calculated for each candidate test based on the relative variance reduction of each test. The best test is kept and all others discarded. The process continues until each leaf node contains no more than n_{min} elements. The output of the M trees are averaged to produce the output of the ensemble. The pseudocode for building one of the M trees is outlined in Algorithm 2 and 3. It should now be clear how the Extra Trees combine the randomization and the averaging. Let us understand the effect of such a strategy on the bias and variance terms of Equation 2.27. First, if we assume a single tree, the variance term can be decomposed, using the law of total variance, as follows:

$$\mathrm{Var}_{\mathcal{TS},\mathcal{TB}}\hat{f}(i_0) = \mathrm{Var}_{\mathcal{TS}}\left[E_{\mathcal{TB}|\mathcal{TS}}\left[\hat{f}(i_0)\right]\right] + E_{\mathcal{TS}}\left[\mathrm{Var}_{\mathcal{TB}|\mathcal{TS}}\left[\hat{f}(i_0)\right]\right], \tag{2.28}$$

where $\mathcal{TS}$ is the training set and $\mathcal{TB}$ is the random variable governing an entire tree-building process.[8] The first term in Equation 2.28 expresses how the average—with respect to the tree-building process—prediction varies with different training samples; the second term represents the expectation, over all training sets, of the variance in the prediction with respect to $\mathcal{TB}$. When building M independent trees, we have:

$$\mathrm{Var}_{\mathcal{TS},\mathcal{TB}^M}\hat{f}(i_0) = \mathrm{Var}_{\mathcal{TS}}\left[E_{\mathcal{TB}|\mathcal{TS}}\left[\hat{f}(i_0)\right]\right] + \frac{E_{\mathcal{TS}}\left[\mathrm{Var}_{\mathcal{TB}|\mathcal{TS}}\left[\hat{f}(i_0)\right]\right]}{M}. \tag{2.29}$$

Therefore, the second term in the prediction variance can be arbitrarily controlled as we increase the number of trees. By introducing randomization, both the bias and variance in the expected prediction error generally increase. This phenomenon is illustrated in Figure 2–2. However, the averaging of the randomized estimates generally reduces the overall expected prediction error.

The exact outcome of such a strategy depends on the particular algorithm used but also on its parameters. In the Extra Trees case, the parameter K determines the amount of randomness introduced. If K is small, then the tree-building process mostly ignores the training set, which reduces the variance due to the distribution of training sets but induces a significant bias.

Any randomized regression tree algorithm could be an appropriate choice of supervised regression algorithm to utilize with the FQI algorithm, given both their

[8] For clarity, it is made implicit in the notation of Equation 2.28 that $\hat{f}$ depends on both $\mathcal{TS}$ and $\mathcal{TB}$.

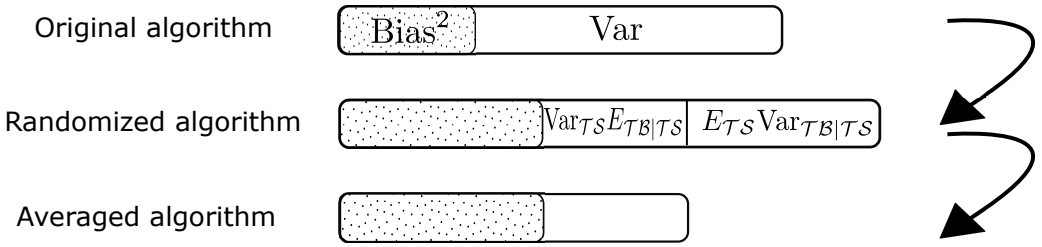

Figure 2–2: *Evolution of the bias and variance terms in Equation 2.27 after randomization and averaging. Adapted from [26].*

efficiency and their excellent performance in the presence of noisy or irrelevant variables. In empirical experiments with several reinforcement learning domains, the Extra Trees algorithm has exhibited excellent performance in terms of both computational efficiency and empirical return relative to other regression tree algorithms [22]; therefore, we select this method for experiments conducted in this thesis.

2.5 Conclusion

This technical background chapter covered sequential decision making in MDPs using RL techniques, function approximation in RL, and a particular batch RL algorithm called Fitted Q iteration. Some basic regression concepts were introduced along with a tree-based regression algorithm called Extremely Randomized Trees. Since the focus of this chapter is on computational methods, we omitted reviews of fundamental neuroscience and electrophysiology concepts.

Algorithm 2: `BuildExtraTree`, adapted from [26].

Input: $\mathcal{TS} = \{(i^l, o^l), l = 1, \ldots, n\}$
Output: tree T

if
  $|\mathcal{TS}| < n_{min}$, *or*
  all input variables are constant in $\mathcal{TS}$, *or*
  all output variables are constant in $\mathcal{TS}$
then
  Return leaf with value the average of the output values in $\mathcal{TS}$: $\frac{1}{|\mathcal{TS}|}\sum_l o^l$.
else
  Let $[i_j < t_j] = \texttt{FindTest}(\mathcal{TS})$
  Split $\mathcal{TS}$ into $\mathcal{TS}_{left}$ and $\mathcal{TS}_{right}$ according to the test $[i_j < t]$.
  Build $T_{left} = \texttt{BuildExtraTree}\ (\mathcal{TS}_{left})$ and
  $T_{right} = \texttt{BuildExtraTree}(\mathcal{TS}_{right})$ from these subsets.
  Create a node T with the test $[i_j < t_j]$, let T_{left} and T_{right} be the left and right subtrees of this node.
  Return T.
end

Algorithm 3: `FindTest`, adapted from [26].

Input: $\mathcal{TS} = \{(i^l, o^l), l = 1, \ldots, n\}$
Output: test $[i_j < t_j]$

Select K input indices $\{i_1, i_2, \ldots, i_K\}$ at random, without replacement, among all input indices that are non-constant across $\mathcal{TS}$.
for $k = 1$ *to* K **do**
  Compute maximal value of i_k in $\mathcal{TS}$: $i^{\mathcal{TS}}_{k,max}$.
  Compute minimal value of i_k in $\mathcal{TS}$: $i^{\mathcal{TS}}_{k,min}$.
  Draw discretization threshold t_k *uniformly* in $]i^{\mathcal{TS}}_{k,min}, i^{\mathcal{TS}}_{k,max}[$.
  Compute score $S_k = \frac{Var(o|\mathcal{TS}) - \frac{|\mathcal{TS}_{left}|}{|\mathcal{TS}|}Var(o|\mathcal{TS}_{left}) - \frac{|\mathcal{TS}_{right}|}{|\mathcal{TS}|}Var(o|\mathcal{TS}_{right})}{Var(o|\mathcal{TS})}$.
end
Return test $[i_j < t_j]$ with highest score S_j.

CHAPTER 3
APPROXIMATION ACROSS ACTIONS

In reinforcement learning, researchers usually study the problem of generalizing over the input states to learn the action-value function. A less-studied problem is the generalization over actions which arises whenever there is a limited amount of training data available to an RL agent; when estimating Q values, it is possible to employ either a function approximator for each action or only one function approximator whose input encompass states and actions. The first option is not practical for large—or continuous—action spaces, but it can be applied in presence of relatively small action sets.

When sufficient data is available about the environment dynamics, the two variants should be able to perform equally well. However, it is frequently the case that the training data for a real-world problem does not contain all of the environment dynamics. This is especially true in batch reinforcement learning where the data is frequently acquired under some behavior policy which is different than the learned policy; therefore, the distribution of state-action pairs can be different in the training set, compared to what is observed during the execution of the learned policy. In particular, entire sections of the state-action space might be missing, or poorly represented, in the training set. If that is the case, the outcome of the two variants might differ considerably. In this chapter, we examine empirically, in a batch RL setting, how incomplete datasets affect the two strategies across a set of contrasting

domains. The conclusions drawn from the obtained results form the basis of some design decisions in our adaptive neurostimulation controller.

3.1 Technical Development

In this section and throughout the rest of this chapter, we adopt the notation and concepts introduced in Chapter 2. The main objective of this chapter is to understand, in the context of batch reinforcement learning, the difference between learning the state-action value function ($Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$) by estimating $Q(\cdot, a)$ for each $a \in \mathcal{A}$ *separately*, and learning Q by estimating $Q(\cdot, \cdot)$ directly—treating $\mathcal{S} \times \mathcal{A}$ as the input space.

For example, when using the FQI algorithm (Algorithm 1), we need to apply a supervised regression algorithm to learn the mapping $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. When $\mathcal{A}$ is a finite set, the regression problem can be constructed as outlined in Algorithm 1:

$$\mathcal{TS} \leftarrow \{\left(i^l, o^l\right), l = 1, \ldots, |\mathcal{F}|\}, \tag{3.1}$$

with $i^l = (s_t^l, a_t^l)$ and $o^l = r_t^l + \gamma \max_{a \in \mathcal{A}} \hat{Q}_{N-1}(s_{t+1}^l, a)$, or it can also be split in $|\mathcal{A}|$ regression problems as follows:

$$\mathcal{TS}_a \leftarrow \{\left(s_t^l, o_t^l\right) \mid a_t^l = a,\ l = 1, \ldots, n\}, \tag{3.2}$$

with $o^l = r_t^l + \gamma \max_{a \in \mathcal{A}} \hat{Q}_{N-1}(s_{t+1}^l, a)$. In the first strategy, the action and state components are assembled into the input vector, allowing the possibility of approximating or generalizing the $\hat{Q}_N$ estimate over actions. In the second strategy, as outlined in Equation 3.2, the tuples are first partitioned based on their actions, then the input vector for each regression partition consists of the state components.

If the FQI algorithm is utilized in conjunction with the Extra Trees, the first regression strategy, which we will call *Strees* for notation convenience, maintains one forest for all actions. In each tree of that forest, the tests can split on the action component in addition to each of the state feature. The other variant, which we will call *Atrees*, maintains a separate forest of trees for each action. Hence, for the *Atrees*, each forest represents the function $Q(., a)$ for a particular action a. One should note that the *Atrees* can be seen as a particular *Strees* construction in which each tree is split on the action component at the root node (using a non-binary split). To extract the policy from the *Atrees* for a state s, every forest is queried with s and, then, the action associated with the forest leading to the maximum value is returned.

3.2 Experiments

We compare the performance of FQI, when dealing with a finite action set, between the two different strategies outlined in the previous section. We conduct experiments in three domains with the Extra trees as a regressor. The first domain, described in Section 3.2.1, is a classical reinforcement learning problem with continuous states. The second domain, described in Section 3.2.2, is a toy problem resembling the epilepsy problem in some high-level way. This toy problem has similar characteristics to the Puddle World domain but its simplicity allows for an analytical interpretation of the results. The third domain, described in Section 3.2.3, is another toy problem with different characteristics from the other two problems. The hypothesis driving this choice of domains is the exhibition of performance differences between the two variants depending on the attributes of each domain (e.g., action set, reward function).

3.2.1 Puddle World

The Puddle World problem is based on [64]; the objective of the task is to reach a goal box in a 2-dimensional continuous unit square that contains an obstacle, the puddle (see Figure 3–1). In this variation of the problem, the puddle has radius 0.1 and the goal region is 0.1×0.1, located at the top right corner. The reward is -1 at each step except if the agent is in the goal region, in which case the reward is 100. A negative reward ranging from 0 to -1000 is given proportional to how close the agent is to the center of the puddle. The agent is free to move in the four cardinal directions within the unit square, the step size being 0.05. Gaussian noise is added to the location of the agent after a step is taken. We make the task non-episodic by teleporting the agent to a random location, uniformly across the two-dimensional state space, from any action within the goal box.

We generate 5 datasets, $\mathcal{F}_1^l$ with $l \in \{1, \ldots, 5\}$, of $40,000$ 4-tuples under a random policy that is enough to learn a close approximation to the optimal policy for the two variants of the FQI algorithm. We then generate tuples for a different kind of training dataset, $\mathcal{F}_2^l$, in a more constrained way. We set the training data to be sparse in a rectangular region around the puddle (see dotted lines in Figure 3–1) by terminating the agent's trajectory and restarting it in a random location with probability ϵ for every step in that region. In a sense, this is a realistic scenario as it is generally easier to acquire data for nice parts of an environment than for parts which present risks. Here we could view the zone around the puddle as hazardous; therefore, less information about would be available in that training data set.

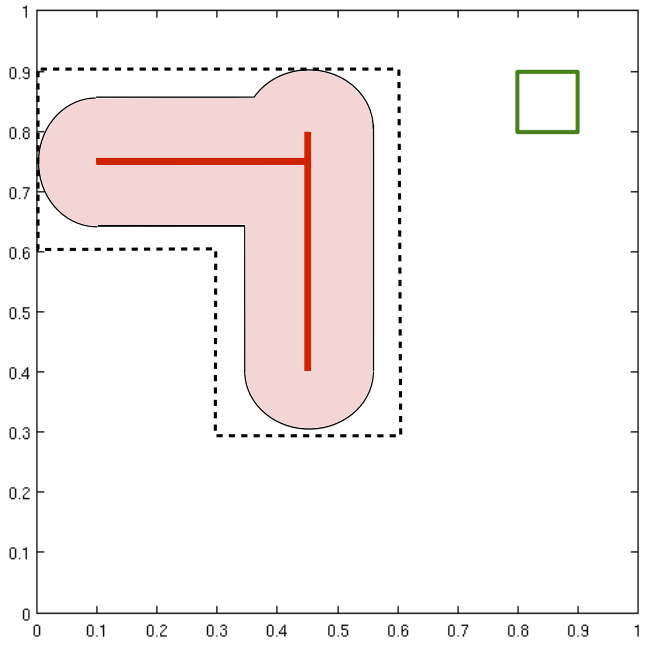

Figure 3–1: *The Puddle World problem. The red line denotes the center of the puddle, and the red-shaded area is the influence zone of the puddle where negative rewards are given. The green square on the top right is the goal zone of this environment where positive rewards are given. The black dotted lines around the puddle zone demarcates the zone where the probability of ending the agent's trajectory is* ϵ.

We feed the sets of tuples, $\mathcal{F}_1^l$ and $\mathcal{F}_2^l$ with $l \in \{1, \ldots, 5\}$, to the *Atrees* and *Strees* in order to obtain estimates $\hat{Q}_1^l$ and $\hat{Q}_2^l$. We then test the control policies derived from $\hat{Q}_1^l$ and $\hat{Q}_2 l$ in the Puddle World environment. This procedure is repeated 10 times for each of the 5 series of datasets before averaging. A standard set of parameters was used for both variants: $M = 80$, $K = 2$, $n_{min} = 5$ with 130 iterations in the FQI algorithm (the structure of the trees is fixed after 50 iterations to ensure convergence).

3.2.2 Toy Epilepsy Problem

This is a simple problem consisting of 6 states and 2 actions that is specially designed to illustrate how a difference can arise between the two variants in the neurostimulation optimization problem. The MDP underlying the toy problem is rendered on Figure 3–2, it is composed of $\mathcal{S} = \{1, \ldots, 5\}$, $\mathcal{A} = \{0, 1\}$. From any state s, both actions take the system to state $s + 1$, except in state 3 where action 1 leads to state 1 and in state 6 where both actions lead to state 1. A reward of -1 is given for being in state 5 or 6 (0 otherwise), in addition action 1 is always associated with a negative reward of -0.01.

The crude parallel between this toy problem and the neurostimulation is realized once we map action 1 in the toy problem to an electrical stimulation event, and states $4 - 6$ to seizure states. Similar to what goes on in the brain, the stimulation events keep the system from going into seizures, but once a seizure is initiated then stimulations do not influence heavily the seizure propagation.

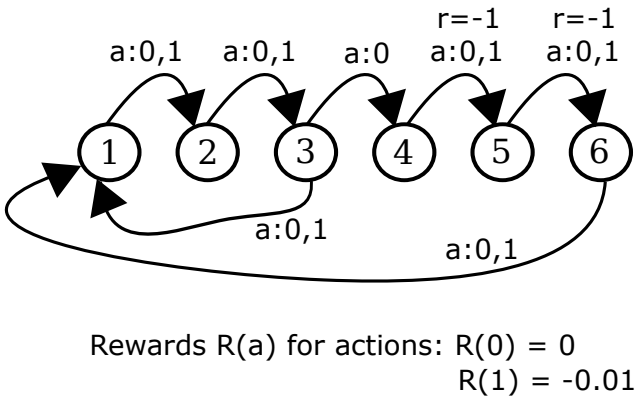

Figure 3–2: *The toy epilepsy problem, this MDP is composed of* 6 *states and* 2 *actions. Action* 1 *always carries a small negative reward and states* $5 - 6$ *incur a greater negative reward. Action* 1 *should be interpreted as a stimulation and states* 5 *and* 6 *as seizure states.*

As can be seen in Figure 3–2, the optimal policy would be to select action 0 at all times except in state 3, in which case action 1 should be selected. We build two kinds of data sets from this environment, one, acting as control, is simply a small data set acquired by following a random policy that selects actions uniformly at random. The other one, meant to show the difference between the two variants, is acquired by running action 0 *only*, until we obtain 100 tuples, and then running action 1 *only*, until we get an additional 100 tuples. It should be noted that the last data set does not contain tuples with state 4, 5 or 6 under action 1. Additionally, we either generate these two data sets with perfect information on the states, or by introducing noise in the observations as well as extra noisy dimensions. We add noise in the observations by simply adding a uniform random number between -1 and 1, as well as adding a small negative random number when action 1 is selected, such that the action has an effect on the observation.

In the noise condition, since the state information is not complete for the observer, an agent that only acts based on the current observation cannot attain the optimal control policy. However, it can still do fairly well by being cautious not to run into states 5 and 6 which carry the large negative reward.

We employ those sets of tuples, generated according to the description above, to train the $\hat{Q}$ function of an agent using the two variants of the FQI algorithm, *Atrees* and *Strees*. We then test the control policies obtained by running them in the environment.

3.2.3 Toy Problem 'Easy choice'

Finally, the third MDP considered for experimentation is composed of 10 states and 11 actions. From each state i, the action a_j for $j \leq 10$, with probability 0.5, leads to state j acquiring a reward of j; with probability $0.5/i$, the action a_j carries the system to state k, for $k \leq i$, incurring a reward of 0. From any state, action a_{11} brings the system to state 1 with a reward of -1. This MDP is illustrated in Figure 3–3. It should be noted that the optimal policy is simply to execute action a_{10} *regardless* of the current state. We choose this MDP because the effects of all actions are mostly similar across states and, therefore, we can speculate that not approximating across actions could be a superior strategy when training data is missing.

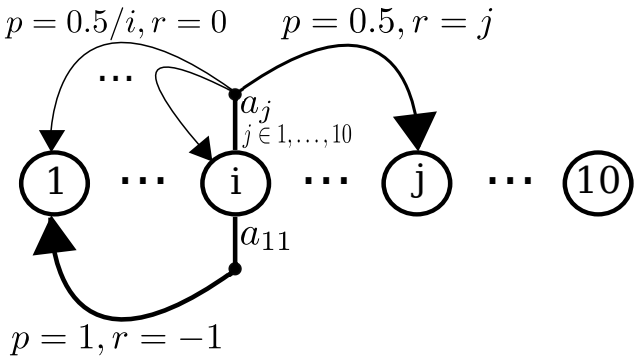

Figure 3–3: *This 'Easy choice' toy MDP is composed of* 10 *states and* 11 *actions. In state* i*, the action* a_j *(*$j \leq 10$*) leads to state* j *and a reward* j *with probability* 0.5 *and otherwise leads to one of the lower states (*$s \leq i$*) uniformly with no reward. Action* a_{11} *always resets the system to state* 1 *with a reward of* -1*.*

We generate two kinds of datasets from the environment. The first, $\mathcal{F}_1^l$, is obtained by sampling states and actions uniformly. The second, $\mathcal{F}_2^l$, is obtained by biasing the selection of action for $s \leq 5$: action a_i is selected with probability $\frac{\delta}{6} + \frac{1-\delta}{11}$ if $i \leq 5$ or if $i = 11$, and with probability $\frac{1-\delta}{11}$ if $i > 5$. In other words,

the bias is controlled by the δ parameter; as δ increases towards 1, actions smaller than or equal to 5 and action a_{11} become more likely to be sampled (when the state is smaller or equal than 5). Therefore, if δ is big, tuples that start in a state lower than 5 and take actions bigger than 5 are rare in the training dataset. Collections of 200 datasets of $\mathcal{F}_1$ or $\mathcal{F}_2$ for various δ, all composed of 2000 tuples, are generated and the estimates $\hat{Q}_1$ and $\hat{Q}_2$ obtained, respectively, by the *Atrees* and *Strees*, are computed each time. The parameters employed for training are identical to those used in the Puddle World environment. After training, the policies induced by the Q-value estimates are subjected to testing in the MDP.

3.3 Results

3.3.1 Puddle World

When we progressively starve the learning agent from environment dynamics in the puddle zone by increasing ϵ, the performance of the agent decreases but the two variants react differently as ϵ goes to 1. We observe a rough trend that indicates that the *Strees* method might be more robust to the loss of information about the environment dynamics around the puddle zone. Figure 3–4 presents the case $\epsilon = 0$ and $\epsilon = 0.9999$. When the training data is gathered without constraints, the performance of the agents trained using the two variants is nearly identical. When the constraint is introduced, the *Strees* perform better on average and with less variance than the *Atrees*.

Figure 3–5 illustrates the similarity of the output of the two variants when training from $\mathcal{F}_1$ ($\epsilon = 0$) by displaying the policy and Q-value function (for action *UP*) over the two-dimensional state space. Contrastingly, when using $\mathcal{F}_2$ ($\epsilon = 0.9999$), the

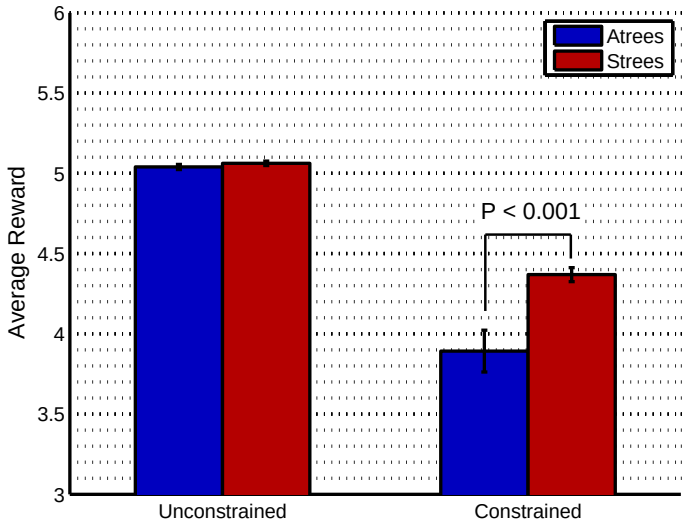

Figure 3–4: *Average reward in the puddle world for the two strategies in two scenarios:* $\epsilon = 0$ *(*Unconstrained*) and* $\epsilon = 0.9999$ *(*Constrained*). The results are averaged over the* 5 *datasets and the* 10 *runs per dataset. The error bar represents the standard error of the mean. When* $\epsilon = 0.9999$*, the performance difference between the two variants is statistically significant with* $P < 0.001$*.*

difference between generalizing or not across actions can qualitatively be appreciated by plotting the resulting policies and Q-value functions, as shown in Figure 3–6. We observe that the *Atrees* barely learn anything about the puddle's presence while the *Strees*, resorting to the same dataset, learns a rough approximation of the optimal policy.

3.3.2 Toy Problem

Both variants (*Strees* and *Atrees*) find the optimal policy given the 4-tuples acquired by the random behaviour policy. In the different noise conditions, they also do equally well. But as mentioned in the description of the problem, they cannot be optimal in this case because of the uncertainty.

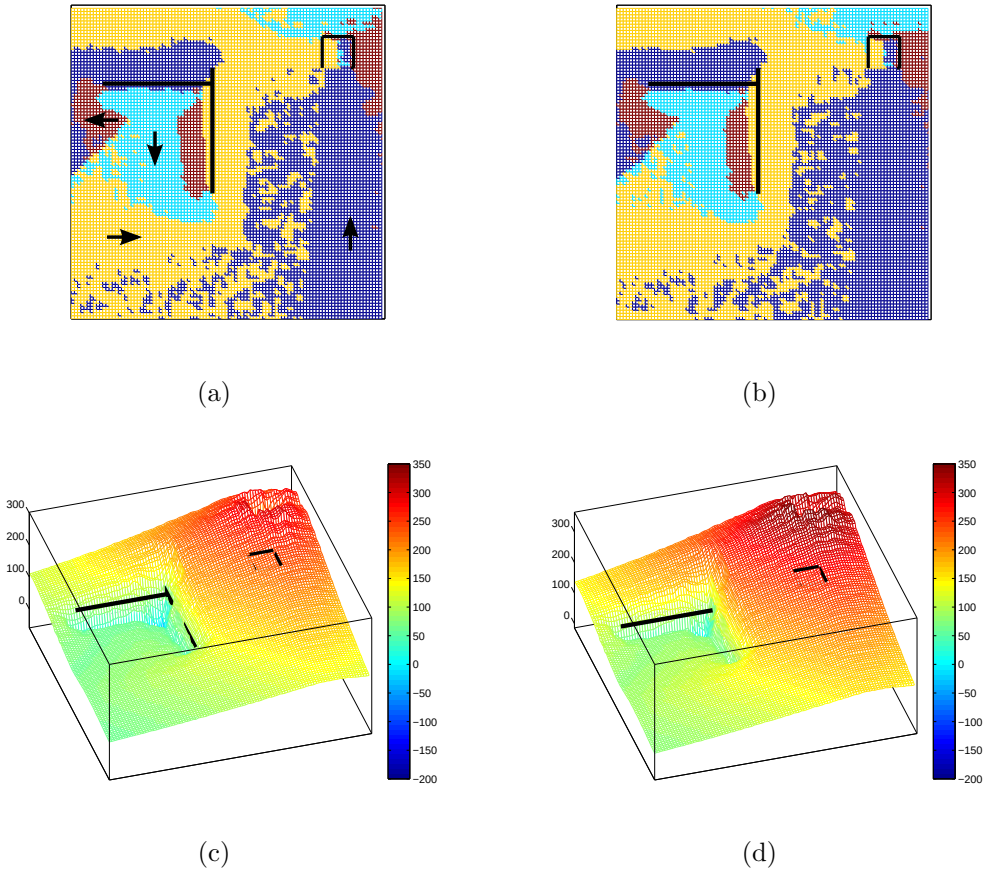

Figure 3–5: *Result of training from $\mathcal{F}_1$ (i.e., $\epsilon = 0$) with and without approximation across actions. In a), the policy obtained by the* Strees *is represented using a four-color scheme. The legend for that scheme is indicated by the arrows on the figure. The two-dimensional representation of the puddle world's state space is identical to what displayed in Figure 3–1. The figure in b) employs the same color scheme to represent the policy obtained by the* Atrees*. The policies in a) and b) are mostly identical (except in parts of the space where two actions can be chosen to act optimally) and avoid going through the puddle in order to reach the goal square. Figures c) and d) present, under each policy, the corresponding Q-value for action* UP*, $Q(\cdot, UP)$. The topology of the Q-value functions is similar but differ slightly because the* Atrees *have the tendency to overestimate the Q-value while the* Strees *have a tendency to underestimate the Q-value.*

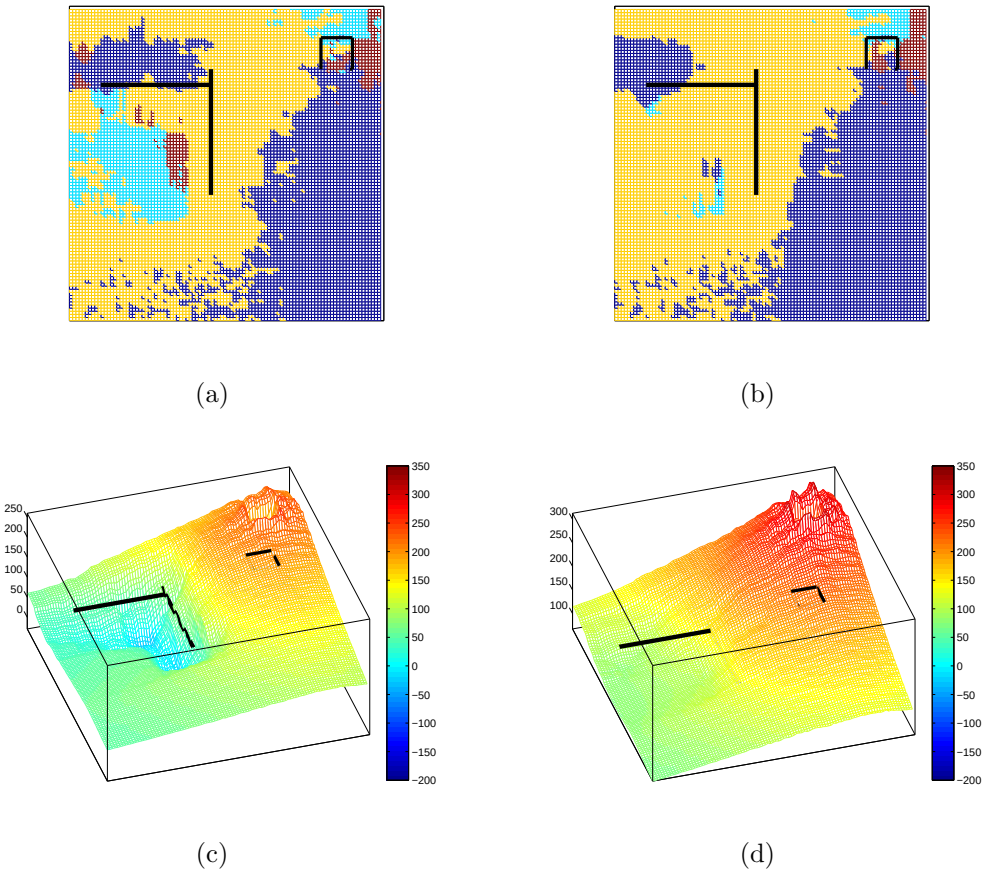

Figure 3–6: *Result of training on a dataset $\mathcal{F}_2$ (i.e., $\epsilon = 0.9999$) with (left column) and without (right column) approximation across actions. The legend is identical to the one in Figure 3–5. In a), the policy obtained with the* Strees *learns, to a certain degree, to circumvent the puddle. In b), the policy obtained with the* Atrees *pays almost no attention to the puddle. This is confirmed in c) and d) by looking at the Q-value function estimate for action* UP *learned by the* Strees *and* Atrees*, c) displays a large depression in the Q-value function around the puddle while it is difficult to observe the puddle effect on the Q-value estimate in d).*

A different story unfolds when using the data set acquired under the fixed behaviour policy. In the condition without noise, which is easier to interpret, the *Strees* find the optimal policy while the *Atrees* find a terrible policy (see Figure 3–7). A similar pattern is found in the conditions with noise.

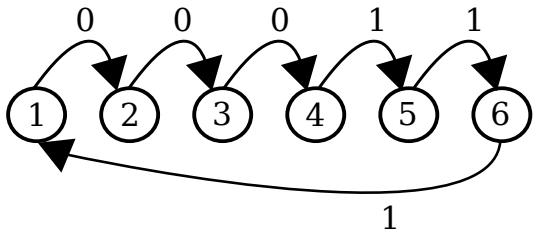

Figure 3–7: *The policy found by the Atrees when learning from the fixed behaviour policy.*

It is worthwhile to study what goes wrong exactly in this particular case. Recall that during the first iteration of the FQI algorithm, an approximation of the immediate reward for each pair of state and action is learned. The rest of the algorithm then builds on that approximation to find the $\hat{Q}$ function. In the *Atrees* case, the trees for a particular action a_0 are built during the first iteration using input-output pairs of the form (s, r) which are directly extracted from the set of 4-tuples. For the toy problem, if $a_0 = 1$, those tuples are exactly $< 1, 1, -0.1, 2 >$,$< 2, 1, -0.1, 3 >$ and $< 3, 1, -0.1, 1 >$ (assuming the condition without noise), so the regressed trees for action 1 return -0.1 for all states. Of course, this is not true since state 4, 5, and 6 are not represented, but states 4, 5, and 6 are represented in the trees for action 0 since the behaviour policy observed them under action 0. So at the end of that iteration, $\hat{Q}_1(4, 0) = -1$ while $\hat{Q}_1(4, 1) = -0.1$ which implies that $\pi(4) = 1$. This is what we see in Figure 3–7. On the other hand, for the *Strees*, tuples that contain state 4, 5, or 6 are not split on actions since they are all observed under constant

action 0 and, therefore, this split wouldn't get a good test score. Hence, for the *Strees*, we get $\hat{Q}_1(4,1) = \hat{Q}_1(4,0) = -1$, which allows the *Strees* to learn the optimal policy for this particular problem.

3.3.3 Toy Problem 'Easy choice'

The resulting statistics for the second toy problem are presented in Figure 3–8. For this problem, the generalization across actions performed by the *Strees* is detrimental when the bias in the data acquisition, δ, is too high. The *Atrees* generalize the effect of actions across the state space; hence, they are able to perform well on this problem even as δ goes to 1.

3.4 Discussion

In this chapter, we described the issue of generalizing across actions in the context of batch reinforcement learning. When environment dynamics are missing in the training data, we showed empirically that, depending on the problem, generalizing or not across actions can substantially affect performance. In applications that require the adoption of a batch RL algorithm, we typically do not have control over the exploration phase and, consequently, incomplete training datasets are commonly encountered. Therefore, a proper understanding of the failure conditions of each variant is crucial. Based on the results presented in the previous section, we can attempt to characterize the properties of a problem that makes one variant more successful than the other.

For each iteration of the FQI algorithm, the *Strees* variant is constructing a function approximation of the state-action space using *all* tuples in the training set whereas the *Atrees* are constructing a different state space approximation for

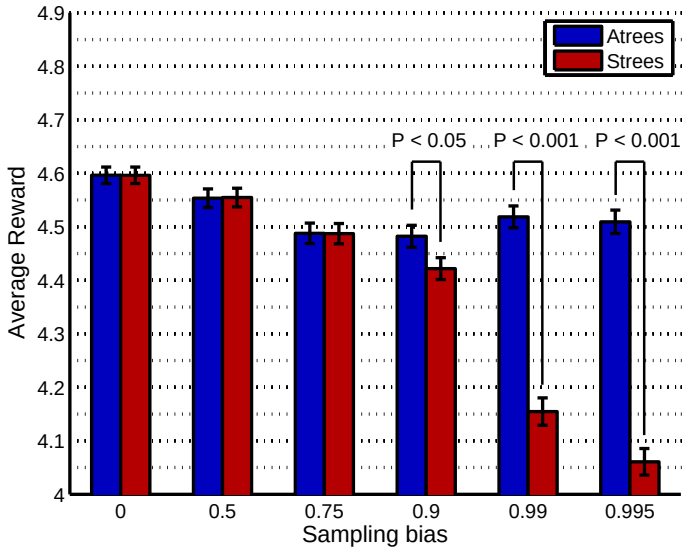

Figure 3–8: *Average reward obtained for different values of the bias, δ, when learning with (*Strees*) and without (*Atrees*) generalization across actions. The error bars represent the standard error of the means. Each bar is the result of an average over* 200 *sampled datasets.For $\delta < 0.75$, the two variants perform indistinctively—both variants suffer the same reduction in performance when $\delta > 0$. Then, as δ increases further, the general trend is that the performance of the* Atrees *stabilizes while the performance of the* Strees *drops by a significant amount. The difference between the performance of the two variants when $\delta \geq 0.9$ is statistically significant with $P > 0.05$ and with $P > 0.001$ when $\delta \geq 0.99$.*

each action in $\mathcal{A}$ using *only* the tuples that contain that action. In the Puddle World domain, that means the *Atrees* forest constructed at each iteration for action *RIGHT*, for example, is constructed independently from tuples figuring actions *UP*, *DOWN*, and *LEFT*. If the exploratory trajectories given to the FQI algorithm do not contain examples of traversing some area of the puddle with a *RIGHT* action, then the Q-value for the *RIGHT* action assigned to that area will depend on tuples with a *RIGHT* action outside that area that are potentially not part of the puddle. If that is the case, then that Q-value will not reflect the presence of the tuple and

the policy is likely to prescribe crossing that puddle area. On the other hand, in the exact same situation, the *Strees* can also rely on tuples containing other actions than *RIGHT* in that area in order to estimate the Q-value of the action *RIGHT* in that region. That value would not be exactly estimated since it would only capture the value of the states in that area and not the subtleties of the action dynamics. Nevertheless, that estimate can be enough to prevent the policy from traversing the puddle. To summarize, the characteristics of the Puddle World problem that benefit the *Strees* are that the actions only have local effects on the dynamics of the system (i.e, $|s_{t+1} - s_t|$ is small after any action a_t), the rewards carried by the actions (in the Puddle World, a small negative reward for each action) are not as significant as the rewards carried by the state themselves (in the Puddle World, the puddle and the goal square), and the region where dynamics are missing is a region that will be avoided by the optimal policy.

For problems in which it is the actions that dominate the reward function (i.e states only give a small contribution) and the actions cannot be easily related, then we expect the *Atrees* variant to do better. The 'Easy Choice' problem presented in this chapter is an extreme example of that: the reward function is completely independent of the states. In that problem, generalizing across actions can be damaging because adjacent actions can have completely different effects, for example action a_{10} gives the highest reward with probability 0.5 whereas action a_{11} always afflicts the lowest reward. Learning about actions separately, like the *Atrees*, is advantageous in those classes of problems.

To recapitulate in short form, here are some clues that might indicate superior performance of the *Strees* in a batch RL setting with an incomplete training dataset:

- local effect of actions,
- actions are on a continuum,
- reward function is dominated by the states,
- poorly-sampled region is a region avoided by the optimal policy.

On the other hand, clues that might indicate superior performance of the *Atrees* are:

- global effect of actions,
- adjacent actions have disparate effects,
- reward function is dominated by the actions,
- effect of actions in poorly-sampled region is similar to effect of actions in adjacent regions.

Finally, since the purpose of this thesis is to learn a neurostimulation controller, we outline some specific recommendations for that problem. The actions in the seizure control problem are electrical stimulations, domain knowledge tells us that the effect of a single stimulation does not have a substantial effect on the brain tissue; therefore, we can consider those as having a local effect. In the next chapter, the problem definition details will make clear that actions are also on a continuum and that the reward is dominated by the states—seizure is the main reward criterion. In addition, because of their diversity and rarity under stimulation protocols in the training data, seizure states can be argued to be less well sampled than non-seizure states; and it should come as no surprise that the optimal policy for our controller would try to stay away from seizure states. Hence, the empirical results

in this chapter suggest that using the *Strees* would be more appropriate for our neurostimulation problem.

3.5 Related Work

In the RL survey by Kaelbling et al. [35], the issue of generalization across actions is briefly mentioned. The authors recommend to generalize over actions 'when actions are described combinatorially to avoid keeping separate statistics for the huge number of actions that can be chosen'.

In [53], separate training sets are employed for each action, so their kernel-based reinforcement learning algorithm does not generalize across actions. In the paper that introduce tree-based batch RL [22], both variants seem to be favored equally without distinction.

On a real-world application of RL for autonomic resource allocation, the adoption of function approximators that generalize across action is justified by a reduction in the number of exploratory samples needed to learn a good policy [69].

In [54], in the context of learning using an embedding based on the graph Laplacian, the authors analyze the potential of the state-action basis functions for Q-value function approximation compared to the more traditional state basis functions. The state-action basis function is able to generalize across actions and is shown to present some advantages in some toy domains.

Finally, a recent work proposed an algorithm, with provable guarantees, that handles the lack of data in batch RL by being conservative [24]; it was tested, with

a similar experiment as the one in this chapter, in the Puddle World domain. However, the algorithm solely produces open-loop policies which is not appropriate for stochastic systems.

It should be mentioned that there exists a considerable body of work on RL with continuous actions that has to deal with generalization across actions [44, 40, 50], but a full review of that material is outside the scope of this thesis.

3.6 Future Work

The motivating question of this chapter is whether one should generalize across actions in a batch RL setting with discrete actions. We partially answered that question by showing that the solution was domain dependent. We also provided some rules of thumb to help decide when to generalize across actions. An evident avenue for future work is to generalize the problem further by defining conditions that can make one variant provably better than the other. Additionally, with respect to the neurostimulation optimization problem, a careful comparison of the two strategies in a realistic computational model of epilepsy would provide a more accurate estimate of the variants' behavior when applied to a real epileptic system; nonetheless, realistic computational models of epilepsy that encompass electrical stimulation are still the subject of active research.

CHAPTER 4
EXPERIMENTAL INVESTIGATION

In this chapter, we describe how to apply the techniques presented in Chapter 2 and 3 to the problem of optimizing neurostimulations in an *in vitro* model of epilepsy. We discuss the data acquisition step and how the data is then processed to form the set of 4-tuples $\mathcal{F}$ described in Section 2.3. We then present results obtained using these techniques. Since the application of the reinforcement learning techniques is directed towards controlling seizure in an in vitro model, we first depict that model and provide some background necessary to understand the rationale behind DBS.

4.1 In Vitro Model

The *in vitro* model of epilepsy we adopt for this experimentavl investigation is a rat hippocampus-entorhinal cortex (EC) slice with ictal [1] activity, induced by superfusion with the convulsant 4-aminopyridine (4AP), generated in the EC [5]. The hippocampus and EC are brain structures located in the medial temporal lobe. Cognitively, they are crucial components of memory formations and spatial reasoning [62, 52]. Pathologically, they play a special role in the formation of seizures,

[1] A recording during an epileptic seizure is said to be ictal.

especially in the case of temporal lobe epilepsy [57]. A schematic of the organization of the hippocampus-EC slice is presented in Figure 4–2, and a real picture of the experimental setup, with recording and stimulating electrode, is displayed in Figure 4–3.

As was briefly mentioned in the introduction, fixed low-frequency stimulation at 1.0 Hz, in a part of the hippocampus called the subiculum, is known to be effective in suppressing epileptiform activity in this *in vitro* model of epilepsy [17]. Those results are summarized in Figure 4–1.

As evidenced in [7], the hypothesis behind the effectiveness of repetitive low-frequency stimulation is that it mimics the naturally occurring, but dysfunctional in epilepsy patients, interictal discharges originating in the CA3. Those discharges are believed to play an important role in controlling the epileptogenesis taking place in the EC. This mechanism is illustrated in more detail in Figure 4–4.

The brain slice preparation and maintenance is detailed in the Appendix, along with a specification of the experimental apparatus.

4.2 Data Acquisition and Processing

The *in vitro* model described in Section 4.1 allows us to gather field-potential recordings. The particular protocol we follow to gather training data is outlined below.

Electrical stimulation was applied to the subiculum using low-frequency single-pulse patterns with varied timing. Each slice was subject to a stimulation protocol consisting of seven phases of stimulation patterns. Each sequence began with a control period of recording with no stimulation. Then, stimulation was applied for

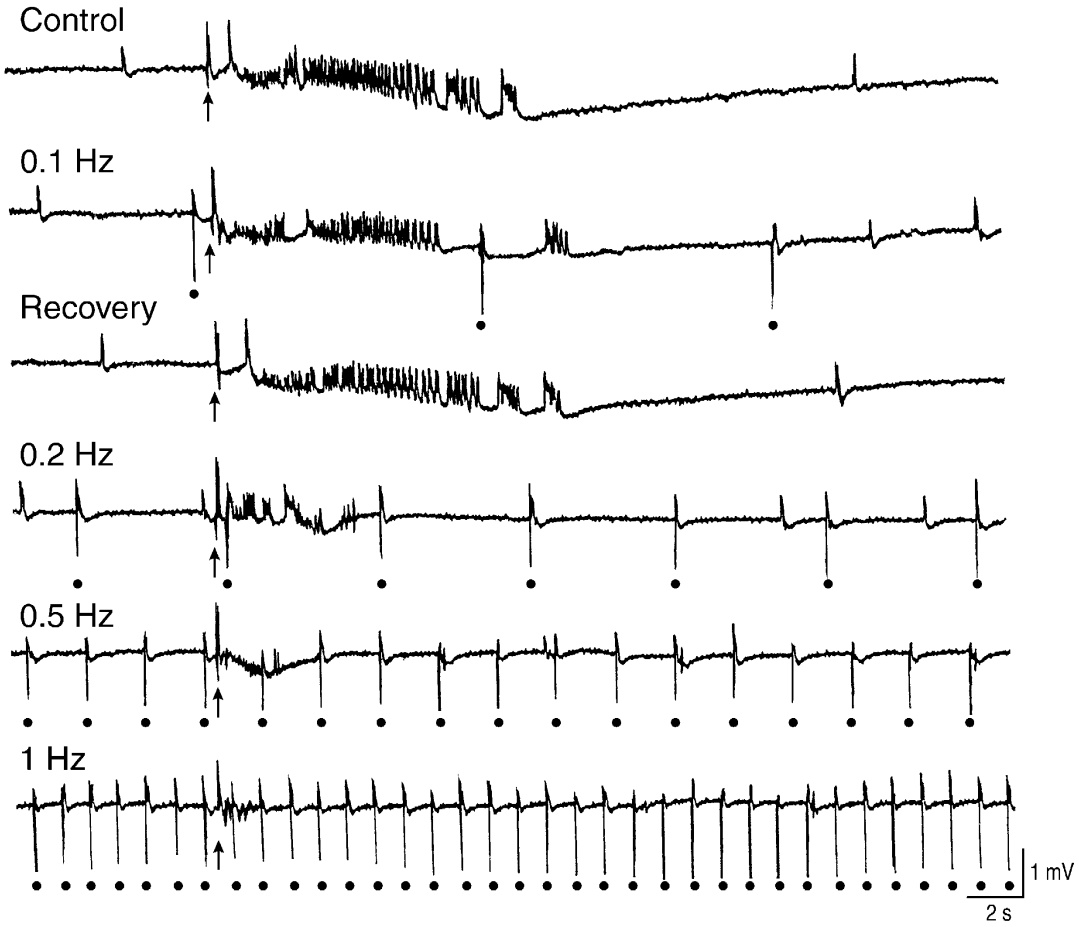

Figure 4–1: *This figure from [17] displays field potential recordings in the EC middle layers from epileptic hippocampus-EC slices under the effect of repetitive low-frequency stimulation in the subiculum for various frequencies. The start of an seizure-like event is denoted by an arrow and electrical stimulations are marked with circles under the trace. The first trace shows a seizure occurring without any stimulation, the seizure lasts approximately 15 seconds. The second trace displays the effect of stimulation at 0.1 Hz, the seizure duration stays almost constant. The third trace demonstrates the return to normal conditions after an episode of stimulation. The three subsequent traces illustrate the reduction in seizure length as the stimulation frequency increases. The 1.0 Hz frequency achieves the higher level of epileptiform activity suppression.*

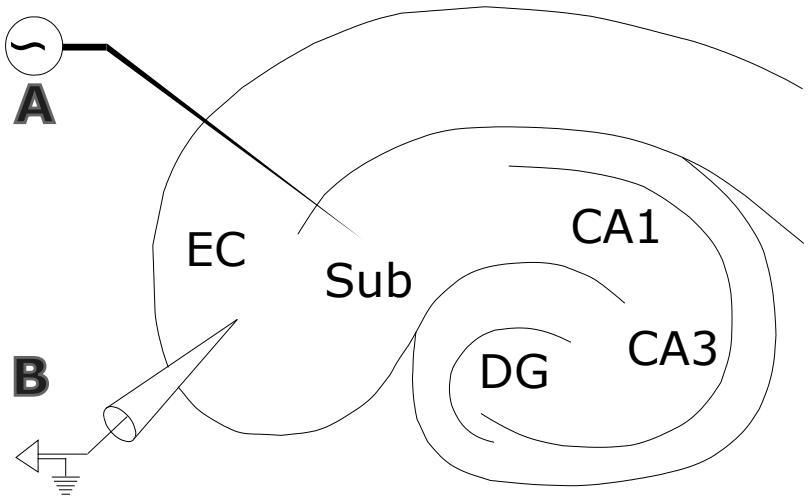

Figure 4–2: *Schematic of the hippocampus-EC slice. Relevant substructures are labeled and the location of the stimulation (A) and recording (B) electrodes is indicated.*

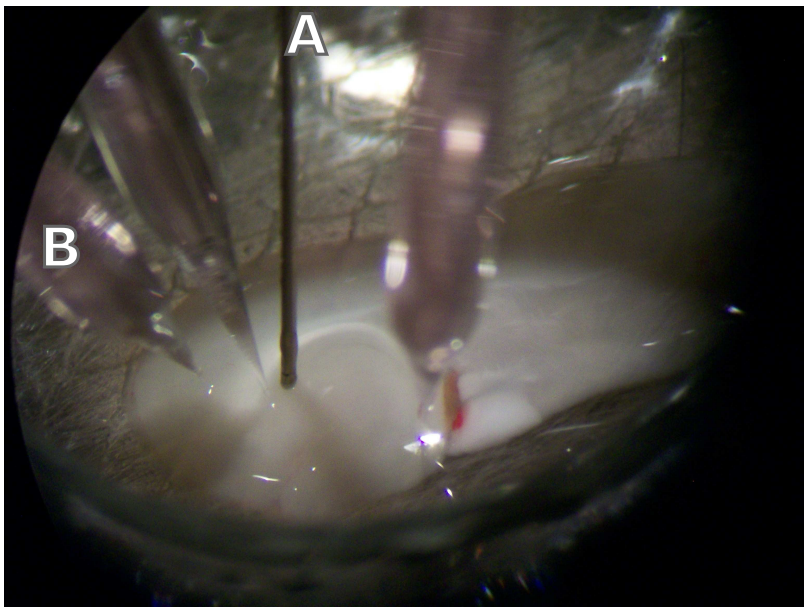

Figure 4–3: *Picture of the hippocampus-EC slice with the stimulation (A) and recording (B) electrodes.*

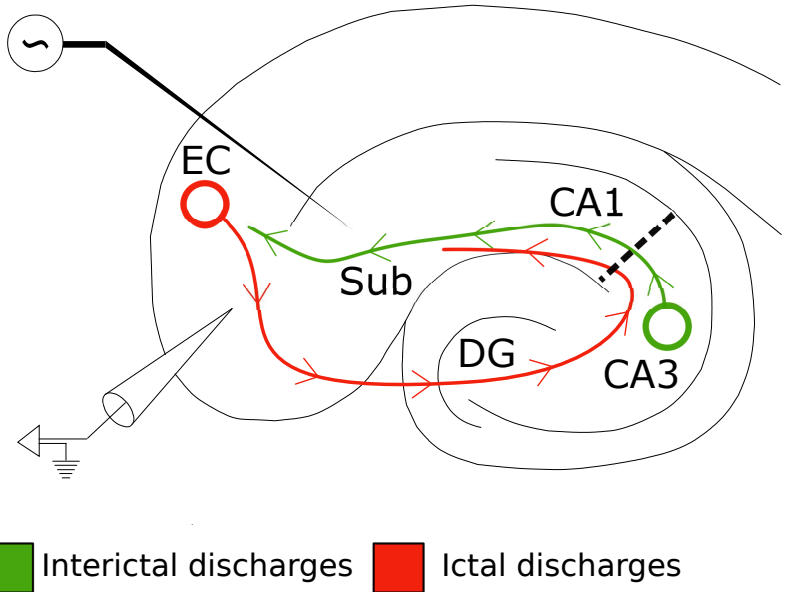

Figure 4–4: *A schematic of the seizure control hypothesis taking place in the hippocampus. Ictal discharges form in the EC, propagate to the Dentate Gyrus via the Perforant Path, and follow the hippocampus loop to be projected back in the EC. Interictal discharges form in CA3, propagate to CA1 via the Schaffer collaterals, and are projected back in the EC. According to [7], the role of the re-entrant activity is to prevent, rather than sustain, prolonged ictal events. Cell damage in the CA3 area is frequently found in temporal lobe epilepsy patients, this damage can be simulated with a cut of the Schaffer collaterals, represented here with a thick dotted line. This cut stops the interictal discharges from propagating, therefore increasing the epileptiform activity; artificial stimulation in CA1 or subiculum can artificially restore the normal condition.*

several minutes at a fixed low-frequency (1.0 Hz). Stimulation was then turned off and the slice was allowed to return to baseline for a period of several minutes. This process was repeated with stimulation at different rates (0.5 Hz and 2.0 Hz), always interleaving, between each stimulation phase, a prolonged recovery period during which no stimulation was performed. The data was collected according to this protocol rather than in a more randomized fashion because it is the procedure electrophysiologist use to gather stimulation data. We could potentially benefit from

a more sophisticated exploration policy, this should be the subject of future investigations.

Figure 4–5 shows a sample trace recorded from the EC while stimulating the subiculum at 0.5 Hz. An ictal event starts around $t = 20$sec. The stimulation artifacts are also visible in this recording. In general, the actions may or may not be visible in the EEG signal, depending on the sample rate and relative electrode placement.

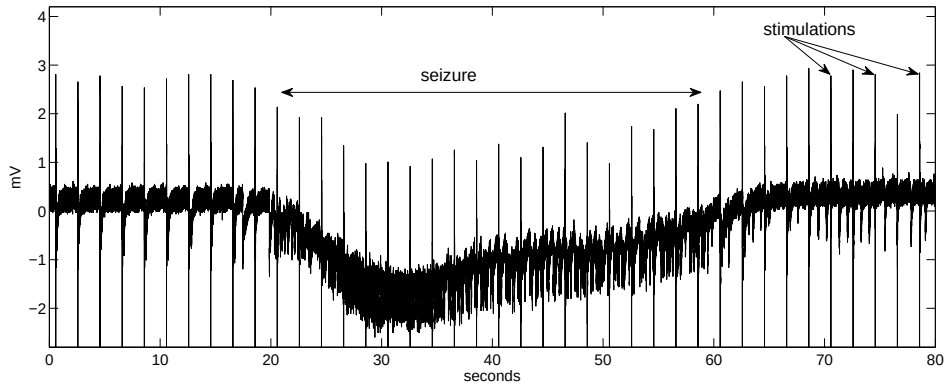

Figure 4–5: *Trace example recorded in the EC. Stimulation is applied to the subiculum at 0.5 Hz. An ictal event appears in the first half, lasting approximately 45 seconds. Periodic stimulation artifacts are observed at 2-second intervals. Interictal spikes are also observed.*

Using this procedure, we obtain raw field-potential recordings with a sample rate of 5000Hz. Subsequently, some signal processing is necessary to extract information into a form appropriate for learning. This is presented in the following section.

4.2.1 Signal Processing

Each trace is divided into a set of overlapping frames of 65536 samples (approximately 13 seconds) in length, with each frame beginning 8192 samples after the previous frame. Each frame is smoothed with a Hann window and normalized, and

the range and energy of the signal is calculated. A discrete fast Fourier transform is used to extract spectral magnitude features from the frame. Within each frame, the smoothing, normalization, and Fourier transform is repeated for the final half frame (32768 samples), quarter frame (16384 samples), eighth frame (8192 samples), and sixteenth frame (4096 samples). Low frequency components are extracted from the full-frame spectrum, and high frequency components from the subframe spectra. These features are combined with the range and energy of each subframe to yield a 109-dimensional continuous feature vector. Many other features could be extracted, for example those proposed in the literature on seizure prediction and on EEG analysis [45, 1, 55]. In fact the question of feature selection is a challenging statistical problem, which should be the subject of future investigations.

The other information which could be included is the time elapsed since beginning of the pulse train.[2] We do not include this information in the current implementation, because we assume that all recordings we use feature periodic stimulation that has been applied for a sufficiently long time to ignore edge effects.

4.2.2 Data Labeling

The adaptive control algorithm described in the following section requires a number of traces with hand-annotated state information, for automatically learning the optimal stimulation strategy. Specifically, we need to know the action a_t associated

[2] Presumably, applying a single pulse is not the same as applying a sequence of 10, or 100, or more; the system adapts to trains and responds differently depending on whether the train is of long or short duration.

with each state s_t along with the immediate reward $r_t = R(s, a)$ (see next section for an exact description of the reward function). Therefore, all recorded traces were labeled by hand, indicating on each frame whether it features ictal or normal activity, as well as which stimulation protocol was used at the time. In the future, this step could be performed by an automatic seizure detection algorithm [15, 74].

4.3 Learning

In section 2.3, the input of the FQI algorithm was specified to be a set of one-step transitions $\mathcal{F} = (s_t, a_t, r_t, s_{t+1})$. Our state space $\mathcal{S}$ is constructed such that each element s_t is a feature vector, summarizing past EEG activity, as specified in Section 4.2.1. Our action set $\mathcal{A}$ consists of four options: no stimulation, stimulation at one of the fixed frequencies of 0.5, 1.0, and 2.0 Hz. Each frame is assigned an action a_t based on the labeling information.

We define a reward function

$$r_t = R_{seizure}(s_t) + \alpha R_{stim}(a_t) \tag{4.1}$$

to penalize both stimulation and seizure occurrences. We assume $R_{seizure}(s_t)$={-1 if seizure is occurring at time t, 0 otherwise} and $R_{stim}(a_t)$={-1 if stimulation is applied at time t, 0 otherwise}. This reward function requires a quantitative trade-off between the penalty for occurrence of a seizure, and the penalty for applying stimulation. This trade-off is defined by the parameter α. In most experiments described below, we assume that a seizure is substantially more costly than delivering a single stimulation event (unless mentioned otherwise, we assume $\alpha = 0.04$). Changing this

parameter may affect the learned stimulation strategy; we investigate this further in the experiments presented below.

Each element of the training set $\mathcal{F}$ is constructed by concatenating the experience-tuples (s_t, a_t, r_t, s_{t+1}).

We assume a discrete time step of 1.6 seconds (=8192 samples). This is sufficient to compute our input features in real time, yet is sufficiently short to allow flexibility in the learned policy. For all of our experiments, the discount factor is $\gamma = 0.95$; this is a common choice in the reinforcement learning literature.

Given this formulation, the FQI algorithm can be applied with the Extra Trees to learn a policy of stimulation. Because the conditions of failure of the *Atrees*, as described in Chapter 3, seem to fit our problem description and learning dataset, we employ the *Strees* variant of the FQI algorithm as a conservative measure. The resulting policy is subjected to a multifaceted evaluation in the next section.

4.4 Results

We first conduct *in silico*, or offline, experiments to measure the quality of the obtained reinforcement learning policy. We then carry experiments *in vitro* and report the results in Section 4.4.2.

4.4.1 Offline Experiments

Many *in vitro* studies have investigated effectiveness of low-frequency periodic pacing for suppressing ictal events. For the particular animal model we are considering, the most effective fixed stimulation frequency was identified to be 1.0-2.0 Hz [7, 17]. In this section, we evaluate the ability of our reinforcement learning framework to automatically acquire an adaptive strategy from the *in vitro* recordings. We

analyze the behavior of the adaptive strategy in comparison with non-adaptive periodic stimulation strategies at low-frequencies as well as a control (no stimulation) strategy.

Measures of Performance

We consider quantitative measures which can be estimated using a *hold-out* test set, which is separate from our training data. Our original data set includes recordings from four animal slices. Therefore, during testing, we perform four-fold cross-validation, whereby the Q-function is estimated using data from three different slices, and we then measure performance on the fourth slice. We then repeat with all slice permutations. This means that data in the test set comes from a different animal than the training data. It is well-documented that epileptic seizures vary between animals, therefore this is an important test for the generalizability of our approach. In future work, an individual Q-function could be learned for each patient (or slice), thereby providing a neurostimulation strategy that is specific to each individual.

There is another subtle difficulty in using a test set to validate a target policy (e.g. the learned policy, π). That is the fact that the test set was collected *using a behavior policy*, π_b, which is different from the target policy. We cannot simply compute a score over the test set. Instead, we create a surrogate data set for the target policy by using rejection sampling to select only those segments of the test set which are consistent with the target policy. Recall that the test set is divided into

single-step episodes: (s_i, a_i, r_i, s_{i+1}). We define an indicator function:

$$I_\pi(s_i, a_i) = \begin{cases} 1 \text{ if } \pi(s_i) = a_i \\ \\ 0 \text{ otherwise} \end{cases} \tag{4.2}$$

to flag experience-tuples where the action in the test set (a_i) matches the target policy $(\pi(s_i))$. We exclude all experience-tuples that do not match the target policy. Using this indicator function, we consider two different scores to quantify the performance of the adaptive neurostimulation strategy.

The first score is an estimated proportion of seizure steps when following a particular strategy π. Again, we compare the action selected by the policy and the action in the test trace for each experience-tuple from the test trace, and count the number of states which were labeled as 'seizure':

$$\hat{S}_\pi = \frac{\sum_{i=0}^{N} I_\pi(s_i, a_i) I_{seizure}(s_i)}{\sum_{i=0}^{N} I_\pi(s_i, a_i)}, \tag{4.3}$$

where $I_{seizure}(s_i)$ indicates whether state s_i was hand-labeled as a seizure (1 if yes, 0 if no). Recall that data instances are defined on a 1.6-second window interval.

The second score calculates the estimated value function (i.e., discounted sum of rewards), averaged over all states in the test set for which the target policy π matches the behavior policy π_b. Formally,

$$\hat{V}^\pi = \frac{\sum_{i=0}^{N} I_\pi(s_i, a_i) \left[r(s_i) + \gamma \hat{Q}(s_{i+1}, \pi(s_{i+1})) \right]}{\sum_{i=0}^{N} I_\pi(s_i, a_i)}, \tag{4.4}$$

where $\hat{Q}$ is the estimated Q-function calculated by the regression algorithm (Equation 2.25). For fixed stimulation strategies, which were in fact deployed during data collection, we use the empirical return instead. This second score is considered because it reflects the expected long-term accumulated reward. Since our reward function is a linear combination of the amount of both stimulation and seizure, this is an aggregate measure of the optimization over these two components.

Offline Results

We first report on results characterizing the performance of the learning algorithm used to acquire the adaptive strategy. All error bars correspond to 1 standard error. In the case of the control and periodic strategies, this is due to variance between the four slices in the dataset. In the case of the adaptive strategy, the standard error includes both slice-to-slice variance and variance in the randomized tree regression algorithm.

Figure 4–6 compares the proportion of states in which epileptiform activity is observed under each of the policies. This corresponds to the score calculated in Equation 4.3. We first note that under control conditions, slices in the dataset exhibit a larger rate of ictal events than under any of the stimulation strategies. Next, we observe that periodic pacing at either 1.0 Hz or 2.0 Hz achieves near-complete suppression, and that performance is slightly less effective when stimulating at 0.5 Hz. Finally, we note that the adaptive strategy is able to achieve similar performance as the 0.5 Hz strategy in terms of seizure suppression.

Figure 4–7 shows the estimated long-term expected return for each of the strategies considered. This corresponds to the score calculated in Equation 4.4. The results

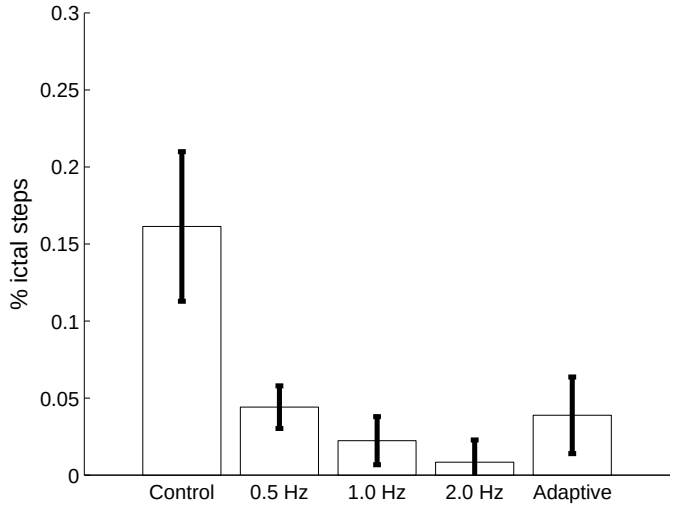

Figure 4–6: *Proportion of seizure steps (compared to non-seizure) under the following strategies: Control (no stimulation), Periodic pacing at 0.5 Hz, 1.0 Hz, 2.0 Hz, and Adaptive stimulation. The proportion of seizure / non-seizure for the Adaptive stimulation is estimated from Equation 4.3. Proportions of seizure / non-seizure for the other strategies is calculated through hand-annotations of the EEG trace by an expert.*

here show a better return for the adaptive policy, compared to the periodic stimulation and control cases. Given that all strategies (except Control) achieve similar suppression efficacy, it seems reasonable to conclude that this return gain is primarily achieved through a reduction of the stimulation in the adaptive strategy (compared to the periodic strategies).

Figure 4–8 supports this by showing the proportion of time during which stimulation is turned on under each of the conditions. We also show how this proportion changes as we re-train the adaptive strategy for different values of the parameter penalizing each stimulation action (α in Equation 4.1). As expected, when the penalty for stimulating is increased, the amount of stimulation is automatically reduced.

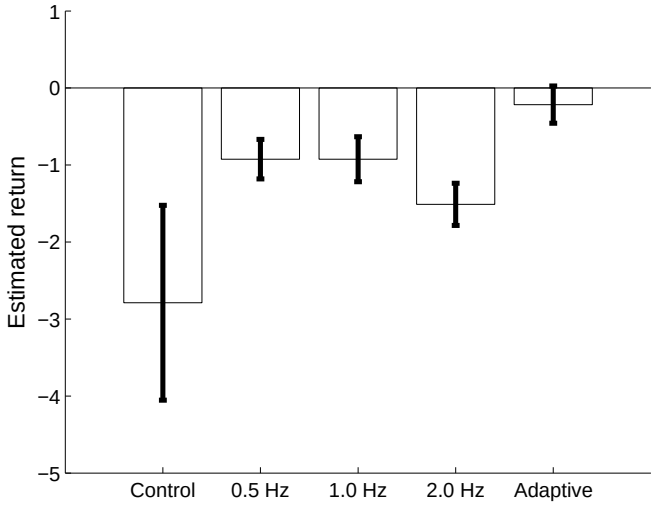

Figure 4–7: *Estimated long-term return under the following strategies: Control (no stimulation), Periodic pacing at 0.5 Hz, 1.0 Hz, 2.0 Hz, and Adaptive stimulation.*

There is substantial variation here between the different slices; in some slices some amount of stimulation would be necessary throughout most of the life of the slice to achieve reasonable suppression; in other slices it is possible to turn off any stimulation for prolonged periods of time.

Lastly, it is worth considering how changes in the reward function impact the suppression efficacy. As shown in Figure 4–9, the effect seems to be quite minimal.

4.4.2 Online Experiments

Given satisfying offline results, a more accurate evaluation is executed in the *in vitro* animal model.

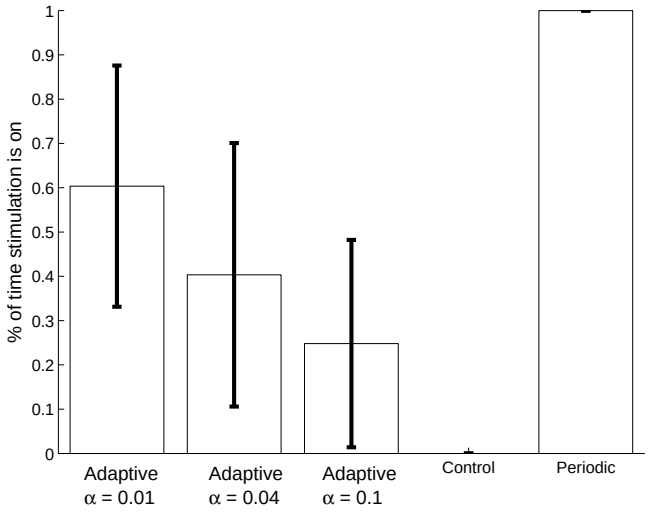

Figure 4–8: *Proportion of time under stimulation. All periodic strategies assume stimulation is on continuously. The proportion for the adaptive strategies is evaluated for different reward parameters.*

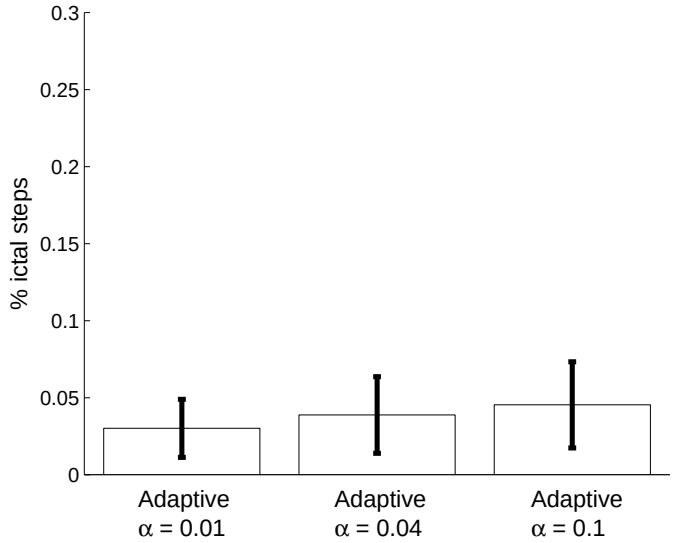

Figure 4–9: *Proportion of seizure steps as a function of the stimulation penalty. The result for $\alpha = 0.04$ is the same as shown in Figure 4–6.*

Experimental protocol

After a control period without stimulation spanning at least 3 seizures, the slice is first stimulated at a fixed frequency of 1.0 Hz. The stimulation intensity is manually tuned at the start of this protocol; the objective of the tuning is to obtain the lowest intensity that provoked a large enough response in the networks surrounding the stimulation electrode and that leads to good suppression efficacy. After the 1.0 Hz protocol, the slice is allowed to recover until the interval between ictal discharges stabilizes. Then the adaptive controller is run. The slice is again allowed to recover. The effective, or mean, frequency f used by the adaptive controller in the previous protocol is computed, $f = n_s/T$ where n_s is the number of stimulations that occurs during the adaptive protocol and T is the duration, in seconds, of the adaptive protocol. The slice is then subjected to a fixed frequency stimulation protocol at f Hz. Stimulating using the mean frequency of the adaptive protocol aims at determining whether the patterns of stimulation in the adaptive protocol provide any advantage compared to a protocol with the *same amount of stimulations* but equally distributed in time. Following this, the slice is again allowed to recover. This entire protocol is carried out 11 times, across 10 slices.[3]

[3] The total number of slices is 11. One slice was discarded because of software issues during the experiment. In one slice which had short ictal intervals, we were able to run two consecutive protocols.

Measures of performance

Let t_p^i denote the proportion of seizure time during protocol p for the slice i, and let $c(p)$ be the protocol with no stimulations immediately preceding a stimulation protocol p. Then, an estimate of the proportion of seizure time across slices, normalized with respect to the protocol without stimulations preceding it, is obtained by averaging:

$$t_p = \frac{\sum_{i=1}^{n} \frac{t_p^i}{t_{c(p)}^i}}{n}, \tag{4.5}$$

where p is a stimulation protocol ($p \in \{\text{fixed 1.0 Hz}, \text{adaptive}, \text{effective frequency}\}$) and n is the total number of slices. In other words, t_p estimates the expected percentage of seizure time compared to a protocol with no stimulation. Note that a lower t_p is better as it corresponds to a greater seizure suppression efficacy. The introduced normalization is meant to control for the fact that the proportion of seizure time is monotonically decreasing during the course of an experiment. The other performance indicator of a controller is the amount of stimulation it uses, which can be measured by its mean frequency during a protocol.

We assume that the performance estimate t_p, described in Equation 4.5, of the stimulation protocols over different experiments follows a normal distribution. However, we cannot assume that those distributions have equal variance across protocols; therefore, we use Welch's t tests in order to compare the performance of different stimulation protocols.

Online Results

In Fig. 4–10, the estimate t_p is computed, along with the *standard error of the mean*, for the different stimulation protocols. A Welch's t test ($n = 11$, $P < 0.05$) indicates that, on average, the performance of the adaptive controller is better than the effective frequency controller. This demonstrates that the patterns of stimulations, and not just the amount of stimulations, have a role to play in controlling the seizures. The performance of the adaptive controller is, in terms of seizure suppression, statistically indistinguishable from the performance of a fixed 1.0 Hz controller. Nevertheless, as exposed in Fig. 4–11, in 7 out of 11 slices the effective frequency

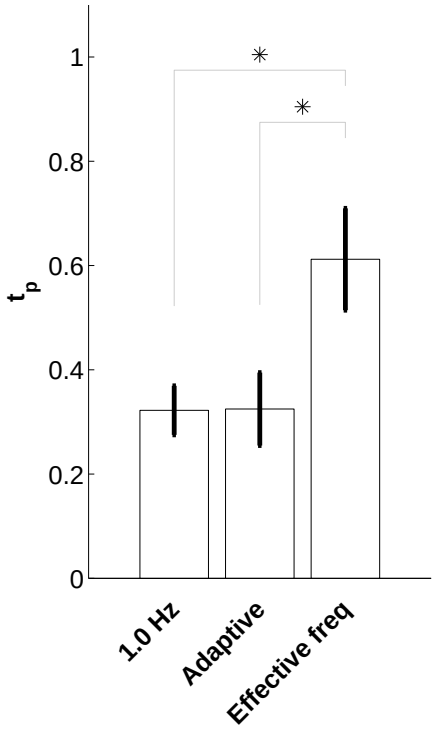

Figure 4–10: *The measure t_p, as described in Equation 4.5, is computed for the three different stimulation protocols. The error bars represent the standard error of the mean. The star symbol (*) indicates a P-value of 0.05.*

of the adaptive controller is under 1.0 Hz (the frequency range for those slices is $[0.32, 0.73]$). In 4 out of 11 slices, the effective frequency of the adaptive controller is above 1.0 Hz (the frequency range for those slices is $[1.32, 1.55]$).

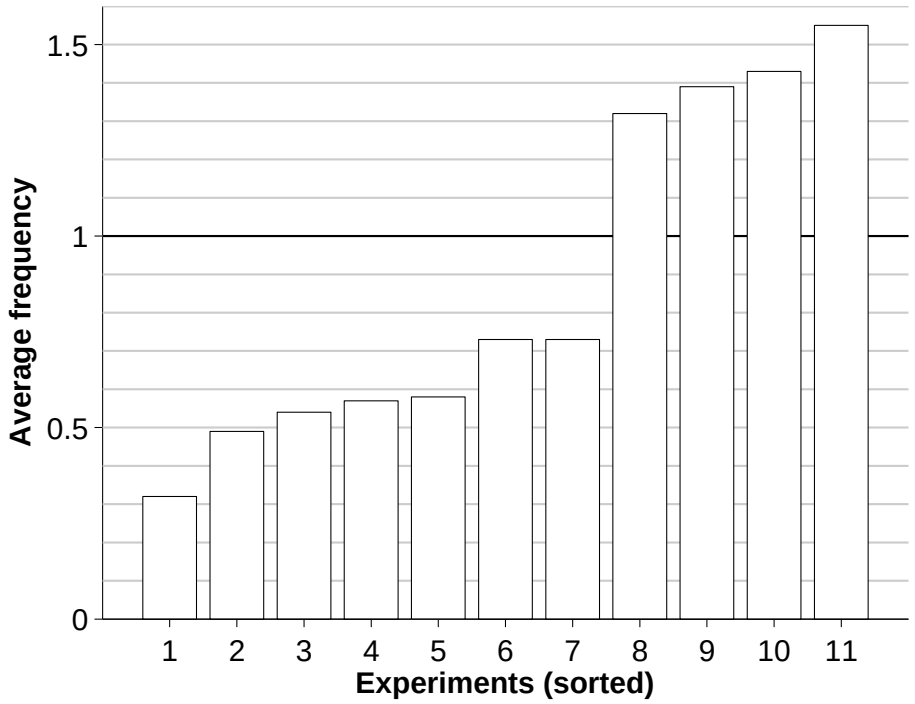

Figure 4–11: *Distribution of the effective frequency of the adaptive controller across the different experiments.*

The behavior of the adaptive policy is exemplified in Figure 4–12 with four different short traces. Each trace displays a different scenario that the RL agent encountered and reacted to. Generally, the amount of stimulation increases when a seizure is forecasted and, depending on the slice and the state of network dynamics at that time, these higher-frequency stimulations might be sustained for some time to prevent other seizures. For some slices, the RL agent stimulates more throughout its

protocol, this could be due to several factors. One possible reason is that the slice's epileptic activity is simply harder to control than in other slices, this is sometimes confirmed by observing a poor effectiveness of the 1 Hz stimulation protocol on that slice. An other reason could be that the slice's dynamics are not well represented in the agent's training data, thus the RL agent is not able to control them efficiently—with few stimulations.

4.5 Discussion

Our experimental investigation in this chapter exposes our methodology, and its evaluation, for using reinforcement learning techniques to optimize neurostimulations in an animal model of epilepsy. Results obtained offline and in real-time suggest that the learned policy can achieve the same level of performance, in terms of seizure suppression, as the best known open-loop strategies. The learned policy is more economical with its stimulations, which is its main benefit when applied to this animal model. Moreover, our results support the fact that merely stimulating at the average frequency of the adaptive controller is not enough to achieve the same suppression efficacy, highlighting the role *patterns of stimulation* play in adaptive seizure control.

Different challenges and possibilities for improvement over open-loop policies are present in more complex animal models, such as chronic *in vivo* models of epilepsy; these are outlined in the next chapter which form the conclusion of this thesis.

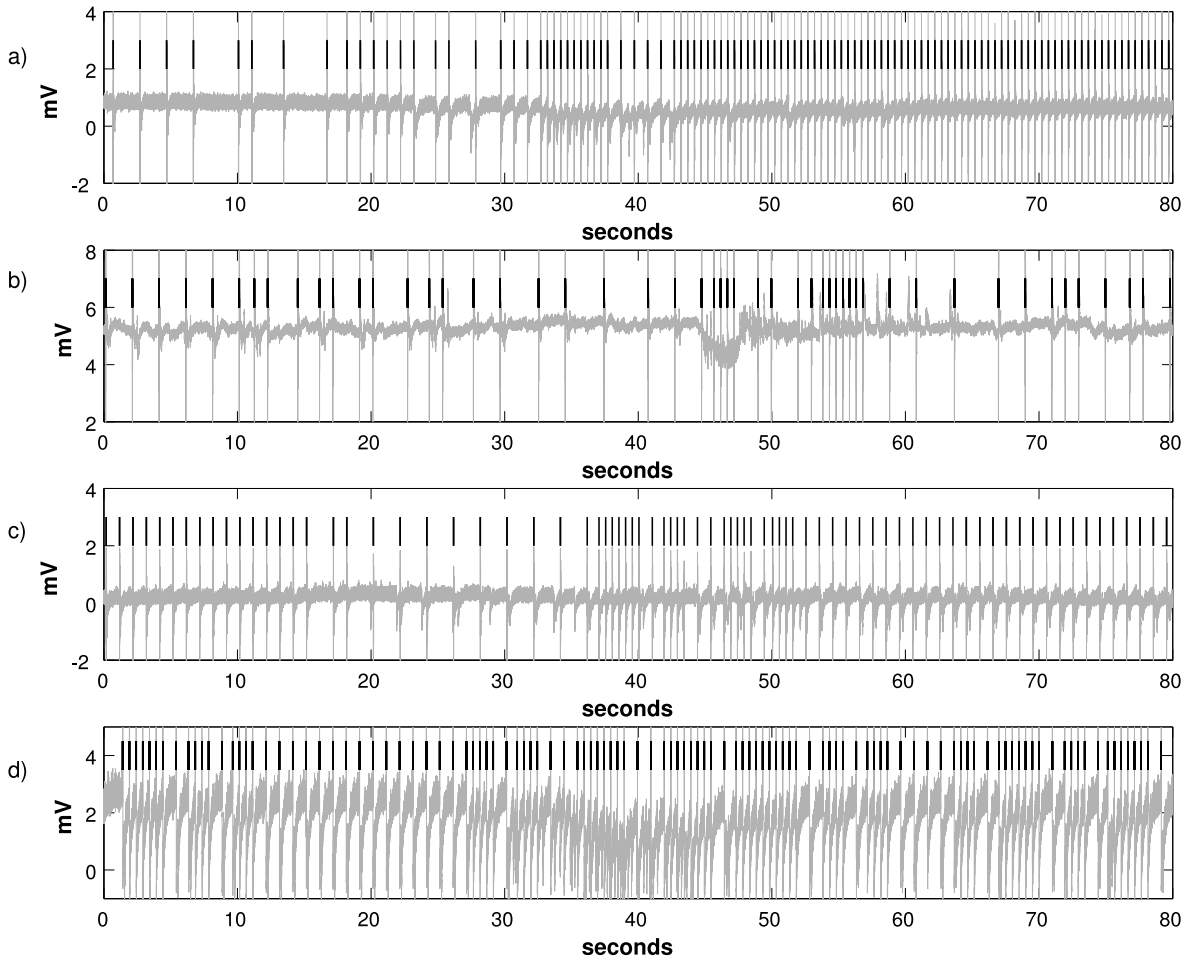

Figure 4–12: *Four different excerpts of EC recordings during the execution of the adaptive policy. The vertical thick black lines indicate stimulation events. a) Stimulation frequency increases (t = 30 s) in order to abate a seizure. The frequency remains high after the short event (t > 45 s). b) Infrequent stimulations are enough to shorten, but not suppress, a seizure (t = 45-55 s). c) Stimulation frequency increases (t = 38 s) to abort a seizure, and decreases after the abnormal activity is over (t = 52 s). d) A slice with unstable activity requires sustained stimulations. The adaptive policy is not able to suppress this seizure (t = 30-45 s), but its duration is still reduced when compared to seizure durations in the control condition.*

4.6 Related Work

We have already investigated a similar problem in a slightly different experimental setup (amygdala stimulation and perirhinal cortex recording), but our results were limited to offline measures [29].

A manifold embedding approach combined with reinforcement learning applied to the same problem (same animal model, same objectives) is still under investigation [14]. The main difference with our approach is the construction of a model from the recording data before learning takes place in the embedded space. Their approach is then subject to incorrect modelling assumptions that can bias the learning, but if the model is faithful to the real system then the obtained policy could achieve better performance than our approach because the learning process is not constrained by a small dataset.

Control, in a different slice preparation (High $[K^+]$), of spontaneous burst activity is achieved using non-linear control techniques in [60]; however, those findings have not been replicated. In that same slice preparation, an approach that stimulates with applied currents only when a seizure is automatically detected is shown to provide seizure suppression [49]. Those findings are difficult to compare directly to our work because of the different experimental setup and performance criteria employed. A review of those findings and others is available in [20].

4.7 Future Work

There are several avenues for improvement regarding the quality of the learned policy in this experimental framework. An obvious one is to improve the training data set so that it contains a more comprehensive representation of the slice dynamics.

Nevertheless, there still exists a significant amount of variance in the dynamics across slices; hence it would be beneficial to tailor the policy to a particular slice by learning from it online. Efficient RL exploration techniques could be used to learn online [51] but it is unclear if they would scale well to this problem because of the high-dimensional state space.

The second avenue to improve the policy is to develop more informative and less noisy features for the state representation. Doing this in a principled way is still an open problem but extensive trial-and-error search, or relying on features found in the seizure detection/prediction literature [15, 46], could reveal features that induce greater policies.

Finally, using a larger set of actions would provide finer-grain control to the RL agent, but at the expense of a more complex learning problem. This is likely to be a necessary step in future investigations, especially in living animals where the small action set that we employed in this study might not provide the accurate toolset required to play with the intricate brain dynamics. Although a larger but finite action set would presumably only require more training data, using a continuous action set is a difficult problem in RL and is still the subject of active research.

CHAPTER 5
CONCLUSION

This chapter concludes the thesis by providing an overview of the work presented and some discussion. Then, avenues for future work are outlined in Section 5.2.

5.1 Discussion

The main contribution of this thesis is to propose a new methodology for automatically learning adaptive neurostimulation strategies for the treatment of epileptiform disorders *in vitro*. We have demonstrated that an adaptive stimulation policy can be learned through pre-recorded data of low-frequency single-pulse fixed stimulation, using a batch reinforcement learning methodology. Analysis of the learned adaptive strategy using pre-recorded data indicates a potential reduction in the total amount of stimulation applied, compared to fixed stimulation strategies. Our analysis also indicates that the expected incidence of seizure under the adaptive policy is similar to that under the best periodic pacing strategies. It is worth emphasizing that suppression efficacy in this *in vitro* model is very high; in cases where suppression is not as effective, it may be possible for the adaptive strategy to outperform the periodic strategies in this respect. This offline analysis is validated on rodent brain slices possessing epileptiform activity. Overall, the results presented in Chapter 4 suggest that reinforcement learning is a promising methodology for learning adaptive stimulation strategies online. One of the key advantages of this methodology is its ability to trade-off between minimizing incidence (and/or duration) of seizures, and

the quantity of stimulation delivered. Another contribution of this thesis, presented in Chapter 3, is to characterize the advantages of generalizing across actions in a batch RL setting when the exploration policy employed to acquire the data does not sample the space comprehensively.

5.2 Future Work

An important question is whether the methodology outlined in this thesis will carry over to *in vivo* models of epilepsy. From a technical perspective, we do not anticipate any major technical obstacles. The reinforcement learning framework is well suited to handling larger state representations, as would be necessary in cases where there are multiple sensing electrodes, placed at different (possibly unknown) locations. The framework is also able to deal with a larger set of possible stimulation parameters (intensity, duration, higher frequencies, multiple stimulation electrodes). However we do foresee two major practical challenges. First, it will likely be necessary to collect larger amounts of data to accurately learn the Q-function. Second, it is imperative to ensure that the action strategy used during the data collection (i.e. before the learning) is 'safe'. Neither of these issues arises when working with *in silico* or even *in vitro* models of epilepsy, but they are of definite concern when dealing with *in vivo* subjects. It is worth noting that there are substantial ongoing efforts in the computer science community to address precisely those problems, namely in developing algorithms that can efficiently learn from very small data sets, and in providing formal guarantees regarding the safety (or worst-case performance) of the system during the data collection process. We hope to leverage such results as they become available.

One of the important amenities of working *in vivo*, compared to working *in vitro*, is the time frame available for data collection about the animal and also for the evaluation of a learned policy; in point of fact, a slice preparation can only be maintained with the right activity for a handful of hours, whereas *in vivo* experiments can go on for weeks. With respect to data collection, this means that a pure—but safe—exploration phase could realistically take place before trying to apply any seizure-suppressing policy. Then, after that initial stage, an agent carefully trading-off the remaining exploration and exploitation could be deployed for the long-term adaptive control of seizures. Since the brain dynamics are, assumably, continuously evolving in an individual, the exploration component might need to be kept permanently to ensure that the RL agent's model is always on par with the real dynamics and can adjust its policy accordingly.

In conclusion, this thesis presents a novel application of reinforcement learning methodologies to a challenging and important optimization problem. The potential impact of this work is tremendous, and while the early results are promising, there remains a long road of exploration and experimentation to come up with an adaptive controller for human patients.

Appendix A: Brain Slice Preparation and Maintenance

Male, adult Sprague-Dawley rat (250-300 g) were decapitated under deep isoflurane anesthesia. The brain was quickly removed and placed in cold (0–2° C) artificial cerebro-spinal fluid (aCSF), having the following composition (mM): 124 NaCl, 2 KCl, 2 $MgSO_4$, 2 $CaCl_2$, 1.25 KH_2PO_4, 26 $NaHCO_3$ and 10 D-glucose, and was continuously bubbled with CO_2 5% and O_2 95% to equilibrate at pH=7.35–7.40. We cut partially disconnected combined hippocampus-EC slices 450μm thick as previously described [56] using a VT1000S vibratome (Leica, Germany). In these brain slices, which included the most ventral part of the hippocampal formation, we observed fast CA3-driven interictal-like activity disclosed by 4AP application to be restrained to the hippocampus proper and not to be propagating to the EC (cf., [3], but see also [4]). We then transferred slices to an interface recording chamber, lying between warm ($\sim$ 32° C) aCSF and humidified gas (CO_2 5% and O_2 95%), where they were allowed to recover for at least 1 hour before beginning continuous bath-application of 4AP, continuously perfusing at $\sim$1 ml/min. We made all efforts to minimize the number of animals used and their suffering. We carried on all procedures in accordance to the CCAC (Canadian Council for Animal Care) and McGill University guidelines.

Appendix B: Field Potential Recording and Stimulation

Field potential recordings were made with ACSF-filled pipettes (tip diameter $< 10\mu$m; resistance= 5-10 MΩ) pulled from borosilicate capillary tubing (World Precision Instruments Inc., Sarasota, FL, USA) using a P-97 puller (Sutter Instrument, Novato, CA, USA). Extracellular signals were fed to a Cyberamp 380 amplifier (Molecular Devices, Palo Alto, CA) connected to a digital interface device (Digidata 1320A, Molecular Devices). Data were acquired at a sampling rate of 5 KHz, using the software Clampex 8.2 (Molecular Devices), stored on the hard drive and analyzed off-line. Recording electrodes were placed in the EC deep layers and the subiculum. Extracellular current pulses (0.1-2.25 mA, pulse width 100 μs) were delivered in the subiculum through a bipolar concentric Pt-Ir electrode (FHC, Bowdoin, ME, USA) plugged into a high voltage stimulus isolator unit (A360, WPI Inc., Sarasota, Florida, USA) connected to the pulse generator Pulsemaster A300 (WPI Inc., Sarasota, Florida, USA). The Cyberamp 380 amplifier was also connected to another digital interface device (USB-6221 M Series, National Instruments, Texas, USA) that acquired data at a sampling rate of 5 KHz using an in-house software. That software was performing signal processing in real time and was querying the adaptive controller software for the stimulation actions. Those stimulation requests were then transmitted digitally from the National Instruments digitizer to the pulse generator.

References

[1] H. Adeli, Z. Zhou, and N. Dadmehr. Analysis of EEG records in an epileptic patient using wavelet transform. *Journal of Neuroscience Methods*, 123(1):69–87, 2003.

[2] A. Antos, R. Munos, and C. Szepesvari. Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems*, volume 20, pages 9–16, 2008.

[3] M. Avoli, M. Barbarosie, A. Lcke, T. Nagao, V. Lopantsev, and R. Khling. Synchronous GABA-mediated potentials and epileptiform discharges in the rat limbic system in vitro. *Journal of Neuroscience*, 16:3912–3924, 1996.

[4] M. Avoli, G. Biagini, and M. de Curtis. Do interictal spikes sustain seizures and epileptogenesis? *Epilepsy Currents*, 6:203–207, 2006.

[5] M. Avoli, M. D'Antuono, J. Louvel, R. Kohling, G. Biagini, R. Pumain, G. DArcangelo, and V. Tancredi. Network and pharmacological mechanisms leading to epileptiform synchronization in the limbic system in vitro. *Progress in Neurobiology*, 68(3):167–207, 2002.

[6] S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta Mathematicae*, 3:133–181, 1922.

[7] M. Barbarosie and M. Avoli. CA3-driven hippocampal-entorhinal loop controls rather than sustains in vitro limbic seizures. *Journal of Neuroscience*, 17(23):9308, 1997.

[8] R.E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[9] D. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[10] D.P. Bertsekas. *Dynamic programming: deterministic and stochastic models*. Prentice-Hall, Inc., 1987.

[11] P. Boon, K. Vonck, V. De Herdt, A. Van Dycke, M. Goethals, L. Goossens, M. Van Zandijcke, T. De Smedt, I. Dewaele, R. Achten, W. Wadman, F. Dewaele, J. Caemaert, and D. Van Roost. Deep brain stimulation in patients with refractory temporal lobe epilepsy. *Epilepsia*, 48(8):1551–1560, 2007.

[12] J. Boyan and A.W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems*, volume 7, pages 369–376, 1995.

[13] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[14] K. Bush and J. Pineau. Manifold embeddings for model-based reinforcement learning under partial observability. In *Advances in Neural Information Processing Systems*, volume 22, pages 189–197. 2009.

[15] A.W.L. Chiu, S. Daniel, H. Khosravani, P.L. Carlen, and B.L. Bardakjian. Prediction of seizure onset in an in-vitro hippocampal slice model of epilepsy using gaussian-based and wavelet-based artificial neural networks. *Annals of Biomedical Engineering*, 33(6):798–810, 2005.

[16] A.A. Cohen-Gadol, M.R. Stoffman, and D.D. Spencer. Emerging surgical and radiotherapeutic techniques for treating epilepsy. *Current Opinion in Neurology*, 16(2):213, 2003.

[17] G. D'Arcangelo, G. Panuccio, V. Tancredi, and M. Avoli. Repetitive low-frequency stimulation reduces epileptiform synchronization in limbic neuronal networks. *Neurobiology of Disease*, 19(1-2):119–128, 2005.

[18] P. Dayan. The convergence of TD(lambda) for general lambda. *Machine Learning*, 8:341–362, 1992.

[19] P. Dayan, L.F. Abbott, and L. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, 2001.

[20] D.M. Durand and M. Bikson. Suppression and control of epileptiform activity by electrical stimulation: a review. In *Proceedings of the IEEE*, volume 89, pages 1065–1082, 2001.

[21] T.L. Ellis and A. Stevens. Deep brain stimulation for medically refractory epilepsy. *Neurosurgical Focus*, 25(3):11–255, 2008.

[22] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[23] D. Ernst, G. Stan, J. Gonçalves, and L. Wehenkel. Clinical data based optimal STI strategies for HIV: A reinforcement learning approach. In *15th Machine Learning Conference of Belgium and The Netherlands*, pages 667–672, 2006.

[24] R. Fonteneau, S. Murphy, L. Wehenkel, and D. Ernst. A cautious approach to generalization in reinforcement learning. In *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence*, 2010.

[25] K.N. Fountas, J.R. Smith, A.M. Murro, J. Politsky, Y.D. Park, and P.D. Jenkins. Implantation of a closed-loop stimulation in the management of medically refractory focal epilepsy. *Stereotactic and Functional Neurosurgery*, 83(4):153–158, 2005.

[26] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[27] G.J. Gordon. Stable function approximation in dynamic programming. In Morgan Kaufmann, editor, *Twelfth International Conference on Machine Learning*, pages 261–268, 1995.

[28] G.J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

[29] A. Guez, R.D. Vincent, M. Avoli, and J. Pineau. Adaptive treatment of epilepsy via batch-mode reinforcement learning. In *Proceedings of the Twentieth Innovative Applications of Artificial Intelligence Conference*, pages 1671–1678, 2008.

[30] C. Hamani, D. Andrade, M. Hodaie, R. Wennberg, and A. Lozano. Deep brain stimulation for the treatment of epilepsy. *International Journal of Neural Systems*, 19(3):213–226, 2009.

[31] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer, 2009.

[32] R.A. Howard. *Dynamic programming and Markov processes*. The MIT Press, 1960.

[33] MD Ivan Osorio, M.G. Frei, S. Sunderam, and J. Giftakis. Automated seizure abatement in humans using electrical stimulation. *Annals of Neurology*, 57:258–268, 2005.

[34] Shivaram K. and Peter S. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 94, 2007.

[35] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[36] E.H. Kossoff, E.A. Ritzl, J.M. Politsky, A.M. Murro, J.R. Smith, R.B. Duckrow, D.D. Spencer, and G.K. Bergey. Effect of an external responsive neurostimulator on seizures and electrographic discharges during subdural electrode monitoring. *Epilepsia*, 45(12):1560–1567, 2004.

[37] E.H. Kossoff, E.K. Ritzl, J.M. Politsky, A.M. Murro, J.R. Smith, R.B. Duckrow, D.D. Spencer, and G.K. Bergey. Effect of an external responsive neurostimulator on seizures and electrographic discharges during subdural electrode monitoring. *Epilepsia*, 45(12):1560–1567, 2004.

[38] I.A.W. Kotsopoulos, T. van Merode, F.G.H. Kessels, M.C. de Krom, and J.A. Knottnerus. Systematic review and meta-analysis of incidence studies of epilepsy and unprovoked seizures. *Epilepsia*, 43(11):1402–1409, 2002.

[39] P. Kwan and M.J. Brodie. Early identification of refractory epilepsy. *New England Journal of Medicine*, 342(5):314, 2000.

[40] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In *Advances in Neural Information Processing Systems*, volume 20, pages 833–840. 2008.

[41] W. Löscher and D. Schmidt. New horizons in the development of antiepileptic drugs: the search for new targets. *Epilepsy research*, 60(2-3):77–159, 2004.

[42] H.R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, and R.S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, volume 22, pages 1204–1212, 2009.

[43] W.J. Marks. Deep brain stimulation in epilepsy. *Deep Brain Stimulation in Neurological and Psychiatric Disorders*, pages 561–569, 2008.

[44] J.D.R. Millán, D. Posenato, and E. Dedieu. Continuous-action Q-learning. *Machine Learning*, 49(2):247–265, 2002.

[45] F. Mohrmann, T. Kreuz, C. Rieke, R.G. Andrzejak, A. Kraskov, P. David, C.E. Elger, and K. Lehnertz. On the predictability of epileptic seizures. *Clinical Neuropsychology*, 116:569–587, 2005.

[46] F. Mormann, R.G. Andrzejak, C.E. Elger, and K. Lehnertz. Seizure prediction: the long and winding road. *Brain*, 130(2):314, 2007.

[47] R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *The Journal of Machine Learning Research*, 9:815–857, 2008.

[48] S.A. Murphy, D.W. Oslin, A.J. Rush, and J. Zhu. Methodological challenges in constructing effective treatment sequences for chronic psychiatric disorders. *Neuropsychopharmacology*, 32(2):257–262, 2006.

[49] M. Nakagawa and D. Durand. Suppression of spontaneous epileptiform activity with applied currents. *Brain Research*, 567(2):241–247, 1991.

[50] G. Neumann and J. Peters. Fitted Q-iteration by advantage weighted regression. In *Advances in Neural Information Processing Systems*, volume 21, pages 1177–1184. 2009.

[51] A. Nouri and M.L. Littman. Multi-resolution exploration in continuous spaces. In *Advances in Neural Information Processing Systems*, volume 21, pages 1209–1216, 2008.

[52] J. O'Keefe and L. Nadel. *The hippocampus as a cognitive map*. Oxford University Press, USA, 1978.

[53] D. Ormoneit and Ś. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49:161–178, 2002.

[54] S. Osentoski. *Action-based representation discovery in markov decision processes*. PhD thesis, University of Massachusetts, 2009.

[55] H. Osterhage, F. Mormann, T. Wagner, and K. Lehnertz. Measuring the directionality of coupling: Phase versus state space dynamics and application to EEG time series. *International Journal of Neural Systems*, 17(3):139–148, 2007.

[56] G. Panuccio, M. D'Antuono, P. de Guzman, L. De Lannoy, G. Biagini, and M. Avoli. In vitro ictogenesis and parahippocampal networks in a rodent model of temporal lobe epilepsy. *Neurobiology of Disease*, 2010.

[57] D. Paré, M. Decurtis, and R. Llinas. Role of the hippocampal-entorhinal loop in temporal lobe epilepsy: extra-and intracellular study in the isolated guinea pig brain in vitro. *Journal of Neuroscience*, 12(5):1867, 1992.

[58] J. Pineau, A. Guez, R. Vincent, G. Panuccio, and M. Avoli. Treating epilepsy via adaptive neurostimulation: a reinforcement learning approach. *International Journal of Neural Systems*, 19(4):227, 2009.

[59] D. Precup, R.S. Sutton, and S. Dasgupta. Off-policy temporal-difference learning with function approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 417–424, 2001.

[60] S.J. Schiff, K. Jerger, D.H. Duong, T. Chang, M.L. Spano, and W.L. Ditto. Controlling chaos in the brain. *Nature*, 370(6491):615–620, 1994.

[61] Y. Schiller and Y. Bankirer. Cellular mechanisms underlying antiepileptic effects of low-and high-frequency electrical stimulation in acute epilepsy in neocortical brain slices in vitro. *Journal of Neurophysiology*, 97(3):1887, 2007.

[62] W.B. Scoville and B. Milner. Loss of recent memory after bilateral hippocampal lesions. *Journal of Neurology, Neurosurgery, and Psychiatry*, 20(1):11, 1957.

[63] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[64] R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044, 1996.

[65] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[66] R.S. Sutton, C. Szepesvári, and H.R. Maei. A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *Advances in Neural Information Processing Systems*, volume 21, pages 1609–1616, 2009.

[67] C. Szepesvári. Reinforcement learning algorithms for MDPs. Technical report, University of Alberta, 2009.

[68] G. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[69] G. Tesauro, N.K. Jong, R. Das, and M.N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[70] W.H. Theodore and R.S. Fisher. Brain stimulation for epilepsy. *The Lancet Neurology*, 3(2):111–118, 2004.

[71] J.N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.

[72] J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[73] S. Vilar, L. Santana, and E. Uriarte. Probabilistic neural network model for the in silico evaluation of anti-HIV activity and mechanism of action. *Journal of Medicinal Chemistry*, 49(3):1118–1124, 2006.

[74] R.D. Vincent, J. Pineau, P. Guzman, and M. Avoli. Recurrent boosting for classification of natural and synthetic time-series data. In *Proceedings of the 20th conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pages 192–203. Springer-Verlag, 2007.

[75] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.