

https://dspace.org/

# A reinforcement learning design for HIV clinical trials

**Parbhoo, Sonali**

**2014-07-30**
http://hdl.handle.net/10539/15067

Downloaded from a DSpace repository. https://dspace.org/

# A Reinforcement Learning Design for HIV Clinical Trials

**Sonali Parbhoo**

School of Computer Science
University of the Witwatersrand
Private Bag 3, Wits 2050, Johannesburg, South Africa




A dissertation submitted for the degree of
*Master of Science*

2014

For Bhavnafoi

# Abstract

Determining effective treatment strategies for life-threatening illnesses such as HIV is a significant problem in clinical research. Currently, HIV treatment involves using combinations of anti-HIV drugs to inhibit the formation of drug-resistant strains. From a clinician's perspective, this usually requires careful selection of drugs on the basis of an individual's immune responses at a particular time. As the number of drugs available for treatment increases, this task becomes difficult. In a clinical trial setting, the task is even more challenging since experience using new drugs is limited. For these reasons, this research examines whether machine learning techniques, and more specifically batch reinforcement learning, can be used for the purposes of determining the appropriate treatment for an HIV-infected patient at a particular time. To do so, we consider using fitted $Q$-iteration with extremely randomized trees, neural fitted $Q$-iteration and least squares policy iteration. The use of batch reinforcement learning means that samples of patient data are captured prior to learning to avoid imposing risks on a patient. Because samples are re-used, these methods are data-efficient and particularly suited to situations where large amounts of data are unavailable. We apply each of these learning methods to both numerically generated and real data sets. Results from this research highlight the advantages and disadvantages associated with each learning technique. Real data testing has revealed that these batch reinforcement learning techniques have the ability to suggest treatments that are reasonably consistent with those prescribed by clinicians. The inclusion of additional state variables describing more about an individual's health could further improve this learning process. Ultimately, the use of such reinforcement learning methods could be coupled with a clinician's knowledge for enhanced treatment design.

**Keywords**: Batch reinforcement learning, fitted $Q$-iteration, extremely randomized trees, neural networks, least squares policy iteration, treatment simplification, HAART, CD4+ T-lymphocyte, viral load.

# Declaration

I declare that this is my own work. It is being submitted for the Degree of Master of Science to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other university.

_____

Signature

_____

Date

# Acknowledgments

I am very grateful to my supervisor Prof. Clint Van Alten for sharing his knowledge, and for his guidance and support throughout this research. I am also thankful to my co-supervisor Pravesh Ranchod for his advice and comments on drafts of this dissertation. I am especially thankful to Dr Sarah Stacey from The Charlotte Maxeke Johannesburg Academic Hospital and Dr Minakshi Jivan for their assistance in obtaining patient data. This research would not have been possible without the hard work of counsellors Sibongile Radebe, Hlengiwe Mtshali and Buyisiwe Bhengu at The Charlotte Maxeke Johannesburg Academic Hospital, who have spent numerous hours gathering and recording patient information – for this, I am extremely grateful. Many thanks to Raphael Fonteneau from the University of Liège for answering questions about extra trees and Louis du Plessis from the Theoretical Biology group at ETH Zürich for helpful discussions regarding this research. I have benefitted from interactions with the Artificial Intelligence and Machine Learning (AIML) group at the University of the Witwatersrand. My sincere thanks go to Raymond Phillips for insightful comments on my simulations, and Bongani Shongwe for proofreading versions of this dissertation at its later stages. I acknowledge financial support from the National Research Foundation (NRF).[1] Finally, I would like to thank my father, Anant, for his constant support and interest in this research; my mother, Nisha, for accompanying me on many trips to the hospital; my aunt, Bhavna, to whom this dissertation is dedicated, and especially my sister, Priya, for all the little things.

[1] Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

# Contents

**Contents** vi

**List of Figures** ix

**List of Tables** xii

**List of Algorithms** xiii

**1 Introduction** 1

**2 Biological Background** 5

2.1 Introduction . . . 5
2.2 The immune system . . . 5
2.2.1 The innate immune system . . . 5
2.2.2 The acquired immune system . . . 6
2.3 The Human Immunodeficiency Virus . . . 11
2.3.1 Viral structure of HIV . . . 11
2.3.2 The HIV replication cycle . . . 12
2.3.3 Subtypes and strains of HIV . . . 14
2.4 The immune response to HIV . . . 15
2.4.1 Innate immune response to primary HIV infection . . . 16
2.4.2 T-cell and antibody response to primary HIV infection . . . 16
2.4.3 Immunologic events during clinical latency and AIDS . . . 17
2.5 Drug therapy for management of HIV . . . 18
2.5.1 Non-nucleoside reverse transcriptase inhibitors (NNRTIs) . . . 18
2.5.2 Nucleoside reverse transcriptase inhibitors (NRTIs) . . . 19
2.5.3 Protease inhibitors (PIs) . . . 20
2.5.4 Fusion inhibitors . . . 21
2.5.5 Integrase inhibitors . . . 21
2.6 Initial treatment strategies: HAART and STIs . . . 22
2.6.1 Rationale for intermittent therapy . . . 22
2.6.2 Assessing the efficacy of treatment interruption . . . 23
2.6.3 Controversy following the SMART study . . . 23
2.7 A treatment simplification approach to HAART . . . 24

2.7.1 PI-sparing approaches for treatment simplification . . . . . 24
2.7.2 PI-boosting approaches for treatment simplification . . . . . 24
2.7.3 Implications for HAART . . . . . 25
2.8 Exploring the alternatives: vaccine development . . . . . 26
2.9 Conclusion . . . . . 28

**3 Reinforcement Learning Background 29**
3.1 Introduction . . . . . 29
3.2 The reinforcement learning paradigm . . . . . 29
3.2.1 Markov decision processes . . . . . 30
3.2.2 The return function . . . . . 31
3.2.3 The value function . . . . . 31
3.3 Model-based value iteration . . . . . 33
3.4 Model-free value iteration: the $Q$-learning framework . . . . . 34
3.5 Batch reinforcement learning methods . . . . . 35
3.6 Fitted $Q$-iteration . . . . . 36
3.6.1 The algorithm . . . . . 38
3.7 Extremely randomized trees . . . . . 39
3.7.1 Single tree regression . . . . . 39
3.7.2 Ensembles of extremely randomized trees . . . . . 39
3.7.3 The extra trees algorithm . . . . . 40
3.8 Neural fitted $Q$-iteration . . . . . 44
3.8.1 The multilayer perceptron as a function approximator . . . . . 45
3.8.2 The algorithm . . . . . 46
3.9 Least Squares methods for approximate policy evaluation . . . . . 47
3.10 Projected policy evaluation . . . . . 49
3.10.1 Least Squares Temporal Difference learning for $Q$-values . . . . . 50
3.10.2 Least Squares Policy Iteration . . . . . 52
3.11 Benchmark domains . . . . . 53
3.11.1 The swing-up acrobot . . . . . 53
3.11.2 The mountain car . . . . . 54
3.12 Conclusion . . . . . 54

**4 Research Methodology 56**
4.1 Introduction . . . . . 56
4.2 Aim of this research . . . . . 56
4.3 Research questions . . . . . 57
4.4 Research methodology . . . . . 57
4.5 Data collection and simulation . . . . . 57
4.5.1 HIV patient data collection . . . . . 58
4.5.2 Simulating HIV patient data using a mathematical model . . . . 58
4.6 Modelling the HIV drug scheduling problem as an MDP . . . . . 61
4.6.1 An MDP formulation for the simulated case . . . . . 62

4.6.2 An MDP formulation for the real data case . . . . . 62
4.7 Implementation of batch reinforcement learning techniques . . . . . 65
4.8 Conclusion . . . . . 66

**5 Results and Discussion** **67**
5.1 Introduction . . . . . 67
5.2 System specifications . . . . . 67
5.3 Benchmark domain experimentation . . . . . 68
5.3.1 Performance assessment metrics . . . . . 68
5.3.2 Comparison of algorithmic outcomes for mountain car . . . . . 68
5.3.3 Discussion of results for mountain car domain . . . . . 77
5.3.4 Comparison of algorithmic outcomes for swing-up acrobot . . . . 78
5.3.5 Discussion of results for the swing-up acrobot domain . . . . . 85
5.4 Experimentation on simulated data . . . . . 86
5.4.1 Discussion of results from testing batch RL on simulated HIV patient data . . . . . 93
5.4.2 A note about the size of the sample set used for simulated data testing . . . . . 94
5.5 Experimentation on real patient data . . . . . 94
5.5.1 Discussion of results in a real HIV setting . . . . . 96
5.5.2 A note about supervised learning . . . . . 99
5.6 Discussion in relation to research questions . . . . . 101
5.7 Conclusion . . . . . 102

**6 Conclusions and Future Work** **104**

**A Benchmark Domains** **106**
A.1 The Acrobot Swing-Up Control Problem . . . . . 106
A.2 Mountain Car . . . . . 107

**B Additional Results** **108**
B.1 Experimentation using a set of simulated HIV data of a smaller size . . 108

**References** **115**

# List of Figures


2.1 Summary of humoral and cell-mediated immune responses (Carter, 2011). 10
2.2 Structure of the HI virion (Mann & Ward, 2006). . . . . 11
2.3 The HIV replication cycle (Figure modified from Archer (2008)). . . . . 12
2.4 Phases of acute HIV infection (Borrow, 2011). . . . . 15

3.1 Stages of batch reinforcement learning. . . . . 35
3.2 Multilayer perceptron structure for neural fitted $Q$-iteration (Figure adapted from Riedmiller (2010)). . . . . 44
3.3 The swing-up acrobot (Boone, 1997). . . . . 53
3.4 Illustration of the mountain car problem (Figure modified from Tanner (2009)). . . . . 54

5.1 Average run times of each algorithm when applied to the mountain car task using sample sets of varying sizes. . . . . 70
5.2 (a) - (e): Graphical representation of optimal policy, $\widehat{\pi}^*_N$, after $N$ steps of neural fitted $Q$-iteration, where $N = 10$, 20, 30, 40 and 50 respectively. (f): Trajectory from $s_0 = (-0.5, 0)$ under policy $\widehat{\pi}^*_{50}$. . . . . 73
5.3 (a) - (e): Graphical representation of optimal policy, $\widehat{\pi}^*_N$, after $N$ steps of fitted $Q$-iteration with extra trees, where $N = 10$, 20, 30, 40 and 50 respectively. (f): Trajectory from $s_0 = (-0.5, 0)$ under policy $\widehat{\pi}^*_{50}$. . . . . 74
5.4 (a) - (e): Graphical representation of optimal policy, $\widehat{\pi}^*_N$, after $N$ steps of LSPI, where $N = 10$, 20, 30, 40 and 50 respectively. (f): Trajectory from $s_0 = (-0.5, 0)$ under policy $\widehat{\pi}^*_{50}$. . . . . 75
5.5 Example of the trajectory resulting from an unsuccessful policy on the mountain car domain. . . . . 76
5.6 Average run times of each algorithm when applied to the swing-up acrobot task using sample sets of varying sizes. . . . . 80
5.7 Policy $\widehat{\pi}^*_{50}$ obtained from applying neural fitted $Q$-iteration to the swing-up acrobot task. . . . . 81
5.8 Policy $\widehat{\pi}^*_{50}$ obtained from applying fitted $Q$-iteration with extra trees to the swing-up acrobot task. . . . . 81
5.9 Policy $\widehat{\pi}^*_{50}$ obtained from applying LSPI to swing-up acrobot task. . . . 81
5.10 Positioning of acrobot at various steps under $\widehat{\pi}^*_{50}$ using neural fitted $Q$-iteration. . . . . 82

5.11 Positioning of acrobot at various steps under $\widehat{\pi}^*_{50}$ using fitted $Q$-iteration with extra trees. . . . . 83
5.12 Positioning of acrobot at various steps under $\widehat{\pi}^*_{50}$ using LSPI. . . . . 84
5.13 Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using neural fitted $Q$-iteration. 87
5.14 Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using fitted $Q$-iteration with extra trees. . . . . 88
5.15 Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using LSPI. . . . . 89
5.16 Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying neural fitted $Q$-iteration. . . . . 90
5.17 Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying fitted $Q$-iteration with extra trees. . . . . 91
5.18 Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying LSPI. . . . . 92
5.19 Frequency of recommended drug combinations when applying neural fitted $Q$-iteration to the validation set used for trial 5. . . . . 98
5.20 Frequency of prescribed drug combinations in the validation set used for trial 5. . . . . 99

B.1 Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using neural fitted $Q$-iteration with $|\mathcal{F}| = 2\ 000$ samples. . . . . 109
B.2 Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using fitted $Q$-iteration with extra trees where $|\mathcal{F}| = 2\ 000$ samples. . . . . 110
B.3 Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using LSPI and $|\mathcal{F}| = 2\ 000$ samples. . . . . 111
B.4 Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying neural fitted $Q$-iteration across a smaller sample set of $|\mathcal{F}| = 2\ 000$ samples. . . . . 112
B.5 Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying fitted $Q$-iteration with extra trees across a smaller sample set of $|\mathcal{F}| = 2\ 000$ samples. . . . . 113

B.6 Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying LSPI across a smaller sample set of $|\mathcal{F}| = 2\ 000$ samples. 114

# List of Tables


2.1 List of currently approved NNRTIs with their dosages (Smith, 2013). . . 18
2.2 List of currently approved NRTIs with their dosages (Smith, 2013). . . . 19
2.3 List of currently approved PIs with their dosages (Smith, 2013). . . . . 20
2.4 List of currently approved fusion/entry inhibitors with their dosages (Smith, 2013). . . . . 21
2.5 List of currently approved integrase inhibitors with their dosages (Smith, 2013). . . . . 21
2.6 List of current single tablet regimens and the contents of each tablet. . . 25

4.1 Parameters used in Equations 4.5.1 – 4.5.6 (Adams et al., 2004). . . . . 60

5.1 Run times of neural fitted $Q$-iteration on mountain car sample sets of varying sizes over 5 trials. . . . . 69
5.2 Run times of fitted $Q$-iteration using extra trees on mountain car sample sets of varying sizes over 5 trials. . . . . 69
5.3 Run times of LSPI on mountain car sample sets of varying sizes over 5 trials. . . . . 70
5.4 Number of successful policies computed under each learning technique over 30 sample sets. . . . . 76
5.5 Number of steps taken by each algorithm to reach goal state under optimal policy over 10 trials. . . . . 77
5.6 Run times of neural fitted $Q$-iteration on acrobot sample sets of varying sizes over 5 trials. . . . . 79
5.7 Run times of fitted $Q$-iteration using extra trees on acrobot sample sets of varying sizes over 5 trials. . . . . 79
5.8 Run times of LSPI on acrobot sample sets of varying sizes over 5 trials. 79
5.9 Number of steps taken by each algorithm to reach goal state of acrobot task under optimal policy over 10 trials. . . . . 85
5.10 Average consistency between learned actions and actions taken by clinicians using FQI with extra trees, NFQ and LSPI respectively. . . . . 96
5.11 Average consistency between learned actions and actions taken by clinicians using neural networks alone. . . . . 100

A.1 Parameters of the swing-up acrobot control problem. . . . . 107

# List of Algorithms


1 Pseudocode for the $Q$-value iteration algorithm (Buşoniu et al., 2010). . 34
2 Pseudocode for the $Q$-learning algorithm (Sutton & Barto, 1998). . . . . 35
3 Pseudocode for the fitted $Q$-iteration algorithm (Ernst et al., 2005). . . 38
4 Pseudocode of the extra trees algorithm (Geurts et al., 2006). . . . . 41
5 Pseudocode of the algorithm used to predict the output of a tree (Buşoniu et al., 2010). . . . . 44
6 Pseudocode of the neural fitted $Q$-iteration algorithm (Riedmiller, 2005a). 47
7 Pseudocode of the LSTD-$Q$ algorithm (Lagoudakis & Parr, 2003). . . . 52
8 Pseudocode of the LSPI algorithm (Lagoudakis & Parr, 2003). . . . . 52

# Chapter 1

# Introduction

Determining suitable treatment regimens for life-threatening illnesses such as HIV, remains one of the key aims of medical research. Typically, patients suffering from these chronic illnesses are prescribed a series of treatments in order to maximize positive outcomes. This usually involves selecting the optimal sequences of treatment or combinations of drugs for the patient over time and specifying the duration of individual drug use. Because of the differences between individuals and their responses to therapy, this task is difficult and ultimately relies on a clinician's judgement to be performed. In situations where newly developed drugs are used for treatment such as clinical trial testing, the task becomes even more challenging since it is difficult to judge how an individual will respond to a treatment with very little experience using the drug.

The Human Immunodeficiency Virus (HIV) is a retrovirus that potentially causes Acquired Immune Deficiency Syndrome (AIDS). Having been identified 29 years ago, HIV remains a worldwide threat after approximately 33 million people have been infected with the virus (Douce et al., 2012). Once an HIV particle comes into contact with and includes itself into certain cells of the immune system such as $CD4^+$ T-lymphocytes, a series of intracellular events occur that result in rapid viral reproduction, death of infected immune cells and eventually a loss of immunity. To date, the only effective way to treat HIV-infected individuals makes use of a combination of anti-HIV drugs in the form of Highly Active Antiretroviral Therapy (HAART) (Adams et al., 2004). These drugs operate by targeting various phases of the viral life cycle in an attempt to prevent the virus from replicating. Because of advances made in these drug therapies since the introduction of HAART in 1996, many individuals have been able to maintain viral loads below detectable limits ($< 40$ copies/ml) and sustain high T-lymphocyte counts for extended periods of time. Combination therapy is based on the premise that all virions are not homogenous (Abadi et al., 2006). Instead, variants of the original virion present in different proportions may exist within an HIV sufferer. Each of these variants has its own level of fitness that determines its chances of survival. While certain strains may be resistant to a particular drug, it is less likely for a strain to be resistant to a combination of two or more drugs. There are currently approximately 24 FDA-approved anti-HIV drugs available (Smith, 2013). These drugs operate specifically by

targeting the stages of viral entry, reverse transcription, integration of the viral genome into the host cell and viral protein formation and maturation in the virus life cycle. In particular, anti-HIV drugs may fall into one of five classes: (i) Non-nucleoside reverse transcriptase inhibitors (NNRTIs), (ii) nucleoside reverse transcriptase inhibitors (NRTIs), (iii) protease inhibitors (PIs), (iv) entry inhibitors or (v) integrase inhibitors (Stayley, 2012). Both categories of reverse transcriptase inhibitors prevent the conversion of viral RNA into DNA to prevent the virus from successfully integrating into the host's genome (Stayley, 2012). Protease Inhibitors interfere with the protease enzyme contained in an HIV particle; usually the protease enzyme dissects HIV proteins into smaller pieces that can be used to create new virus particles. PIs thus prevent proper cutting and structuring of these viral proteins (Stayley, 2012). In doing so, PIs effectively reduce the number of infectious virus particles released by a cell. Entry inhibitors prevent infection in cells of the immune system by preventing virus particles from binding to host cells. This binding process is crucial for the genetic material and enzymatic content of a virus to be inserted into an immune cell and cause infection. Integrase inhibitors interfere with the action of the integrase enzyme that enables viral DNA to be incorporated into the host's original genome (Stayley, 2012). Without this step, it is impossible for the virus to replicate.

Despite the fact that HAART has helped manage the virus for many HIV-infected individuals, it has not allowed for an outright cure. While eradication of HIV using HAART may not be possible because of the manner in which the virus establishes reservoirs within its host, there are a few other problems that have been encountered with HAART use. Perhaps the most significant of these problems are the issues of poor patient adherence to medication and the development of drug-resistant HIV strains in certain situations. In addition, when antiretrovirals are used poorly, patients can experience many negative side effects, such as gastrointestinal disturbances, hepatotoxicity and metabolic abnormalities (Abadi et al., 2006). As a result, it has become necessary to determine how to schedule these drugs according to how a patient responds to their treatment. To do so, a patient's individual needs and immune responses need to be taken into consideration. In a clinical trial setting, where newly developed anti-HIV drugs are tested, this is even more important since experience using the drug is limited.

A strategy for sequencing anti-HIV drugs that gained popularity during earlier years of HAART was the use of Structured Treatment Interruptions (STIs). During STIs, patients were cycled on and off different drug therapies (Bonhoeffer et al., 2000). It was initially thought that using STIs would prevent continued use of the same drugs and hence inhibit the formation of drug-resistant strains. In addition, it was thought that allowing patients periods of relief from treatment would elicit a stronger adaptive immune response that would prove more effective once treatment was re-initiated. The Strategies for Management of Antiretroviral Therapy (SMART) trial was conducted in 2001 to compare the use of continuous HAART and STIs amongst approximately 6 000 patients from 33 different countries worldwide (Lawrence & El-Sadr, 2006). Initially the trial was aimed at collecting follow-up patient data for eight years unless a patient progressed to AIDS, experienced very serious complications or died. Entry into the trial

was eventually stopped in 2006 after several safety concerns were raised. Overall, it was concluded that the use of STIs in HIV-infected individuals is inferior to continuous HAART as patients experienced more complications, had poorer immune responses and were more likely to progress to AIDS under STIs (Lawrence & El-Sadr, 2006).

A newer strategy for HAART has since emerged known as treatment simplification. Treatment simplification is targeted specifically at improving patient adherence to medication by either reducing the number of tablets a patient has to take or by attempting to mitigate the negative side effects experienced from the medications prescribed (Pozniak, 2007). The former requires suitable drug combinations to be determined in advance such that the drugs can be reformulated to reduce the number of pills a patient is given. The latter requires proper assessment of the side effects experienced by an individual to determine which drugs are suitable for use and which are not. The difficulty in combining different drugs into single tablets for a patient lies in the fact that not all combinations of drugs are suitable for all patients. The introduction of new anti-HIV drugs from advances in HAART research further complicates the process.

This research is aimed at determining suitable actions to take when treating an HIV-infected individual based on their treatment history and their immune responses under HAART. Specifically, we examine the use of reinforcement learning techniques to do so. Reinforcement learning is a machine learning paradigm for choosing the best sequence of actions in a system that changes over time (Sutton & Barto, 1998). In the context of treatment design, reinforcement learning enables us to determine the long term effect of a given treatment and hence determine an appropriate drug strategy over time. This is particularly important since a treatment that produces favourable outcomes from short term use is not necessarily guaranteed to do so in the long term. The hope is that using machine learning techniques can provide some insight as to what combinations of drugs are suitable for particular patients. In the context of clinical trials, this knowledge can be combined with a clinician's expertise to produce a judgement of how well a particular drug works. A similar approach to design HIV treatment strategies has been used by Ernst et al. (2006). Here, the authors propose the use of reinforcement learning specifically for computing STI control strategies for simulated HIV patient data. It may be possible to adapt the methods in this research to apply to treatment simplification.

The remainder of this thesis is structured as follows. Chapter 2 presents the biological background related to HIV. In particular, we examine the structure of an HI virion; this viral structure is fundamental to understanding the manner in which HIV is able to attack and weaken the immune system. We also present the detailed series of events that occur once HIV infects an individual as well as existing strategies for treating the virus. This includes the various classes of drugs that are currently in use under HAART. The chapter concludes with a discussion on attempts at developing a vaccine as a potential cure for HIV-infection and the implications this would have on the global pandemic. Until a successful vaccine is developed, improved HAART strategies are still necessary for managing the virus.

In Chapter 3, we present the reinforcement learning framework that forms the basis of this thesis. We specifically focus on a class of learning techniques called batch rein-

forcement learning methods that are suited to the problem of determining the suitable course of action to take when treating an HIV patient. We also provide examples of benchmark domains that have been used to compare the relative performances of each learning technique under consideration.

In Chapter 4, we provide a detailed description of the specific aim of this research and develop a series of research questions on which this work is focused. We also discuss the methodology followed to complete this research which includes a description of how we have formulated the HIV drug therapy problem as a Markov Decision Process.

We present, in condensed form, the main results obtained from applying the research methodology from Chapter 4 to various data sets in Chapter 5. These results are discussed with reference to the research questions posed. The overall outcomes of this research and potential future investigations are summarized and presented in Chapter 6.

# Chapter 2

# Biological Background

## 2.1 Introduction

In this chapter, we explore the biology of the HI virus and its effect on the immune system. Section 2.2 presents the basics of the immune system and how it works. We focus specifically on the two major responses in place for acquired immunity namely, humoral immunity and cell-mediated immunity. In Section 2.3, we discuss the structural details of HIV and its attack of the immune system, as well as differences between strains of the virus. Section 2.4 presents the specifics of the immune response to HIV. In particular, we examine the immunologic events occurring in the acute infection, clinical latency and AIDS stages of the disease. In Section 2.5, the current antiretrovirals used in treating HIV are introduced. These antiretrovirals are classified according to the stages of HIV infection they inhibit. Section 2.6 explores early treatment strategies and attempts at combatting HIV. Here, we focus on the introduction of HAART, and the use of treatment interruptions as a method for stimulating an immune response among HIV infected individuals. In Section 2.7 we examine a newer treatment simplification approach to HAART which aims to reduce medication costs and improve patient adherence. Finally, Section 2.8 concludes the chapter by discussing an entirely alternative approach to HAART in the form of vaccine design as an attempt to eradicate the virus.

## 2.2 The immune system

In order to understand the process by which HIV infects the body, it is important to have an understanding of the immune system. Any animal should be able to defend itself against viruses, pathogens and bacteria that it may encounter during the course of its life. For these reasons, two major defense mechanisms exist, namely *innate immunity* and *acquired immunity*.

### 2.2.1 The innate immune system

The innate immune system is non-specific as to which organisms to defend the body against and hence acts as its "first line of defense" against any invading pathogens

(Mayer, 2011). That is, innate defenses usually respond quickly to a large variety of micro-organisms, regardless of their specific nature or identity. Innate defenses are usually in place from the time of birth and do not have to be developed. The innate immune system consists of anatomic barriers formed by the skin and mucous membranes, as well as a group of chemical and cellular defenses that protect against those infectious agents that may bypass the external barriers (Mayer, 2011). Examples of such internal cellular and chemical defenses include phagocytic cells, dendritic cells, natural killer cells, inflammatory responses and antimicrobial proteins. Phagocytic cells such as macrophages are responsible for attaching to and destroying pathogens through enzymatic action (Campbell & Reece, 2005). They may also be responsible for presenting pathogens to members of the acquired immune system. Natural killer cells are responsible for the removal of virus-infected immune cells or cancer cells. This usually occurs using a process called apoptosis whereby the killer cells release chemicals that induce the death of an infected cell (Campbell & Reece, 2005). Antimicrobial proteins operate in the innate defense by either attacking microbes directly or by hampering their reproduction (Mayer, 2011).

Unlike the innate immune system, acquired immunity is adaptive and hence develops only after exposure to foreign substances. These defenses are highly specific and are hence slower in their response time in comparison to innate defenses (Campbell & Reece, 2005). The major components of acquired immunity include antibodies and lymphocytes although certain components of innate immunity may also function in the acquired immune system. Next, we examine the details of acquired immunity.

### 2.2.2 The acquired immune system

When the first line of defense fails, it becomes necessary for the acquired immune system to take control and eliminate any existing pathogens. Adaptive or acquired immunity is a defense mechanism that is based on specific cellular targeting. While it is slower in its response time than innate immunity, the adaptive response is more successful because of its precision (Fisher, 2011). Specificity of acquired defenses is made possible by white blood cells in the immune system. There are many different kinds of white blood cells each with their own associated functions. The most important group of white blood cells are referred to as *lymphocytes* and are responsible for recognizing and eliminating any foreign substances or *antigens* that the body may encounter. This is done using a range of distinct receptors. Lymphocytes originate from stem cells in the bone marrow and may relocate to other parts of the body where they differentiate and mature (Fisher, 2011). There are three major groups of lymphocytes namely *B-lymphocytes* (B-cells), *T-lymphocytes* (T-cells) and *natural killer cells*[1]. T-lymphocytes may further be classified as either *helper T-cells* or *cytotoxic T-cells*. Helper T-cells may also be referred to as $CD4^+$ cells since they express the CD4 protein on their cell surface. Similarly, cytotoxic T-cells may also be referred to as $CD8^+$ cells.

B-cells remain in the bone marrow until they are mature. Upon maturation, they

[1]Natural killer cells also play a role in the innate immune system as discussed in Section 2.2.1.

circulate through the bloodstream in search of antigens that they can recognize and interact with. Once a B-cell attaches to an antigen, it begins replicating. The newly formed B-cells can then differentiate into specialized B-cells known as either *plasma cells* or *memory B-cells*. Plasma cells are responsible for the secretion of antibodies that neutralize and defend against any pathogens that may be present in extracellular fluid (Abbas & Lichtman, 2009). Memory B-cells are long-lived B-cells which may survive for several years after an infection (Fisher, 2011). These cells are responsible for maintaining a record of which antigens the body has previously encountered. In doing so, they are able to provide faster and more effective action against those antigens, should they be re-encountered (Kumar et al., 2007). T-cells mature in the thymus into *effector cells* before they circulate the lymph and blood in search of antigens (Abbas & Lichtman, 2009). Effector cells can either assist other cells in counteracting antigens or can directly kill infected cells, depending on their type (Fisher, 2011). Upon encountering an antigen, helper T-cells inform other cells such as macrophages, to assist in eliminating the pathogens. Cytotoxic T-cells are specialized to secrete substances that destroy the cells to which they have attached. This is particularly useful and important for killing those cells that have been infected by a virus to prevent it from spreading further. Like B-cells, certain helper and cytotoxic T-cells may differentiate into long-lived T-cells that are able to respond rapidly to any secondary encounter with an antigen.

The acquired immune system may be divided into two parts: *humoral immunity* and *cell-mediated immunity*. Humoral immunity is governed by B-cells (with assistance from T-cells) and involves managing infectious agents that may be present in the blood or tissues of the body (Fisher, 2011). Cell-mediated immunity is accomplished by T-cells. Together, both parts of the immune system are able to provide a suitable immune response to almost any antigen encountered. We examine each of these reactions in turn.

### The humoral immune response

The humoral immune response begins with the activation of a B-cell. There are two ways in which a B-cell may be activated namely *T-cell independent activation* and *T-cell dependent activation* (Fisher, 2011). During T-cell independent activation, antigens interact with antigen receptor molecules on B-cells and activate the B-cells (Kumar et al., 2007). This usually occurs for sugar or fat-based antigens. However, certain protein antigens may not be able to bind to antigen receptors on B-cells. In these cases, it may be necessary for helper T-cells to assist. Here, the B-cells typically ingest the protein antigens, break them down and present the broken down peptides to the helper T-cells to recognize. The helper T-cells then secrete chemical cytokines which assist in activating the B-cells to elicit a suitable immune response (Kumar et al., 2007).

The activation of a B-cell is known as *clonal selection*. That is, once an antigen activates a B-cell, it also activates all the B-cells that are capable of recognizing the same antigen (known as a "clone") (Fisher, 2011). Once the cells of a clone are activated, they begin to proliferate by dividing. The division of activated cells is necessary to keep

up with the rate at which the antigen proliferates. This process is referred to as *clonal expansion* (Fisher, 2011).

Once the B-cell clones expand, the B-cells differentiate into plasma cells that produce antibodies to combat the antigen. Each plasma cell secretes antibodies that are specific to the antigen that was initially recognized during the activation step (Kumar et al., 2007). These antibodies are capable of binding to the antigen which in turn, prevents it from attaching to and infecting cells. Alternatively, the antibodies may mark a particular antigen for destruction by members of the innate immune system such as macrophages (Fisher, 2011). In this way, the antibodies are able to neutralize the effect of any particular antigen. As the humoral immune response declines, some plasma cells undergo 'cell-suicide' or apoptosis and die.

While the majority of the plasma cells die, certain plasma cells migrate to the bone marrow where they continue to secrete antibodies for many years (Fisher, 2011). That is, certain cells develop into memory B-cells following an infection. Should the body be re-exposed to such an antigen, the memory cells will be able to respond more effectively than naive lymphocytes.

### Cell-mediated immune response

The second type of adaptive immune response is governed by T-cells. Unlike B-cells that are able to recognize antigens with various chemical structures, T-cells are only able to recognize particular fragments of peptides (Fisher, 2011). These peptide fragments are exhibited by specialized molecules known as *major histocompatibility complex* (MHC) molecules (Kumar et al., 2007). MHC is a protein-complex that is expressed on the surfaces of cells of the body; that is, MHC molecules serve as identifiers for cells belonging to the body. This is particularly important since it enables the body to distinguish between 'self' and 'non-self', i.e. which substances belong to the body and which do not. In general, MHC molecules function by exhibiting the antigens present in different cells (Tamarkin, 2011). That is, when a cell comes into contact with a particular antigen, it decides what course of action to take. The MHC molecules of the surface of that cell in turn exhibit properties of the antigen on which the cell acts.

Consider the simple example of a virus invading cells of the stomach, as in gastroenteritis. Upon invasion, MHC molecules on the cells of the stomach display pieces of the invading virus at the cell surfaces so that they may be recognized and acted on by T-cells. This provides a scheme for labeling which cells should be acted on.

MHC molecules may be classified as either MHC I or MHC II molecules. MHC I molecules are located on all the nucleated cells of the body and are responsible for exhibiting those peptide antigens found within the cytoplasm of cells. MHC I molecules are recognized by receptors on $CD8^+$ cells (Fisher, 2011). In contrast, MHC II molecules are located on antigen-presenting cells (APCs) and lymphocytes. These molecules are responsible for displaying antigens from within the vesicles of the cells and are recognized by receptors on $CD4^+$ cells (Fisher, 2011).

There are five major steps in cell-mediated immunity: *antigen presentation, anti-*

*gen recognition and binding, differentiation and co-stimulation, antigen destruction and elimination*, and *memory* (Tamarkin, 2011). We examine each of these steps in turn.

1. **Antigen presentation:**
   When a cell is recognized as 'non-self' because of its lack of MHC, immediate action is required. This action is usually brought about by the innate immune system through APCs such as macrophages or dendritic cells. Macrophages phagocytose or ingest these foreign substances to eliminate any immediate threat they may pose to the body. During this process, MHC molecules on the macrophage surfaces exhibit fragments of the ingested antigen. The display of an antigen at the surface of a cell is known as the "presentation" of an antigen (Tamarkin, 2011). It enables the antigen to be seen safely and alerts the adaptive immune system to eliminate any copies of the antigen that may be present.

2. **Antigen recognition and binding:**
   APCs travel to the lymphoid tissue and exhibit antigens using MHC molecules on their cell surfaces. T-cells with receptors that are specific to an antigen come into contact with these APCs, bind with them and are activated (Fisher, 2011).

3. **Differentiation and co-stimulation:**
   Once a clone of T-cells is activated, it expands by secreting a number of factors. These factors are necessary for the growth, proliferation and differentiation of the cells in the clone (Fisher, 2011). Certain members of the T-cell clone differentiate into effector cells that release cytokines that perform different functions. The effect of these cytokines is many-fold: cytokine secretion may increase phagocytotic activity, encourage T-cell proliferation and differentiation and stimulate further cytokine secretion (Tamarkin, 2011).

4. **Antigen destruction and elimination:**
   The manner in which an antigen is destroyed is largely dependent on the kind of cytokines released during co-stimulation. The kind of cytokines that are secreted is in turn dependent on the kind of T-cells that have been produced following differentiation. In general, there are 4 subsets of T-cells that may be produced during differentiation: Th-1, Th-2, Th-17[2] and cytotoxic T-cells (Fisher, 2011). Th-1 and Th-17 cells secrete factors that activate B-cells and stimulate macrophages whereas Th-2 cells secrete special cytokines called interleukins (Kumar et al., 2007). These interleukins are primarily responsible for antibody production and activation of other white blood cells such as eosinophils. Eosinophils in turn, are capable of destroying pathogens non-specifically. Cytotoxic T-cells produce a chemical called perforin that destroys other T-cells that have already been infected by a pathogen by inducing apoptosis (Tamarkin, 2011). In doing so, a given antigen or antigen-infected cells may be destroyed and removed.

[2]Th-1, Th-2 and Th-17 are all types of helper T-cells.

5. **Memory:**
   Activation of T-cells stimulates the production of memory T-cells. These memory T-cells, like memory B-cells, have the ability to survive for many years after an infection and are capable of responding more effectively to a secondary encounter of an antigen (Fisher, 2011).

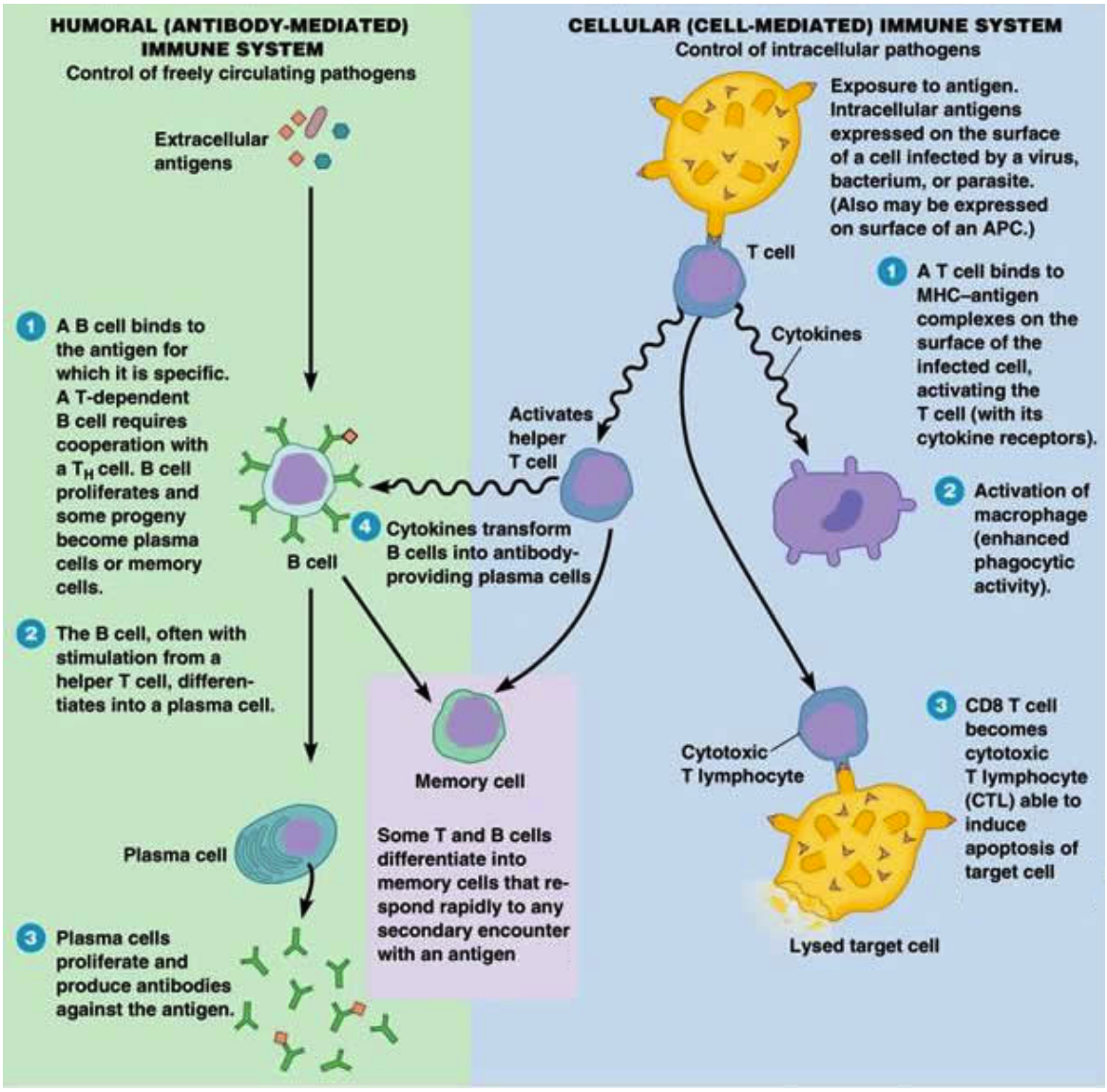



Figure 2.1: Summary of humoral and cell-mediated immune responses (Carter, 2011).

Figure 2.1 presents a summary and comparison of both the humoral and cell-mediated immune responses. The key stages of each response are illustrated.

Having examined the basic functioning of the immune system, we now explore the structural details of HIV and the pathogenesis of the virus. In particular, the rest of the

chapter presents some insights as to how the virus brings about changes in the human body ultimately to override its defense system. We also look at existing treatment strategies being implemented to control the spread of the virus.

## 2.3 The Human Immunodeficiency Virus

The Human Immunodeficiency Virus (HIV) is a lentivirus belonging to the group of viruses known as *retroviruses*. Retroviruses are those viruses that characteristically contain ribonucleic acid (RNA) as their genomic content. This is enclosed in a protein capsid and lipid envelope (Mann & Ward, 2006).

### 2.3.1 Viral structure of HIV

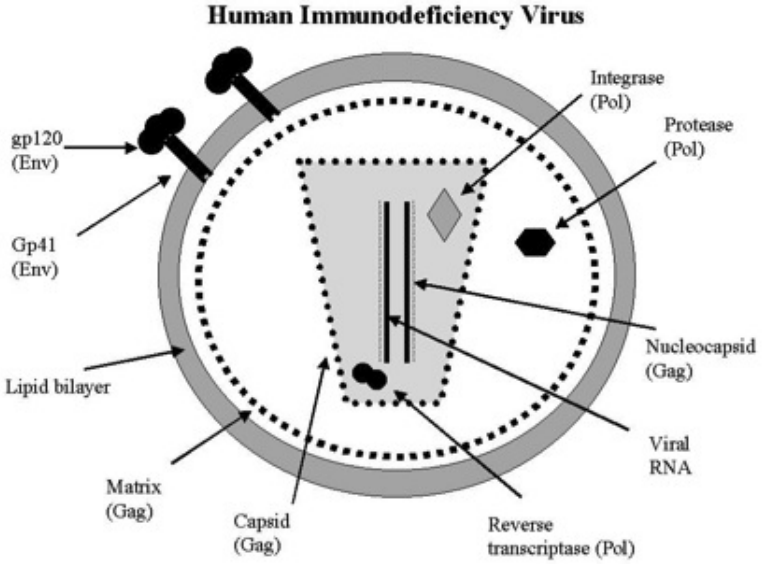



Figure 2.2: Structure of the HI virion (Mann & Ward, 2006).

The HI virus is roughly spherical in shape and contains two copies of positive single-stranded RNA. This RNA encodes for the virus's nine genes. Differences in strains of HIV may be attributed to differences in the genetic sequences encoded in the RNA. The RNA is enclosed in a conical shaped capsid composed of a number of viral proteins (Mann & Ward, 2006). The RNA is, in turn, tightly bound to a number of nucleocapsid proteins and enzymes that are necessary for the ultimate development of the virion. These enzymes include reverse transcriptase, proteases, ribonuclease and integrase (Chinen & Shearer, 2002). The capsid of the virion particle is surrounded by a matrix of other viral proteins to ensure its integrity. Finally, a viral envelope composed of two layers of phospholipids or fat molecules surrounds the entire virion. A number

of glycoproteins, including gp120 and gp41, are embedded in this viral envelope (Chinen & Shearer, 2002). These glycoproteins are critical for the correct fusion with, and attachment to, target cells during the infectious cycle.

Figure 2.2 shows a diagram of the structure of an HI virion (Mann & Ward, 2006). The bilayered viral envelope and glycoproteins on its surface are visible. The genetic content and viral enzymes are also included.

### 2.3.2 The HIV replication cycle

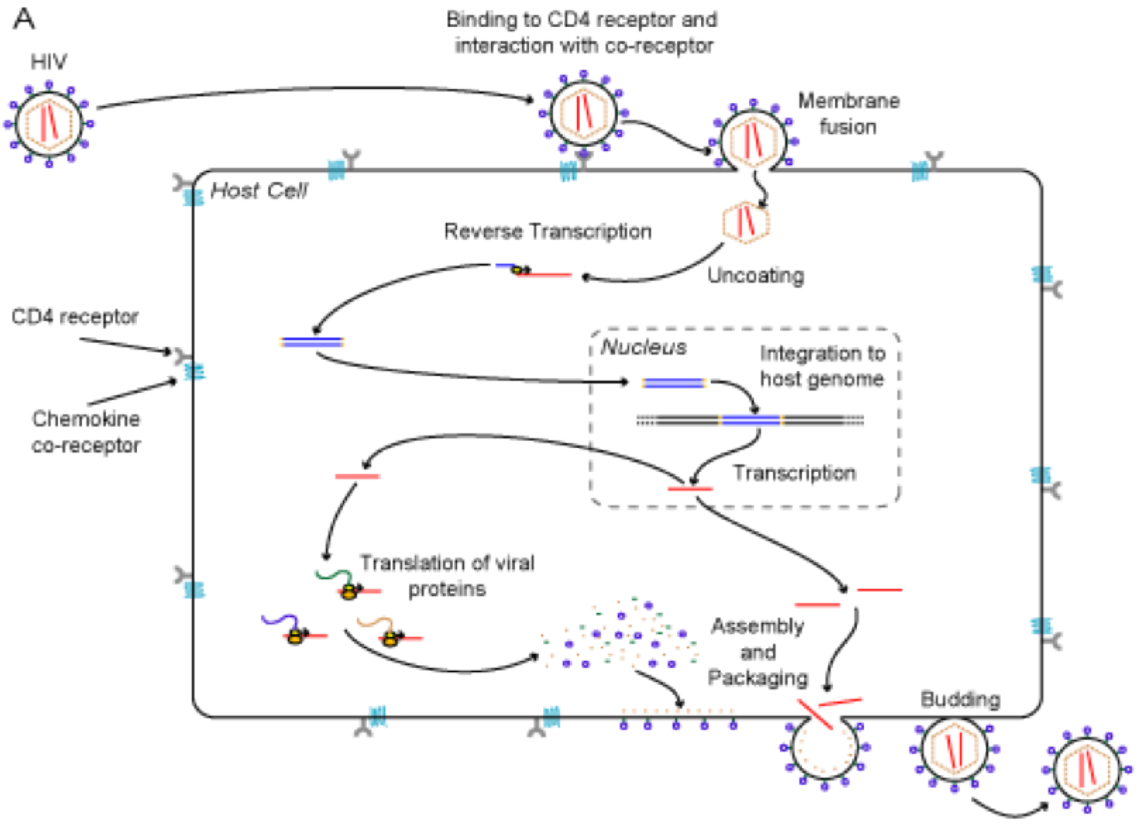



Figure 2.3: The HIV replication cycle (Figure modified from Archer (2008)).

The HIV replication cycle consists of 6 different phases: *binding and fusion, reverse transcription, integration, transcription, assembly* and *budding*. We examine each of these stages in turn.

1. **Binding and fusion:**
   HIV commences its life cycle through recognition of the viral glycoprotein gp120 with the CD4 cell surface molecule (Chinen & Shearer, 2002). That is, HIV begins its life cycle with CD4 receptors on the surface of a $CD4^+$ T-lymphocyte

identifying the gp120 glycoprotein on the HI virus. Interaction of the proteins causes the gp41 viral protein to interact with other molecules on the host's plasma membrane which in turn induces the viral envelope to fuse with the host cell membrane (Chinen & Shearer, 2002). Following fusion, the virus releases its capsid into the host cell. This includes all its genetic material, reverse transcriptase, integrase and the viral proteins that initially comprised the virus.

2. **Reverse transcription:**
   Entry of the capsid into the host cell results in the immediate release of the viral genome and proteins into the cytoplasm. Reverse transcriptase converts the single-stranded HIV RNA into complementary DNA (cDNA). In general, this process integrates an incorrect nucleotide every 1 500 to 4 000 bases (Chinen & Shearer, 2002). This rapid rate of mutation may explain why the virus is more capable of surviving and developing drug-resistant strains.

3. **Integration:**
   Following the conversion of HIV RNA to cDNA, the viral cDNA is transported across the nucleus where the HIV enzyme integrase masks the cDNA within the host cell's DNA. This enables the viral cDNA to integrate randomly into the host genome. The newly integrated HIV DNA is called a *provirus*. In this form, the provirus may remain inactive for a number of years, and may or may not produce new copies of HIV.

4. **Transcription:**
   After integration, certain transcription factors are able to trigger the host cell to activate viral gene transcription (Chinen & Shearer, 2002). This is done using the enzyme RNA polymerase to create multiple copies of the HIV genomic material. In addition, a number of shorter strands of messenger RNA (mRNA) are produced. The mRNA transcripts provide a blueprint for the HIV proteins that are to be produced.

5. **Assembly:**
   The components of the viral core are initially translated into pre-proteins each consisting of a long chain of peptides. HIV protease mediates the cleavage of these pre-proteins into smaller individual protein particles (Chinen & Shearer, 2002). These proteins move to the cell surface where they combine with the viral RNA to form a new immature virus particle.

6. **Budding:**
   The newly assembled virus particle is pushed out of the host cell. This process is known as budding. At this stage, the virus merges with part of the cell's outer envelope. The cell envelope is embedded with glycoproteins. By including these glycoproteins into the new virus particle, the virus is able to bind with and infect other cells.

Figure 2.3 is taken from Archer (2008). Here, the HIV cycle within a host cell is indicated by the black arrows. The red and blue lines represent viral RNA and reverse-transcribed cDNA respectively. Each of the phases of the viral replication cycle are shown.

This description is a gross simplification. The interested reader should refer to Cann & Karn (1989) for a more detailed explanation.

### 2.3.3 Subtypes and strains of HIV

One of the characteristics of HIV that makes it particularly difficult to treat is its ability to mutate. These mutations mean that the virus is highly variable and hence different strains of HIV may exist even within an infected individual. There are two phylogenetically distinct groups of HIV known as HIV-1 and HIV-2 (Archer, 2008). Both forms of HIV are thought to originate from non-human primates: HIV-1 has been linked to chimpanzees, whereas HIV-2 exhibits close similarity to viruses found in sooty mangabeys *(Cercocebus atys)*, a monkey species in west Africa. (Lihana et al., 2012). Both groups of HIV have the same means of transmission and may result in AIDS; however there are certain differences: HIV-2 is less easily transmitted, results in a slower decline of T-cells and has a better clinical forecast (Sousa et al., 2002). Within each group, different viral subtypes and strains may exist. The subtypes may be associated with varying degrees of transmissibility and virulence (Chinen & Shearer, 2002). Strains of HIV-1 may be classified into three types: Majority (M), Outliers (O) or New (N) (Archer, 2008).

Group M accounts for the majority of the global HIV pandemic and consists of at least ten genetically distinct subtypes or *clades* of HIV-1 (Archer, 2008). These are clades A through to K. Subtypes A, C, D and E predominate the developing world; clade B is more common in Europe, America and Australia; clades H, K and J are limited to parts of central Africa, west Africa and central America respectively (Chinen & Shearer, 2002). Many of the new strains do not have a high chance of survival individually but rather as circulating recombinant forms (CRFs)[3]. These CRFs have arisen as a result of genetic recombination between HIV strains in a particular host (Archer, 2008). The overall outcome is a new hybrid virus that is capable of replicating and invading more cells. It is this rapid ability of HIV-1 to recombine, mutate and generate extensive diversity that results in continuous infection within a host in spite of the actions of the immune system.

There are currently 8 known subtypes of HIV-2 (Santiago et al., 2005). These clades are A through to H. Of these, only subtypes A and B are widespread. Clade A is common in west Africa as well as Angola and parts of Brazil. Clade B is almost entirely confined to west Africa. Subtypes C and D, E and F, and G and H are very rare and have been found in parts of Liberia, Sierra Leone and the Ivory Coast respectively (Santiago et al., 2005).

[3]Subtype I is a CRF that is a recombinant of subtypes A, G, H and K. As a result, it may not necessarily be viewed as a subtype on its own.

For purposes of this research, we focus entirely on the treatment of HIV-1. This is largely because of its prevalence in South Africa and the data resources that we have access to.

## 2.4 The immune response to HIV

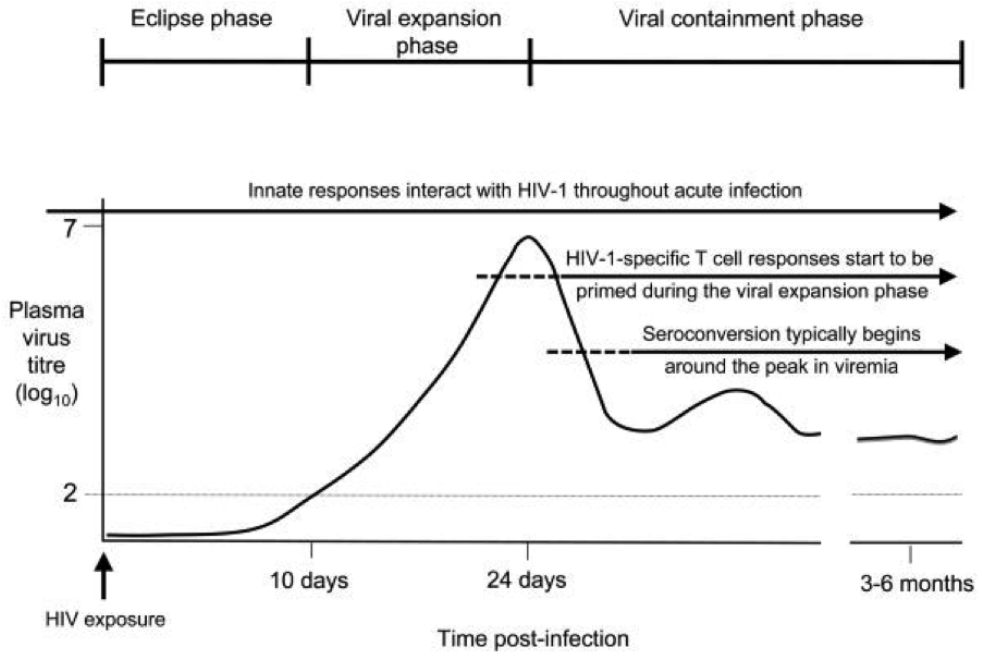



Figure 2.4: Phases of acute HIV infection (Borrow, 2011).

Having examined basic immune functionality, we now examine the impact of HIV infection on the immune system. The clinical course of HIV infection consists of three stages namely, (i) acute/primary infection, (ii) clinical latency and (iii) AIDS (Pantaleo & Fauci, 1996). Primary infection refers to the initial period of infection during which viral RNA is detected by the immune system, and is characterized by a rapid rate of viral reproduction; it ends once HIV antibodies begin developing several weeks after infection (Mogensen et al., 2010). It is comprised of three phases: (i) the eclipse phase, (ii) the viral expansion phase and (iii) the viral containment phase (McMichael et al., 2010). The eclipse phase is a short period of time before viral RNA is detectable in the blood plasma (McMichael et al., 2010). Typically, acute infection is characterized by a rapid rate of viral reproduction. This occurs during the viral expansion phase. Because of this, $CD4^+$ counts can fall rapidly during this phase. Ultimately, the overall immune response enables the virus level to be brought down to *viral set point* during which the virus stabilizes and its rate of reproduction settles (Weber, 2001). At this point, the $CD4^+$ count may begin to increase. This is the viral containment phase (Borrow, 2011).

Figure 2.4 is taken from Borrow (2011). It presents a typical example of a patient's

viral load during the acute stage of infection. The peak viraemia is shown during the viral expansion phase.

Clinical latency refers to the stage where despite being active, HIV reproduces at a very low level. As a result, an HIV-infected individual may present few or no HIV-related symptoms for many years during this phase. The clinical latency phase typically has a variable length and depends largely on an individual's immune responses and medication used. Clinical latency may also be termed *chronic HIV infection* or *asymptomatic HIV infection*. Once the clinical latency phase reaches completion, it is common for the viral load to increase hence resulting in a depletion of $CD4^+$ cells. If an individual's $CD4^+$ count falls below 200 cells/$mm^3$, the individual is diagnosed with AIDS (Weber, 2001). At this point, an individual's immune system is severely compromised and becomes particularly vulnerable to opportunistic infections. Next, we examine the clinical course of HIV and the immunologic events associated with each stage of infection.

### 2.4.1 Innate immune response to primary HIV infection

The most basic innate defenses at the initial stages of HIV infection consist of epithelial tissue and the mucosal membrane at the site of HIV transmission (Paranjape, 2005). The mucosal epithelium is a physical barrier that provides protection against invading virus particles. In addition, the presence of certain receptors at the epithelial layer enables the early detection and recognition of molecular patterns specific to the virus (Sivro et al., 2010). This detection promotes cytokine secretion which, in turn, ensures the recruitment of other innate defenses such as dendritic cells and natural killer cells. Following the entry of HIV into the bloodstream, adaptive immune responses may also come into play. Dendritic cells, macrophages and other members of the innate immune system may serve as APCs to induce adaptive immunity (Tamarkin, 2011). Geijtenbeek et al. (2000) observe that dendritic cells also have the ability of transmitting the virus without being infected through a process called *transinfection*. This particular situation is an example of how HIV is able to use the immune defenses for its benefit. At this stage, it is also common for natural killer cell counts to be elevated and for natural killer cells to exhibit increased activity. This is largely because these cells have the ability to kill any infected T-cells from the adaptive immune system, should the need arise. As the acute phase of HIV infection progresses, dendritic cells are significantly reduced (McMichael et al., 2010). Possible reasons for this include migration of activated dendritic cells to the lymph nodes or dendritic apoptosis.

### 2.4.2 T-cell and antibody response to primary HIV infection

The acute phase of HIV infection is typically characterized by a rapid rate of viral replication and hence high viral load. As a result, HIV-specific cell-mediated responses can be detected at the extremely early stages of primary infection (Pantaleo & Fauci, 1996). That is, the acute phase may be dominated by large increases in the number of $CD8^+$ T-lymphocytes in order to control the spread of infection. A spike in the $CD8^+$ T-cell response usually causes the virus to undergo mutation making it more difficult

to control (McMichael et al., 2010). This response is usually accompanied by a marked decline in the number of circulating helper T-cells corresponding to the peak in viraemia (Paranjape, 2005). During the entire infection process, HIV depletes the immune system of memory T-cells however, the exact mechanism(s) by which HIV kills T-cells is controversial. It is thought that a combination of direct killing of infected cells and indirect killing of uninfected cells contributes to the weakening of the immune response. Indirect killing may be made possible by the participation of certain HIV viral proteins in apoptotic pathways. That is, certain HIV viral proteins may engage in activities that induce apoptosis in T-cells (Streeck & Nixon, 2010). In other instances, indirect T-cell destruction involves damaging the cell membrane through continual budding of the virus. This procedure is believed to increase cell permeability thereby promoting the entry of more pathogens into the cell, and ultimately contributing to cell death (Paranjape, 2005).

The initial spike in viraemia eventually ends with a decrease in viral RNA levels as a result of $CD8^+$ cells targeting the HIV. Once this occurs, the RNA levels establish a viral set point. It is widely believed that $CD8^+$ T-cells are critically important for the maintenance of this viral set point. Pantaleo & Fauci (1996) observe that patients with a stronger early cytotoxic T-cell response experience lower viral loads over longer periods resulting in slower progression of the virus.

### 2.4.3 Immunologic events during clinical latency and AIDS

Clinical latency during HIV is used to describe the generally asymptomatic stage of HIV infection during which viral replication slows down considerably. One possible explanation for the decline in the viral load during this phase is provided by Siliciano & Greene (2011): it is likely that during the clinical latency phase, certain actively infected $CD4^+$ T-lymphocytes survive long enough to revert back to their previously inactive state. In this state, any gene expression of the virus is terminated and is hence unaffected by any immune responses or antiretroviral drugs. At the end of the clinical latency phase, these T-cells can be reactivated and latency can be reversed (Siliciano & Greene, 2011). In this way, latency reservoirs enable viral copies to begin re-circulating. If this is true, HIV latency enables the virus to exploit the most critical aspect of immunity namely the memory in long-lived T-lymphocytes. The implications of this are tremendous for any attempts to eradicate the virus.

Innate immune activation during the chronic phase of HIV is thought to be a significant contributor to the destruction of the immune system (Sivro et al., 2010). Many innate responses that serve as protective mechanisms during acute infection can be harmful in the chronic phase where they may be 'too little too late'. A particular example of this is the secretion of a chemical called interferon by dendritic cells; these secretions have deleterious effects as they induce cell death in both infected and uninfected T-lymphocytes - an undesirable response at this stage, that ultimately leads to a state of immune deficiency (Boasso & Shearer, 2008).

As the immune system weakens an individual increasingly becomes at risk of devel-

oping opportunistic infections or cancers, resulting in the diagnosis of AIDS.

The explanation provided in this section has been simplified considerably. The reader should refer to Walker & McMichael (2012) and Swanstrom & Coffin (2012) for a more rigorous treatment of the immune response to HIV.

## 2.5 Drug therapy for management of HIV

Currently a variety of HIV treatments exist with over 20 FDA-approved anti-HIV drugs available for use. These drugs when combined are termed antiretrovirals (ARVs). Typically, ARV drugs function by inhibiting certain phases of the HIV life cycle in an attempt to stop replication and further spread. There are five main classes of ARVs: (i) Non-nucleoside reverse transcriptase inhibitors (NNRTIs), (ii) Nucleoside reverse transcriptase inhibitors (NRTIs), (iii) Protease inhibitors (PIs), (iv) Fusion inhibitors, and (v) Integrase inhibitors (Stayley, 2012). As attempts have also been made to reduce the number of tablets a patient has to take daily, several single tablet regimens exist that combine drugs from these classes together in a single dose.[4] We examine each of the drug classes in terms of their functionality.

### 2.5.1 Non-nucleoside reverse transcriptase inhibitors (NNRTIs)

| **Name** | **Drug Combination** | **Dosage** |
|---|---|---|
| Edurant | rilpivirine hydrochloride (RPV) | One 25mg tablet daily. |
| Intelence | etravirine (ETV) | One 200mg tablet daily, or one 100mg tablets twice daily. |
| Rescriptor | delaviridine (DLV) | Two 200mg tablets three times a day. |
| Sustiva | efavirenz (EFV) | One 600mg tablet daily, or three 200mg daily. |
| Viramune | nevirapine (NVP) | One 400 mg tablet once a day. |

Table 2.1: List of currently approved NNRTIs with their dosages (Smith, 2013).

Enzymes are typically specific in terms of the substrates they interact with and the reactions that they catalyze. Complementarity of the shape of the enzyme and the shape of the substrate is partly responsible for this specificity. NNRTIs operate by binding to the reverse transcriptase enzyme and inducing conformational changes in the structure of the enzyme (Stayley, 2012). This, in turn, affects the catalytic ability of the enzyme and inhibits the transcription of viral RNA. The implications are that

[4]While these single tablet regimens are designed to make HIV treatment easier, many patients in South Africa still do not have access to such medication.

viral RNA cannot be reverse transcribed into cDNA correctly, and integration of the virus into the host genome is not possible.

Table 2.1 lists the major NNRTIs as of 2013 and their daily dosages.

### 2.5.2 Nucleoside reverse transcriptase inhibitors (NRTIs)

| Name | Drug Combination | Dosage |
|---|---|---|
| Combivir | lamivudine/zidovudine (3TC/AZT) | One tablet (150mg lamivudine/300mg zidovudine), twice daily. |
| Emtriva | emtricitabine (FTC) | One 200mg tablet daily. |
| Epivir | lamivudine (3TC) | One 300mg tablet daily, or one 150mg twice daily. |
| Epzicom | abacavir/lamivudine (ABC/3TC) | One tablet (600mg abacavir/300mg lamivudine) daily |
| Retrovir | zidovudine (AZT) | One 300mg tablet, twice daily. |
| Trizivir | abacavir/lamivudine/ zidovudine (ABC/3TC/AZT) | One tablet (300mg abacavir/150mg lamivudine/300mg zidovudine), twice daily. |
| Truvada | emtricitabine/tenofovir disoproxil fumarate (FTC/tdf) | One 300mg tablet daily. |
| Videx EC | didanosine (ddl) | One 400mg capsule daily. |
| Viread | tenofovir disoproxil fumarate (tdf) | One 300mg tablet daily. |
| Zerit | stavudine (d4T) | One 30mg capsule, twice daily. |
| Ziagen | abacavir sulphate (ABC) | One 300mg tablet, twice daily. |

Table 2.2: List of currently approved NRTIs with their dosages (Smith, 2013).

NRTIs may also be referred to as nucleotide reverse transcriptase inhibitors, nucleotide analogues or nucleoside analogues which inhibit the reverse transcription of viral RNA into cDNA (Stayley, 2012). NRTIs do so by incorporating faulty nucleotides into the reverse transcription process. The implications are that the new DNA is not constructed correctly. In this way, the genetic content of HIV cannot integrate with the genetic material already in the host cell thereby preventing the cell from reproducing the virus. Generally nucleoside analogues and nucleotide analogues have small structural differences but perform in much the same way: nucleoside analogues need to undergo an extra phosphorylation step in order to be activated for use; nucleotide analogues do not need this step (Stayley, 2012).

Table 2.2 lists the major NRTIs as of 2013 and their dosages per day.

### 2.5.3 Protease inhibitors (PIs)

| Name | Drug Combination | Dosage |
|---|---|---|
| Aptivus | tipranavir (TPV) | Two 250mg capsules with two 100mg tablets ritonavir, twice daily. |
| Crixivan | indinavir (IDV) | Two 400mg capsules every eight hours, or two 400mg capsules with 100mg ritonavir twice a day. |
| Invirase | saquinavir (SQV) | Two 500mg tablets with 100mg ritonavir twice daily. |
| Kaletra | lopinavir/ritonavir (LPV/r) | Four tablets (200mg lopinavir/50mg ritonavir) once daily, or two (200mg lopinavir/50mg ritonavir), twice daily. |
| Lexiva | fosamprenavir calcium (FPV) | Two 700mg tablets with 100mg ritonavir daily. |
| Norvir | ritonavir (r) | 100-400mg dosed once or twice daily with another PI. |
| Prezista | darunavir (DRV) | One 800mg tablet or two 400mg tablets with 100mg ritonavir daily. |
| Reyataz | atazanavir sulphate (ATV) | One 300mg capsule and 100mg ritonavir daily. |
| Viracept | nelfinavir (NFV) | Two 625mg tablets, twice daily. |

Table 2.3: List of currently approved PIs with their dosages (Smith, 2013).

PIs prevent the process of viral replication by binding to viral proteases that are critical for the production of new virus particles during the budding phase. In particular, these drugs operate by binding selectively to HIV-1 protease thereby blocking the cleavage process of many of the protein precursors needed for the formation of new virions (Stayley, 2012). The majority of virions produced in the presence of PIs contain major defects and cannot cause further infection.

The major concern with use of PIs is their specificity in terms of their target. That is, PIs are very specific about the viral proteases they interact with. Because of this specificity, it is common for patients to develop drug-resistant strains of HIV that are no longer affected by PIs. In order to reduce this risk, PIs are most often used in conjunction

with other anti-HIV drugs, each targeting a different aspect of viral replication.

Table 2.3 provides a summary of the current PIs in use and their dosages.

### 2.5.4 Fusion inhibitors

Fusion inhibitors, also termed entry inhibitors, interfere with the ability of HIV to bind with, fuse and enter healthy host cells in the body. This is the key difference between entry inhibitors and other anti-HIV drugs that are active only after HIV has infected a cell (Stayley, 2012). The process by which fusion inhibitors prevent HIV from entering immune cells involves attaching to the gp41 and gp120 proteins at the surface of $CD4^+$ cells or proteins on the surface of the HIV virion. If entry inhibitors are fully functional, they can be successful in preventing the binding and entry of HIV to immune cells.

Patients who have developed resistance to other antiretroviral drugs could benefit from using fusion inhibitor drugs.

Table 2.4 provides a list of the current fusion inhibitor drugs and their dosages.

| **Name** | **Drug Combination** | **Dosage** |
|---|---|---|
| Fuzeon | enfuvirtide (ENF or T-20) | 90mg subcutaneous injection twice daily. |
| Selzentry | maraviroc (MVC) | 150mg, 300mg or 600mg twice daily, depending on other medication used. |

Table 2.4: List of currently approved fusion/entry inhibitors with their dosages (Smith, 2013).

### 2.5.5 Integrase inhibitors

In order for HIV to take control of a host cell's machinery to produce more virions, viral DNA must be incorporated with the cell's original DNA. This is the integration step of viral replication. Integrase inhibitors operate by inhibiting the action of the integrase enzyme during this phase of viral replication (Stayley, 2012). In doing so, integrase inhibitors may block the formation of a provirus.

Table 2.5 lists the integrase inhibitors in use in 2013 and their dosages.

| **Name** | **Drug Combination** | **Dosage** |
|---|---|---|
| Isentress | raltegravir (RAL) | One 400mg tablet twice daily. |
| elvitegravir | elvitegravir (EVG) | 150mg once a day with 150mg cobicistat. |
| dolutegravir | dolutegravir (DTG) | One 50mg tablet daily. |

Table 2.5: List of currently approved integrase inhibitors with their dosages (Smith, 2013).

Having examined the various kinds of drugs available for treatment of HIV, we now focus on previous attempts at combating the virus and the outcomes of these approaches.

## 2.6 Initial treatment strategies: HAART and STIs

Conventional treatment for HIV patients makes use of highly active anti-retroviral therapy (HAART) (Bartlett, 2006). This treatment consists of using a mixture of multiple drugs, typically from more than one class, and is hence commonly referred to as combination therapy. The overall aim of HAART is to help patients maintain functionality of the immune system, decrease their overall viral load, and reduce risk of those opportunistic infections that usually result in death. Combination therapy is based on the premise that all virions are not homogeneous (Adams et al., 2004). Different variants may be present in varying proportions. Each variant has its own corresponding fitness which determines its chances of survival. While certain strains may be resistant to a particular drug, it is less likely for a strain to be resistant to two or more drugs.

The advent of HAART can be dated to the 11th International Conference on AIDS at British Columbia, Vancouver, during July 1996 (Bartlett, 2006). Findings at this conference indicated that an HIV-infected individual produces on average 10 billion virions per day (Bartlett, 2006). This conference was followed by subsequent publications in The New England Journal of Medicine by Gulick et al. (1997) and Hammer et al. (1997) in which the benefits of using a cocktail of three drugs (also known as triple drug therapy) were highlighted. The concept of combination therapy has since been incorporated into most hospital treatment plans and has been of considerable importance in the progression of combatting HIV. Typical combinations of ARVs in HAART include using 2 NRTIs in conjunction with 1 PI or using 2 NRTIs with 1 NNRTI, although new combinations are emerging since the introduction of entry inhibitors (Gulick et al., 1997).

### 2.6.1 Rationale for intermittent therapy

Despite the general worldwide success in managing the virus through HAART, concerns about the side-effects and costs of continual drug therapy as well as the possibility of developing drug resistance led to the design of intermittent therapy as a possible means of combatting these concerns (Lawrence & El-Sadr, 2006). Structured Treatment Interruptions (STIs) were an experimental approach designed specifically with the goals of (i) boosting the immune response in both the acute and chronic phases of infection, (ii) improving adherence to treatment (ii) regaining drug-sensitive forms of the virus in patients that had previously experienced virologic failure and (iii) reducing drug-toxicity levels and mitigating the negative side effects of ARV therapy. In patients suffering acute HIV infection, interruptions were thought to stimulate innate immune responses to antigens; in patients with chronic HIV that had developed drug-resistant viral types, interruptions were thought to allow drug-sensitive viral types to reappear.

### 2.6.2 Assessing the efficacy of treatment interruption

Many studies have been conducted in order to investigate the impact of treatment interruption on managing HIV. The Stacatto trial was one such study in which 430 candidates received randomized treatment either in the form of continuous therapy or in the form of intermittent therapy (Lawrence & El-Sadr, 2006). The patients in the group receiving intermittent therapy were only given treatment if their $CD4^+$ count was below the 350 cells/$mm^3$ mark. The major results from this trial were that adverse side effects such as diarrhea, were experienced more frequently by the patients undergoing continuous therapy, and a 62% reduction in ARV costs was reported for patients using treatment interruption (Lawrence & El-Sadr, 2006). While the results of this study were encouraging for treatment interruption, the study gave no indication of the clinical efficacy of each procedure.

Another such study more focused on the value of each treatment strategy, was the ANRS 1260 Trivican trial (Lawrence & El-Sadr, 2006). 326 patients with $CD4^+$ counts over 350 cells/$mm^3$ received randomized ARV therapy in the form of either continuous therapy or one of two treatment interruption strategies: (i) $CD4^+$ count guided or (ii) time guided. In $CD4^+$ count guided therapy, patients stopped anti-HIV drugs when their T-lymphocyte counts were over 350 cells/$mm^3$ and re-started drug therapy when cell counts fell below 250 cells/$mm^3$ (Lawrence & El-Sadr, 2006). In time-guided therapy, patients were cycled on and off drug therapy every two months. Results from this trial indicated that patients under $CD4^+$ count guided intermittent therapy were particularly at risk of further disease progression.

### 2.6.3 Controversy following the SMART study

Perhaps the most significant study comparing intermittent therapy to continuous HAART, is the SMART trial. The SMART trial is the largest treatment interruption clinical trial to date (Lawrence & El-Sadr, 2006). Pre-requisites for entry into the trial were a $CD4^+$ count above 350 cells/$mm^3$. Approximately 6 000 patients from 33 countries and over 318 sites were recruited providing several years of followup data (Lawrence & El-Sadr, 2006). Each patient was randomly assigned a treatment strategy i.e. continuous therapy or treatment interruption. The aim of intermittent therapy was to keep candidates off ARVs until $CD4^+$ counts were below the 250 cells/$mm^3$ mark and then continue treatment until cell counts increased above 350 cells/$mm^3$. Patients were removed from the trial in certain special cases such as if HIV had progressed to AIDS, death or other serious complications. Comparisons concerning the side effects, complications, drug resistance, cost, patient adherence and overall clinical efficacy of each strategy were made (Lawrence & El-Sadr, 2006).

Results from the SMART study showed significant differences between both types of treatment particularly concerning death among the groups. A total of 117 AIDS or death occurrences were reported in the group of patients using STIs; 47 such events were reported in the continuous HAART group (Lawrence & El-Sadr, 2006). In addition, a lower incidence of cardiovascular, hepatic and renal complications was reported among

patients belonging to the continuous therapy group. Investigators from the SMART trial eventually deemed treatment interruption strategies as risky for patients suffering HIV with few benefits. Dr Wafaa El-Sadr from the SMART trial concluded: "Episodic use of antiretroviral therapy based on $CD4^+$ cell counts, as utilized in the SMART study design, is inferior to continuous antiretroviral therapy for the management of antiretroviral-experienced patients," (Lawrence & El-Sadr, 2006).

## 2.7 A treatment simplification approach to HAART

Assuring good patient adherence to HAART may require treatment regimens to be simplified. Among other techniques, this treatment simplification may be in the form of reducing the number of pills a patient needs to take, or reducing use of drugs from a particular class according to their relative toxicities (Pozniak, 2007). The overall benefits of simplifying HAART thus include improving adherence, mitigating negative side effects, and reducing the costs of medication. Currently, there are two popular methods of treatment simplification that have been investigated: (i) PI-sparing approaches and (ii) PI-boosting approaches (Pozniak, 2007). We examine both of these techniques in turn.

### 2.7.1 PI-sparing approaches for treatment simplification

The aim of PI-sparing approaches for treatment simplification is to sufficiently suppress the virus using fewer pills. PIs are known to have several side effects including gastrointestinal disturbances, lipohypertrophy, and cardiovascular problems. PI-sparing drug combinations hope to reduce such side effects (Pozniak, 2007). Since the introduction of NNRTIs, there have been a number of clinical trials in which PIs were substituted with this class of drug in the hope of reducing toxicity levels and improving quality of life for HIV infected individuals. One such trial used 498 PI-treated patients who were randomized to substitute their PI with one of two NNRTIs namely EFV, NVP or the NRTI ABC. It was observed that after 48 weeks viral loads were lower in patients using NNRTIs and NRTIs in place of PIs, but negative side effects were more frequent: 84% of patients using PI-free therapy maintained viral suppression in comparison with 73% of patients using PI-inclusive strategies (Pozniak, 2007).

### 2.7.2 PI-boosting approaches for treatment simplification

Reduced-pill regimens are being investigated as a means of maintaining viral suppression in HIV patients. Certain ARV drugs have been re-formulated for once-daily use (Pozniak, 2007). In particular, the NRTIs ddl, FTC, 3TC and tdf and boosted PIs, ATV and fosamprenavir, are currently used in this form. In addition, certain drugs have been coupled with others; examples of this include lopinavir which is now used only in conjunction with ritonavir. Use of a single boosted PI is thought to have the associated advantages of reducing lactic acidosis, mitochondrial malfunctioning and lipoatrophy among others (Pozniak, 2007).

Table 2.6 provides a list of the current single tablet regimens in use.

| Name | Drug Combination | Dosage |
|---|---|---|
| Atripla | efavirenz/emtricitabine/ tenofovir disoproxil fumarate (EFV/FTC/tdf) | One tablet (600mg efavirenz/ 200mg emtricitabine/300mg tenofovir df) daily. |
| Complera | rilpivirine/emtricitabine/ tenofovir disoproxil fumarate (RPV/FTC/tdf) | One tablet (25mg rilpivirine /200mg emtricitabine/300mg tenofovir df) daily. |
| Stribild | elvitegravir/cobicistat/ emtricitabine/tenofovir disoproxil fumarate (EVG/COBI/FTC/tdf) | One tablet (150mg elvitegravir/150mg cobicistat/200mg emtricitabine/300mg tenofovir DF) daily. |
| "572-Trii" | dolutegravir/abacavir/ lamivudine (DTG/ABC/3TC) | One tablet (50mg dolutegravir/600mg abacavir/300mg lamivudine) once a day for first time HIV therapy patients; two times a day for patients who have developed resistance to Isentress and elvitegravir. |
| cobicistat | COBI | 150mg daily. Cobicistat is not an HIV drug, but is used to increase the levels of elvitegravir and HIV protease inhibitors. |

Table 2.6: List of current single tablet regimens and the contents of each tablet.

### 2.7.3 Implications for HAART

While treatment simplification regimens for HAART aim to reduce pill burden in patients and mitigate negative side-effects, using such strategies has a number of implications.

1. Suitable drug combinations for a reduction in pill burden need to be determined.

2. Typically this would involve a series of clinical trials to be conducted to determine the safety of certain combinations.

3. Clinical trials are expensive to conduct and are usually conducted over a long period of time.

For these reasons, it may be necessary to couple clinical trials with other methods of predicting the effects of certain drug combinations on the immune system.

## 2.8 Exploring the alternatives: vaccine development

Despite advances made in developing antiretrovirals suited to suppressing HIV infection and preventing its progression to AIDS, an effective vaccine for eradicating the virus has not been established. Developing a successful HIV vaccine that is capable of inactivating the virus as a whole, is perhaps one of the most significant global health challenges to date. Scientists have attempted developing classic prophylactic vaccines, as used for polio and measles, to combat HIV infection. These vaccines make use of inactive or live attenuated virus particles in which the major viral genes contain deletions (Lai & Heeney, 2012). Both types of prophylactic vaccines have failed since inactive HI virus particles are poorly immunogenic and introducing live attenuated virus particles into a system has proven to increase chances of HIV progressing to AIDS (Lai & Heeney, 2012).

Unlike the polio virus or measles, HIV has two distinguishing features which have complicated the process of discovering a suitable vaccine. The first is that HIV uses a DNA intermediate, specifically a provirus, to establish itself and replicate (Lai & Heeney, 2012). In this form, the virus can integrate with a host's genome and remain inactive for years until an unknown series of events triggers the onset of viral replication. Once replication begins, the virus starts invading T-cells and other APCs. The most severe implication is that the virus establishes reservoirs which are used to continuously attack and invade more immune cells. The second feature that distinguishes HIV from similar viruses is its ability to mutate at a high frequency. The mutations produced are a direct result of a lack of proofreading mechanism incorporated into the reverse transcriptase enzyme. Traditionally, enzymes that catalyze such reactions have a built-in mechanism of correcting any erroneous nucleotide bases that are transcribed to prevent the occurrence of mutations; the HIV reverse transcriptase enzyme does not have this. The implications of this are significant: the cDNA that is transcribed contains many mutations resulting in altered viral protein production. This variability of HIV makes developing a protective vaccine extremely difficult since such a vaccine would have to be able to provide protection against all viral strains.

Newer strategies for vaccine development have differed according to the ways in which they seek to combat HIV. The majority of these strategies can be broadly classified into three categories. The first category is a set of vaccines that have aimed to utilize a class of antibodies termed *broadly neutralizing antibodies* to combat the spread of the virus. These antibodies, unlike traditional antibodies, are not specific in the antigens they can neutralize; instead they can neutralize a broad spectrum of viral variants by attaching to regions of antigens that do not change upon viral mutation. The discovery of such antibodies could prove extremely important for determining a suitable means of combatting the virus in the future. The second set of vaccines are aimed at using certain properties of T-lymphocytes to sustain an adequate immune response in situations where antigens are not capable of inducing broadly neutralizing antibody responses (Lai & Heeney, 2012). The third set of vaccines are based on using gene therapy to produce an immunization against the virus. Briefly, gene therapy steps

away from developing a T-cell or B-cell based vaccine for HIV-1. Instead, the focus is on targeting the virus using *intracellular immunization* (Lai & Heeney, 2012). The goal of intracellular immunization is to prevent viral replication by modifying target cells of the immune system to express antiviral genes. Gene therapy vaccination would involve collecting T-lymphocytes, genetically modifying each cell to include an antiviral gene, and injecting the infusion back into a HIV-infected individual (von Laer & Brandenburg, 2001).

Vaccine research is typically a lengthy process that is conducted in phases. At each phase of a clinical trial, the candidate vaccines are tested on human volunteers. Despite scientists exploring numerous strategies for HIV vaccine development, only three potential candidate vaccines have completed large-scale clinical trial testing to date. These are the AIDSVAX by VaxGen, STEP by Merck and the RV144 which combines two previous vaccines that have failed individually. The AIDSVAX was an experimental prophylactic HIV vaccine that was initially developed by Genentech in 1991 and tested later by VaxGen (Billich, 2001). At its core, the vaccine made use of proteins that are recombinant forms of the surface protein gp120 from two types of HIV. In particular, the primary goal of the AIDSVAX was to induce a broadly neutralizing antibody response which could potentially provide protection against infection (Billich, 2001). Clinical trial testing for the vaccine began in 1998 across the United States with 5403 participants, the majority of whom were homosexual men. Phase II of the trial started in 1999 in Thailand and reached completion in 2003. Results from the clinical trial showed that neither version of the vaccine succeeded in preventing HIV infection (Lai & Heeney, 2012). The STEP vaccine, also known as Merck's V520-023, was a T-cell based vaccine containing a weakened adenovirus such as a common cold virus, in addition to three HIV-1 subtype B genes (Sekaly, 2008). The vaccine was aimed at targeting cell-mediated responses. These responses are believed to be particularly important in the early stages of viral infection. It was thought that the addition of an adenovirus could elicit stronger cell-mediated responses which could in turn be used to eliminate HIV-infection. The vaccine was tested on 3 000 participants until September 2007 where use was discontinued following substantial evidence that the vaccine failed to provide protection against HIV and potentially increased chances of developing infection (Lai & Heeney, 2012). The closest attempt thus far at developing a suitable vaccine for HIV is the RV144 vaccine from a Thai trial in 2009. This vaccine was based on combining two previously failed vaccine attempts to form a new vaccine. In particular, a bird virus containing three HIV genes was combined with a modified version of the AIDSVAX vaccine in the hope of eliciting a boosted immune response (Rerks-Ngarm et al., 2009). The vaccine was tested on 16 402 individuals making the trial the largest clinical trial for HIV vaccination to date (Rerks-Ngarm et al., 2009). It was initially thought that such a vaccine would be unsuccessful since both components of the vaccine had failed individually. However, results from the trial indicated otherwise: overall the vaccine combination proved to be somewhat effective and reduced the risk of HIV infection among participants by 31% (Rerks-Ngarm et al., 2009). While this figure is too small for a feasible vaccine, it is still significant.

The expansion of the HIV pandemic has highlighted the need for a suitable HIV vaccine to be developed in the future. A number of scientific advances have been made in the field of HIV vaccine development in recent years. The majority of these advances have been based on having a better understanding of the viral structure of HIV and the human immune response to the virus. It is likely that future success in this field will come from a vaccine that is able to elicit an immune response consisting of both neutralizing antibodies that target regions of the virus that are conserved during mutation, as well as cytotoxic T-cells that target a variety of antigens. Overall, progress in the field will require continued commitment of researchers as well as adequate government support.

## 2.9 Conclusion

This chapter provided an overview of the key role players in the immune system and how they interact during HIV interaction. The major components of the acquired immune system are particularly important for keeping the virus under control and are usually good indicators of a patient's state of health. These members of the acquired immune system may be used to model the HIV drug scheduling problem as a reinforcement learning task which we explore in Chapters 3 and 4. We also presented an overview of the different classes of drugs that are currently in use for HIV treatment and discussed treatment simplification as a strategy in place to reduce pill burden and improve adherence among HIV sufferers. Treatment simplification strategies require determining suitable drug combinations usually through a clinical trial basis. Coupling these studies with the batch reinforcement learning techniques that we examine in Chapter 3 could provide a reasonable basis for solving the drug scheduling problem. The next chapter discusses the reinforcement learning paradigm which has been used to model the HIV drug scheduling task in this research.

# Chapter 3

# Reinforcement Learning Background

## 3.1 Introduction

In this chapter, we discuss the reinforcement learning framework that forms the basis of this research. Section 3.2 presents the specifics of the reinforcement learning paradigm in terms of how an agent and environment interact to achieve a specific goal. In particular, we examine how a reinforcement learning problem can be formulated in terms of a Markov decision process and introduce the concept of a value function to assess the overall performance of the agent relative to the environment at a particular time. Section 3.3 explores how basic value iteration may be performed when the dynamics of agent-environment interaction are available. In Section 3.4 we extend this to the model-free case by examining the online $Q$-learning algorithm that forms the basis of the batch reinforcement learning techniques we use throughout this research. The subsequent Sections 3.5 - 3.9 discuss the need for batch reinforcement learning methods within the context of traditional reinforcement learning and present the various techniques being used in this research namely fitted $Q$-iteration, neural fitted $Q$-iteration and least squares policy iteration. In particular, we focus on the specifics of the function approximation techniques employed by each of the algorithms by discussing extremely randomized trees, the multilayer perceptron and least squares methods respectively. We conclude the chapter by presenting two popular benchmark domains, other than the HIV domain, that we will be using for experimentation.

## 3.2 The reinforcement learning paradigm

The material in this section is based on Chapters 2 and 3 of Sutton & Barto (1998).

*Reinforcement learning* (RL) is a machine learning paradigm in which a decision maker or *agent* interacts with an *environment* to learn a particular task. Typically, agent-environment interaction involves three signals: a *state* signal describing the current situation of the environment, an *action* signal that allows the agent to influence

the environment, and a *reward* signal produced by the environment to provide the agent with feedback on its immediate performance or to evaluate the quality of taking a particular action at a certain time (Buşoniu et al., 2010). The aim of the agent is to make decisions optimally by determining those actions that maximize accumulated future rewards.

Agent-environment interaction may be broken down over a sequence of discrete time steps, $t = 0, 1, 2, \ldots$. At each of these points in time, the agent receives a summary of the conditions of the environment in the form of its state at time $t$, $s_t$. Here $s_t \in \mathcal{S}$ where $\mathcal{S}$ is the collection of all possible states. Based on the environment's situation, the agent selects an action at time $t$, $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of all available actions. Applying a particular action causes the environment to transition to a new state $s_{t+1}$. The agent receives a scalar reward $r_{t+1} \in \mathbb{R}$ assessing the quality of the transition to the new state. Negative rewards may be viewed as penalties for making a poor choice of action at a particular time. Once the agent receives information about the new conditions of the environment, the whole cycle of agent-environment interaction repeats.

The behaviour of an agent is governed by the sequence of actions it takes following each state it finds itself in. This is known as the agent's *policy*, $\pi : \mathcal{S} \to \mathcal{A}$. Alternatively, in cases when policies are non-deterministic, the agent's policy may also be viewed as a mapping of states to probabilities of actions. The behaviour of the environment may be described in terms of its dynamics which dictate how the state changes as a result of the agent's choice of actions. In a deterministic setting, taking a given action from a given state always results in the same next state; in a stochastic setting, the next state may vary (Buşoniu et al., 2010). Together, the states, actions, rewards and environment dynamics constitute a Markov Decision Process.

### 3.2.1 Markov decision processes

The interaction between an agent and its environment can be modelled in terms of a *Markov Decision Process* (MDP). Formally, an MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the set of all possible environmental states and $\mathcal{A}$ is the set of actions the agent can take. $\mathcal{P}$ is the set of transition probabilities defining the likelihood of ending up in each possible next state, $s'$, having taken any action, $a$, from any state, $s$. $\mathcal{R}$ is the set of numerical rewards received after each transition. Given a particular state, $s$, and an action $a$, the probability of moving to a state, $s'$, is given by

$$P(s, a, s') = \mathbb{P}\left[s_{t+1} = s' | s_t = s; a_t = a\right]. \tag{3.2.1}$$

Similarly, the expected reward for taking an action $a$ from a state $s$ and ending up in a state $s'$ is given by,

$$R(s, a, s') = \mathbb{E}\left[r_{t+1} | s_t = s; a_t = a; s_{t+1} = s'\right]. \tag{3.2.2}$$

Here, the set $\mathcal{P}$ consists of all transition probabilities $P(s, a, s')$ and the set $\mathcal{R}$ consists of all numerical rewards $R(s, a, s')$.

### 3.2.2 The return function

The agent's goal is to maximize the accumulated future rewards. The *return* function, or simply the return, $R_t$, is a long-term measure of rewards. We can distinguish between *finite-horizon* and *infinite-horizon* models. In a finite-horizon model, the return is calculated over a finite number of time steps. The following equation gives the return over a finite-horizon:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_{t+K-1}. \tag{3.2.3}$$

Here, $K$ is the number of steps before the terminal state.

In certain complex tasks where the sequence of actions taken by an agent is long or infinite, immediate rewards may be valued more than those in the future. In these cases, a discount factor $\gamma \in (0, 1)$ is introduced. This discount factor is a measure of how much foresight the agent has in considering its rewards (Buşoniu et al., 2010). The return may then be expressed as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{3.2.4}$$

Note that we need $\gamma < 1$ to ensure that the infinite sum converges. In what follows, we shall use the infinite-horizon model for the return.

### 3.2.3 The value function

It is possible for some states of the environment to be more advantageous than others. When following a policy $\pi$, we can estimate how good it is to be in a certain state in terms of a *value function*. That is, we can value a state under a particular policy by the expectation of future reinforcement or rewards an agent receives. Using this information, the value, $V^\pi : \mathcal{S} \to \mathbb{R}$, of a given state $s$ at an arbitrary time $t$, if an agent follows a policy $\pi$, is given by

$$V^\pi(s) = \mathbb{E}_\pi \left[ R_t \,\middle|\, s_t = s \right] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right]. \tag{3.2.5}$$

Here, the operator $\mathbb{E}_\pi[\cdot]$ represents the expected value given that the agent follows policy $\pi$.

A key property of the value function is that it satisfies a recursive relationship. Formally, this relationship may be expressed according to the *Bellman operator* under a policy $\pi$, $B^\pi(\cdot)$, as follows:

$$B^\pi(V^\pi) = V^\pi. \tag{3.2.6}$$

The existence of this recursive relationship means that we can rewrite the expectation in Equation 3.2.5 as a summation over states and hence express the value of a particular state in terms of the value of its successor states (Sutton & Barto, 1998)[1]:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} P(s, a, s') \left\{ R(s, a, s') + \gamma V^\pi(s') \right\}. \tag{3.2.7}$$

[1]The interested reader should refer to Sutton & Barto (1998) for this proof.

Here $\pi(s,a)$ is the probability of taking an action $a$ from the state $s$. Equation 3.2.7 is known as the *Bellman equation* for $V^\pi$. Intuitively, an agent can take a number of actions from a given state $s$, each of which may result in different subsequent states $s'$ and different rewards. Equation 3.2.7 averages the outcomes by weighting each according to its probability of occurring.

We can also define the *optimal value* for a state, $V^*(s)$ in terms of the *Bellman optimality equation*, i.e.

$$V^*(s) = \max_\pi V^\pi(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s') \left\{ R(s,a,s') + \gamma V^*(s') \right\}. \tag{3.2.8}$$

Here, determining the optimal value involves selecting the policy and hence actions, which maximize the value of a particular state.

The Bellman optimality equation is a central construct in RL algorithms. Given the optimal value function for a particular RL problem, we can derive the *optimal policy* by considering those actions that maximize the Bellman optimality equation. The optimal policy for a particular RL problem is

$$\pi^*(s) = \arg\max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s') \left\{ R(s,a,s') + \gamma V^*(s') \right\}, \tag{3.2.9}$$

where $\arg\max_{a\in\mathcal{A}}$ finds the action which maximizes the expression.

It is not always possible to compute an optimal policy directly from the Bellman optimality equation. In these cases, it is useful to estimate the value of a state-action pair $(s,a)$ in terms of a $Q$-function, $Q^\pi : \mathcal{S}\times\mathcal{A} \mapsto \mathbb{R}$. $Q^\pi(s,a)$ defines the expected long-term return of applying action $a$ when in state $s$ under a policy $\pi$. That is,

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[ R_t \,\middle|\, s_t = s; a_t = a \right] = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s; a_t = a \right]. \tag{3.2.10}$$

Like the value function $V^\pi$, it is possible for us to express $Q^\pi$ recursively and use this recurrence relation to compute the exact $Q$-value for all state-action pairs $(s,a)$. Here,

$$Q^\pi(s,a) = \sum_{s'\in\mathcal{S}} P(s,a,s') \left\{ R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(s',a') Q^\pi(s',a') \right\}, \tag{3.2.11}$$

where $\pi(s',a')$ is the probability of taking action $a'$ from a state $s'$ under the policy $\pi$.

For notational purposes, we can define $R : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ as the expected reward for a state-action pair $(s,a)$ where,

$$R(s,a) = \sum_{s'\in\mathcal{S}} P(s,a,s') R(s,a,s'). \tag{3.2.12}$$

Then we can rewrite the Bellman equation for $Q^\pi$ in matrix form as,

$$Q^\pi = R + \gamma \mathbf{P}\mathbf{\Pi}_\pi Q^\pi, \tag{3.2.13}$$

where $\mathbf{P}$ is a stochastic matrix of the transition probabilities of size $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ such that $\mathbf{P}((s,a),s') = P(s,a,s')$, and $\mathbf{\Pi}_\pi$ is a stochastic matrix with dimensions of size

$|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$ describing the policy $\pi$. That is, we can choose to write the policy matrix $\mathbf{\Pi}_\pi$ in the following form,

$$\mathbf{\Pi}_\pi(s', (s', a')) = \pi(s', a'). \tag{3.2.14}$$

In this case, the optimal value for a state action pair, $Q^*(s, a)$ is

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') \left\{ R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right\}. \tag{3.2.15}$$

Here, the optimal policy can be written as

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \left\{ R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right\}. \tag{3.2.16}$$

RL algorithms can typically be classified into one of three categories, depending on the strategy they employ to determine an optimal policy. These are: *value iteration*, *policy iteration* and *policy search* (Buşoniu et al., 2010). Value iteration methods search for the optimal value function by iteratively computing the maximal returns for each state or state-action pair; the optimal policy is then derived from the optimal value function. Policy iteration methods determine an optimal policy by constructing a sequence of monotonically improving policies and evaluating the performance of each of these policies. Policy search algorithms use optimization methods to explicitly search for an optimal policy. In this study, we will be focusing on value iteration methods and policy iteration methods only.

## 3.3 Model-based value iteration

When a model of the MDP dynamics and reward function is available, it is possible for the Bellman optimality equation to be solved directly. In this case, the optimal value function may be computed iteratively and can be used to derive an optimal policy. Model-based value iteration begins by arbitrarily assigning an estimate to either the $Q$-function or $V$-function and progressively refining this value until no further refinement is required. At each iteration, the relevant value function is updated by transforming the original associated Bellman equation into an update rule. We present a $Q$-value iteration algorithm here (Buşoniu et al., 2010). We use the notation $Q_k$ to denote the value of the $Q$-function after $k$ iterations of the learning algorithm, where $k = 0, 1, 2, \ldots$.

**Algorithm 1** Pseudocode for the $Q$-value iteration algorithm (Buşoniu et al., 2010).

*Input:* Transition dynamics $P$, reward function $R$ and discount factor $\gamma$ .
*Output:* Optimal $Q$-function, $Q^*$.
**Q_iteration($P$, $R$, $\gamma$):**
Initialize $Q$ arbitrarily everywhere on $\mathcal{S} \times \mathcal{A}$. For example, $Q_0(s,a) \leftarrow 0$ for each state-action pair $(s,a)$
$k = 0$
**repeat**
  **for** each state-action pair $(s,a)$ **do**
    $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s,a,s') \left\{R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q_k(s',a')\right\}$
  **end for**
**until** $Q_{k+1} = Q_k$
**return** $Q^* = Q_k$

## 3.4 Model-free value iteration: the $Q$-learning framework

In most situations, a complete model of the MDP dynamics will not be available. In these cases, the agent is required to learn a task without prior knowledge of the environment's transition function or the probability distribution of the random variables concerned. The $Q$-learning framework due to Watkins (1989) is one of the most important and widely used model-free *off-policy* techniques developed in the context of reinforcement learning, whereby an optimal policy can be learned while evaluating another policy. Since $Q$-learning is an off-policy method, an optimal $Q$-function, $Q^*$, can be approximated directly, independent of the policy being evaluated at a certain time (Sutton & Barto, 1998). In particular, the algorithm operates by learning a $Q$-function which gives us an estimate of the expected return given that a particular action is taken from a certain state and the optimal policy is followed thereafter. Typically a tabular representation of the $Q$-values for each ($s$,$a$) pair is maintained. Initially, these $Q$-values are assigned arbitrary values which are then modified throughout the learning process. During $Q$-learning an agent's experience is divided into subsequences of repeated interaction called *episodes*. Learning progresses in a manner analogous to temporal difference (TD)-learning (Sutton, 1988) where an agent takes an action at a certain state, evaluates its immediate reward and then updates its current estimate of the value of the state based on the value of the resulting state (Watkins & Dayan, 1992). That is, the value of a ($s$,$a$) pair is updated using the immediate reward that an agent receives with the discounted estimated optimal future value. An episode reaches completion once the resulting state $s_{t+1}$ is an absorbing or terminal state.

In its simplest form, the $Q$-learning algorithm makes use of a standard value iteration type update where a previous value is modified on the basis of newly available data. The $Q$-learning update rule is given by

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1})\right]. \tag{3.4.1}$$

Here $r_t$ represents the reward at time $t$, $\gamma$ is the discount factor and $\alpha \in (0, 1]$ is the learning rate which measures the rate at which new information influences previously learned values. The $(1 - \alpha)$ weight can be viewed as an inverse learning rate that is attached to the old estimate of the value of the state-action pair $(s_t, a_t)$. $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ is an estimate of the optimal future value of the subsequent state $s_{t+1}$.

By rearranging terms, the $Q$-learning update rule 3.4.1 can be rewritten as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]. \tag{3.4.2}$$

We refer to Sutton & Barto (1998) for the $Q$-learning algorithm presented here as Algorithm 2.

**Algorithm 2** Pseudocode for the $Q$-learning algorithm (Sutton & Barto, 1998).

*Input:* $Q(s, a)$ initialized arbitrarily; start state $s_0$.
*Output:* $Q(s, a)$ for all pairs $(s, a)$
**Q_learning($Q$, $s_0$):**
**repeat**
  Choose $a$ for the current $s$ on the basis of $Q(s, a)$
  Take action $a$, observe reward $r$ and resulting state $s'$
  $Q(s, a) \leftarrow Q(s, a) + \alpha \{r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)\}$
  $s \leftarrow s'$
**until** the end of the learning episode
**return** updated list of $Q$ values

## 3.5 Batch reinforcement learning methods

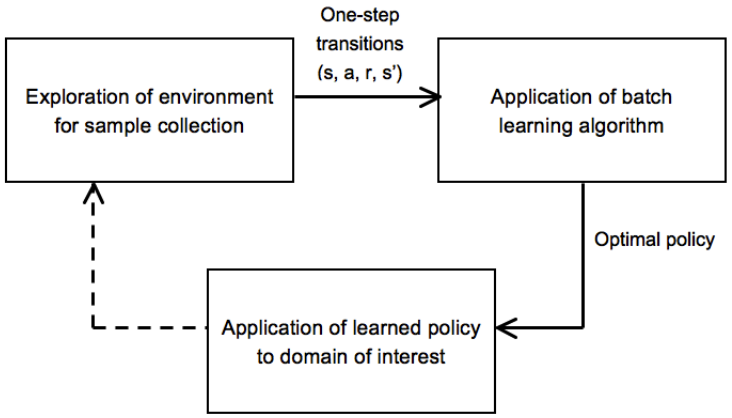



Figure 3.1: Stages of batch reinforcement learning.

Conventional reinforcement learning uses online learning to update policies. In this approach, the agent interacts with the environment dynamically and updates its control policy at each time step (Sutton & Barto, 1998). However, in many medical domains this may not be possible: the effects of having an untrained agent may be potentially hazardous and impose an unacceptable risk to the patients concerned. In these situations, the agent and system cannot interact directly during learning, hence interaction must be decoupled from the learning step (Lange et al., 2012). That is, experience from agent-environment interaction must be collected beforehand through a series of experimental trials. The agent is then trained on this set of previously recorded information containing state, action and reward data. The policy that is learned from the samples collected can be applied back to the environment for further sample collection and policy refinement. Learning techniques that make use of pre-recorded data are termed *batch reinforcement learning* or simply *batch learning* methods. Figure 3.1 shows the main steps involved in the batch reinforcement learning problem.

As with the standard reinforcement learning problem, the aim of batch reinforcement learning is to determine a policy such that the expected cumulative reward is maximized. In general, the agent cannot make any assumptions about the manner in which transition data is collected, since sampling may be completely random or in accordance to an arbitrary policy (Lange et al., 2012). Popular examples of batch reinforcement learning methods are the fitted $Q$-iteration algorithm due to Ernst et al. (2005) and the least squares policy iteration method due to Lagoudakis & Parr (2003). We examine each of these learning techniques at length in the rest of this chapter.

## 3.6 Fitted $Q$-iteration

The fitted $Q$-iteration (FQI) algorithm is a batch reinforcement learning technique inspired by the online $Q$-learning framework where an agent learns a $Q$-function without explicit knowledge of the transition probability function. Unlike the $Q$-learning algorithm, the exact $Q$-function representation is replaced by an approximation, and a set of predetermined four-tuples, $\mathcal{F} = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\}_{i=1,\ldots,|\mathcal{F}|}$, is used. Here, $|\mathcal{F}|$ is the cardinality of the set $\mathcal{F}$ and each tuple is an example of the one-step transition dynamics of the system.

Typically, in domains with small or finite state and action spaces, it is possible for the $Q$-function to be represented as a table where one entry exists for each state-action pair. Deriving an optimal policy for the $Q$-function in this case, is straightforward. However for larger problems where state or action spaces are continuous, this is not feasible since an infinite number of states or actions may exist. In these instances, the $Q$-value function must be approximated to generalize similar circumstances and/or actions. Furthermore, in situations where batch reinforcement learning methods such as fitted $Q$-iteration are used, an approximation of the $Q$-function must be derived using sparse sets of four-tuples (Ernst et al., 2005). This problem is common in large reinforcement learning systems and is referred to as the *generalization problem* (Sutton & Barto, 1998).

To overcome the generalization problem, *function approximation* techniques are used. Function approximators attempt to use examples from the $Q$-value function to generalize the $Q$-values over a larger subset (Sutton & Barto, 1998). When attempting to estimate a value function, examples of available $Q$-values are used to construct an approximation of the entire $Q$-function. Currently, a number of supervised learning techniques have been used as function approximators. In particular, Ormoneit & Sen (1999) apply the idea of *kernels* to the fitted value iteration method due to Gordon (1999). Here, the problem of approximating a $Q$-function is reformulated into a sequence of kernel regression problems. Kernels are a non-parametric function approximator that map two elements from a space of input patterns to a real number. This number may be viewed as a similarity measure between the patterns on the input space (Bethke et al., 2008). A formal definition of the kernel is provided below (refer to Buşoniu et al. (2010) and Taylor & Parr (2009)) for details).

**Definition 3.6.1.** *A **kernel** $\kappa(\cdot,\cdot)$ is a map that is defined over two state-action pairs where $\kappa : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. A symmetric kernel matrix, $\mathbf{K}$, stores the kernel values for all input pairs in a dataset. If $\mathbf{K}$ is a positive semi-definite matrix[2], the kernel function may be viewed as the inner product between two points (i.e. two state-action pairs) in a higher dimensional space.*

As with other non-parametric approximators, kernels are highly flexible since their shape is dependent on the data used when running the associated reinforcement learning algorithm (Buşoniu et al., 2010). By expressing the $Q$-function in terms of a series of regression problems, Ormoneit & Sen (1999) allow for various regression algorithms to be applied to the same problem.

The fitted $Q$-iteration algorithm builds on the idea of fitted value iteration presented in Ormoneit & Sen (1999). In particular, it constructs an approximation of the $Q$-function iteratively. That is, on the first iteration, the algorithm produces an approximation of the $Q_1$-function, $\widehat{Q}_1$, which corresponds to a one-step optimization (Ernst et al., 2005). This approximation is produced by applying a regression algorithm to training data where the input consists of the pairs $(s_t, a_t)$ and the target output, $y$, consists of the rewards $r_t$ (Ernst et al., 2005). Similarly, the $N^{th}$ iteration produces an approximation of the $Q_N$-function, $\widehat{Q}_N$, which corresponds to an $N$-step optimization. This approximation is calculated in the same manner as before except the target output values are obtained from applying a value-iteration based update to the $Q$-function approximation from the previous time-step. That is, the output values, $y$, are updated using,

$$y = r_t + \gamma \max_{a \in \mathcal{A}} \widehat{Q}_{N-1}(s_{t+1}, a). \tag{3.6.1}$$

We note that any regression algorithm may be used to determine the mapping $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Examples of techniques employed to learn this mapping include using extremely randomized trees (Geurts et al., 2006), neural networks (Riedmiller, 2005a), sparse Gaussian processes (Rasmussen & Williams, 2006) and kernel-based methods

[2]We refer the reader to Hoffman et al. (2008) for a description of this mathematical property.

(Ormoneit & Sen, 1999). For the purposes of this research, we focus on extremely randomized trees and neural networks only. Reasons for this are based on results from Ernst et al. (2005) where the advantages of extremely randomized trees are highlighted in comparison to classical approaches such as tree-bagging, pruned CART trees, KD-trees, and other supervised learning techniques such as $k$-nearest neighbours. In particular, Ernst et al. (2005) demonstrate that ensembles of extremely randomized trees perform significantly better when dealing with large state spaces. Reasons for using neural networks as opposed to other supervised learning techniques are based on the popularity of the technique. Fitted $Q$-iteration has often been used with neural networks in the past and these results are well-documented and frequently cited.[3]

As $N \to \infty$, the approximation, $\widehat{Q}_N$, is refined and converges to the actual $Q$-function. The interested reader should refer to Ernst et al. (2005) for the proof of convergence of the fitted $Q$ algorithm.

### 3.6.1 The algorithm

We refer to Ernst et al. (2005) for the following fitted $Q$-iteration algorithm.

**Algorithm 3** Pseudocode for the fitted $Q$-iteration algorithm (Ernst et al., 2005).

*Input:* A set $\mathcal{F} = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i) | i = 1, \ldots, |\mathcal{F}|\}$ of four-tuples, discount factor $\gamma$, maximum number of iterations $N_{max}$ and a regression algorithm.
*Output:* $\widehat{Q}^* = \widehat{Q}_N$.
**Fitted_Q-iteration($\mathcal{F}$, $\gamma$, $N_{max}$):**
Initialize number of iterations $N$ to 0
Initialize $\widehat{Q}_N$ everywhere on $\mathcal{S} \times \mathcal{A}$
**repeat**
$N \leftarrow N + 1$
Build a training set $\mathcal{TS}$ from $\mathcal{F}$ where $\mathcal{TS} = \{(x^i, y^i), i = 1, 2, \ldots, |\mathcal{F}|\}$ based on $\widehat{Q}_{N-1}$ and on the set of four-tuples $\mathcal{F}$. Here $(x^i, y^i)$ are given by

$$x^i = (s_t^i, a_t^i) \tag{3.6.2}$$

$$y^i = r_t^i + \gamma \max_{a \in \mathcal{A}} \widehat{Q}_{N-1}(s_{t+1}^i, a). \tag{3.6.3}$$

Deduce the value $\widehat{Q}_N(s, a)$ using non-parametric regression on $(x^i, y^i)$.
**until** $\widehat{Q}_N$ is satisfactory or $N_{max}$ has been reached.
**return** updated $\widehat{Q}_N$ values

The algorithm builds a training set $\mathcal{TS}$ using a set of one-step transitions and calculates a new target $Q$-value with each transition. A supervised learning technique is used to train a function approximator on the training set. The resulting approximation $\widehat{Q}_N$ is an approximation of the $Q$-function after $N$-steps of value iteration performed on the state-action value function (Lange et al., 2012). Consecutive runs of the regression

[3] See for instance Riedmiller (2005a).

algorithm are entirely independent (Ernst et al., 2005). The implications of this are that it is possible to adapt the model learned at each step so as to find a $Q$-function that best models the input data.

The learning process continues until a certain predefined number of iterations, $N_{max}$ has been reached or the difference between the $Q$-functions from successive iterations is significantly small i.e. the $Q$-function has converged reasonably. Once an accurate enough representation of the actual $Q$-function has been obtained after $N$ steps where $N$ is arbitrarily large, an optimal stationary control policy, $\widehat{\pi}^*_N$, can be determined using

$$\widehat{\pi}^*_N(s) = \arg\max_{a \in \mathcal{A}} \widehat{Q}_N(s, a). \tag{3.6.4}$$

The existence of such a policy is a classical result from dynamic programming theory (see for instance Bellman (1957)).

## 3.7 Extremely randomized trees

Typically when dealing with continuous state spaces consisting of numerous features that interact in a complex way, determining a suitable representation for the $Q$-function is difficult. In such situations, it may be necessary to use non-linear regression techniques to subdivide the space into smaller regions to which we can apply simple models and derive an approximation. Each subregion is partitioned into successively smaller regions until an acceptable level of interaction between features is observed. This procedure is referred to as *recursive partitioning*. Recursive partitions may be represented as regression trees where each leaf node represents a cell of the overall partition with a particular model or label associated with it; the branches represent the combining of features that ultimately result in these labels (Strobl et al., 2009).

### 3.7.1 Single tree regression

Single tree regression grows one regression tree from a training set and uses it for the classic purposes of prediction and classification. Trees are grown by splitting a sample set into smaller subsets on the basis of certain attributes. Splitting occurs in accordance with certain criteria: most often, the split which optimizes the associated node's entropy is chosen. Informally, entropy is a measure of disorderedness or information content of a set. When we split a set of samples into two smaller subsets on the basis of a particular attribute, we choose the attribute which maximizes the difference between these two sets. That is, we select the split that reduces the disorderedness the most or maximizes the information gained from performing the split. Popular regression techniques that produce a single tree are CART, KD-tree, ID3, C4.5 and QUEST (Marsland, 2009). These trees are often pruned to improve their prediction accuracy (Ernst et al., 2005).

### 3.7.2 Ensembles of extremely randomized trees

*Extremely randomized trees*, also called *extra trees*, refer to a class of unpruned regression trees used for the purposes of prediction and classification. As with standard regression

trees, extra trees approximate the relationship between an input $x$ and output $y$ using a set of samples. They do so by employing a top-down approach to growing trees: the input space is divided into several partitions that are refined progressively to produce a reasonable approximation. Specifically, within the context of fitted $Q$-iteration, extra trees can be used to partition the input space of a set of samples to produce a suitable $Q$-function approximation (Wehenkel et al., 2006). Extremely randomized trees may be viewed as an ensemble tree-based method for regression since sets of trees are grown to derive an approximation. In particular, the extra trees algorithm constructs a forest of regression trees at each iteration and averages the outputs of each tree to produce a prediction. It is thought that the predictions of multiple classifiers can be used to produce a single classifier that exhibits improved prediction accuracy in general (Geurts et al., 2006). Unlike classical tree-based ensemble methods, extra trees are constructed by selecting cut points at random. That is, each tree grows by choosing $k$ random split points at each node. This, in turn, requires $k$ random candidate attributes or input variables comprising the state and/or action space to be selected, and determining a random split on each of these attributes (Geurts et al., 2006). The best split which maximizes a certain score measure is retained (Wehenkel et al., 2006). Trees are grown recursively until the outputs are constant or contain less than a specified number of samples, $n_{min}$. Furthermore, the entire training sample is used to grow trees at each iteration as opposed to the bootstrapping techniques employed by similar methods such as tree-bagging (Geurts et al., 2006). We discuss the details of the extra trees algorithm in the next section.

### 3.7.3 The extra trees algorithm

The complete extra trees algorithm is presented in Algorithm 4 (Geurts et al., 2006). The algorithm uses three parameters, $n_{min}$, $k$ and $M$ to build an ensemble model of extra trees recursively from a given training set, $\mathcal{TS}$. $n_{min}$ is the smallest sample size required to split a node of a tree in the ensemble; $k$ is the number of attributes selected at each node of a tree that are used to determine a split point; $M$ is the number of trees in the ensemble (Geurts et al., 2006). To construct an individual tree, a root node is initially created containing the entire training set of samples. The root node is subsequently split to produce left and right child nodes respectively. Splitting involves selecting an attribute or direction to split on, $a$, as well as a corresponding scalar cut value, $a_c$. Once a node is split, the set of samples associated with the node is divided into two disjoint sets, $\mathcal{TS}_l$ and $\mathcal{TS}_r$, containing the samples to the left and right of the split-point respectively. Using set theoretic notation, we can denote the sets $\mathcal{TS}_l$ and $\mathcal{TS}_r$ as,

$$\mathcal{TS}_l = \{(x^i, y^i) \in \mathcal{TS} | x_a^i < a_c\}, \tag{3.7.1}$$

$$\mathcal{TS}_r = \{(x^i, y^i) \in \mathcal{TS} | x_a^i \geq a_c\}. \tag{3.7.2}$$

Here $(x^i, y^i)$ is the $i^{th}$ input-output pair of the initial training set where $i = 1, 2, \ldots, |\mathcal{F}|$,

**Algorithm 4** Pseudocode of the extra trees algorithm (Geurts et al., 2006).

**Pick_a_random_split(**$\mathcal{TS}, a$**):**
*Input:* A training set $\mathcal{TS}$ and an attribute to split on $a$.
*Output:* A split $[a < a_c]$.
Determine the maximal and minimal values, $a_{max}^{\mathcal{TS}}$ and $a_{min}^{\mathcal{TS}}$, of the attribute $a$ in $\mathcal{TS}$
Select a splitting point $a_c$ uniformly from $[a_{min}^{\mathcal{TS}}, a_{max}^{\mathcal{TS}}]$
**return** the split $[a < a_c]$

**Build_an_extra_tree(**$\mathcal{TS}, n_{min}, k$**):**
*Input:* A training set $\mathcal{TS}$, parameters $n_{min}$, $k$.
*Output:* A tree $\mathcal{T}$.
**return** a leaf node whose value is given by the average output in $\mathcal{TS}$ **if**
(i) $|\mathcal{TS}| < n_{min}$, **or**
(ii) the output is constant in $\mathcal{TS}$, **or**
(iii) the candidate attributes are constant in $\mathcal{TS}$
**else**
Choose $k$ non-constant attributes $\{a_1, a_2, \ldots, a_k\}$ at random in $\mathcal{TS}$
Create $k$ random splits $\{a_{c_1}, a_{c_2}, \ldots, a_{c_k}\}$ where $a_{c_i}$ = **Pick_a_random_split(**$\mathcal{TS}, a_i$**)**, $\forall i = 1, 2, \ldots, k$
Pick an attribute $a$ and split $a_c$ such that Score($a$, $a_c$, $\mathcal{TS}$) = $\max_{i=1,2,\ldots,k}$ Score($a_i$, $a_{c_i}$, $\mathcal{TS}$)
Split $\mathcal{TS}$ into two subsets, $\mathcal{TS}_l$ and $\mathcal{TS}_r$, according to $a$ and $a_c$.
Build trees $\mathcal{T}_l$ = **Build_an_extra_tree(**$\mathcal{TS}_l$**,** $n_{min}$**,** $k$**)** and $\mathcal{T}_r$ = **Build_an_extra_tree(**$\mathcal{TS}_r$**,** $n_{min}$**,** $k$**)** using the subsets $\mathcal{TS}_l$ and $\mathcal{TS}_r$
Build a node with split $a_c$ with $\mathcal{TS}_l$ and $\mathcal{TS}_r$ as left and right subtrees of the node
**return** the resulting tree $\mathcal{T}$.

**Build_an_extra_tree_ensemble(**$\mathcal{TS}$**,** $M$**,** $n_{min}$**,** $k$**):**
*Input:* A training set $\mathcal{TS}$, parameters $M$, $n_{min}$, $k$.
*Output:* An ensemble of trees $\mathcal{E}$.
**for** $j = 1$ to $M$ **do**
Tree $\mathcal{T}_j$ = **Build_an_extra_tree(**$\mathcal{TS}$**,** $n_{min}$**,** $k$**)**
Add $\mathcal{T}_j$ to ensemble $\mathcal{E}$
**end for**
**return** $\mathcal{E} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_M\}$

$x_a^i$ is the $i^{th}$ input value at the attribute $a$ and $a_c$ is the split value (Buşoniu et al., 2010). The splitting process continues at each child node produced thereafter. Each leaf node containing more than $n_{min}$ samples, where $n_{min} \geq 2$, is split.

To determine precisely how a node is split, $k$ non-constant attributes, $a_i$, with $k$ corresponding cut-points, $a_{c_i}$, where $i = 1, 2, \ldots k$, are selected. These cut-points are drawn uniformly in the range of their associated attribute. That is, for an attribute, $a_i$, we randomly choose the cut point, $a_{c_i}$, such that $a_{c_i} \in [a_{i_{min}}, a_{i_{max}}]$, where $a_{i_{min}}$ and $a_{i_{max}}$ represent the minimal and maximal values of the attribute $a_i$ respectively, $\forall i = 1, 2, \ldots k$. Ultimately the cut point, $a_c$, corresponding to the attribute, $a$, that maximizes the score

$$Score(a, a_c, \mathcal{TS}) = \frac{\mathbb{V}\text{ar}[y|\mathcal{TS}] - \frac{|\mathcal{TS}_l|}{|\mathcal{TS}|}\mathbb{V}\text{ar}[y|\mathcal{TS}_l] - \frac{|\mathcal{TS}_r|}{|\mathcal{TS}|}\mathbb{V}\text{ar}[y|\mathcal{TS}_r]}{\mathbb{V}\text{ar}[y|\mathcal{TS}]}, \tag{3.7.3}$$

is retained. Here, $Score(a, a_c, \mathcal{TS})$ denotes the score of splitting the training set $\mathcal{TS}$ on the attribute $a$ according to the split value $a_c$, and $\mathbb{V}\text{ar}[\cdot]$ is the variance of the output $y$ in the associated training set (Geurts et al., 2006). The score may be viewed as a variance reduction over time on the output of the original training set based on the Shannon entropy: we continue to split nodes of the tree until the output observed remains constant or until the variance of the output is reduced sufficiently. To perform a single split, we evaluate each of the $k$ randomly selected cut points according to the information gained and choose to split the attribute which maximizes this information gain. In doing so, we can determine which attributes are most relevant: attributes with a higher mutual information tend to be tested first and are hence used for splitting earlier than others. Intuitively, we aim to select those attributes that split the data in such a way that the samples of each successor node belong to a single class or are as "pure" as possible. When this occurs, the splitting procedure reaches completion since no new information can be gained from further splitting. We refer the reader to Wehenkel (1996) for a detailed treatment of uncertainty measures such as Shannon entropy, that are used for decision tree construction.

The parameters $k$, $n_{min}$ and $M$ serve different purposes: $M$ dictates the degree of variance reduction when aggregating the results from the ensemble of trees while $n_{min}$ determines the influence of averaging output noise and influences the size of the trees produced. Larger $n_{min}$ values produce smaller trees with less variance (Wehenkel et al., 2006). The parameter $k$ determines how attributes are selected (Geurts et al., 2006). In particular, it is a direct measure of the randomization strength of the attribute selection process: a lower $k$ results in trees with stronger randomization; for the specific case of when $k = 1$, the resulting splits are entirely independent of the output values of the training samples (Geurts et al., 2006). This results in the production of *totally randomized trees* which are a special instance of extra trees. The values of the parameters $k$, $n_{min}$ and $M$ should be adjusted according to the problem at hand. However, larger $k$-values are suited to problems with a large number of input variables, few of which are pertinent; larger $n_{min}$ values are used in domains where the output variable is particularly noisy (Wehenkel et al., 2006).

Once a tree is constructed, determining its output is relatively straightforward: starting at the root node, a test consisting of the node's split value and cut-point are applied to the node. The result of this test determines which subtree to traverse. This continues until a leaf node is reached. The value of the leaf is returned as the average output of the samples associated with it. This procedure is outlined by Algorithm 5 (Buşoniu et al., 2010).

It is possible to express the structure of the extra trees approximator in terms of kernels. If we consider the $j^{th}$ regression tree, $\mathcal{T}_j$, in the ensemble of trees $\mathcal{E}$, we can define a function $p_j(x)$ that assigns each input $x$ to the region or partition it belongs to given by the tree $T_j$. Assume the prediction or approximate output of the $j^{th}$ regression tree in an ensemble is given by $\widehat{y}_j(x)$. The prediction may be viewed as the average output of the samples belonging to the region $p_j(x)$. Using kernels, this can be written as:

$$\widehat{y}_j(x) = \sum_{i=1}^{|\mathcal{F}|} \kappa(x, x^i) y^i. \tag{3.7.4}$$

The kernel $\kappa(x, x^i)$ is given by

$$\kappa(x, x^i) = \frac{\mathbb{1}\{x^i \in p_j(x)\}}{\sum_{i'=1}^{|\mathcal{F}|} \mathbb{1}\{x^{i'} \in p_j(x)\}}, \tag{3.7.5}$$

where $\mathbb{1}\{\cdot\}$ denotes the indicator function that produces 0 if its argument is false and 1 if it is true (Buşoniu et al., 2010).

Using Equation 3.7.4, we can express the final prediction or output produced by the ensemble of regression trees as an average of the predictions of each tree. That is,

$$\widehat{y}(x) = \frac{1}{M} \sum_{j=1}^{M} \widehat{y}_j(x). \tag{3.7.6}$$

The final output can also be expressed in terms of kernels where the kernel function $\kappa(x, x^i)$ is given by

$$\kappa(x, x^i) = \frac{1}{M} \sum_{j=1}^{M} \frac{\mathbb{1}\{x^i \in p_j(x)\}}{\sum_{i'=1}^{|\mathcal{F}|} \mathbb{1}\{x^{i'} \in p_j(x)\}}. \tag{3.7.7}$$

**Algorithm 5** Pseudocode of the algorithm used to predict the output of a tree (Buşoniu et al., 2010).

*Input:* A tree $\mathcal{T}$ and a point $x$.
*Output:* The output of the tree $\mathcal{T}$.
**while** $\mathcal{T}$ is not a leaf
  $(a, a_c) \leftarrow$ test corresponding to root node of $\mathcal{T}$.
  **if** $x_a < a_c$ **then**
    $\mathcal{T} \leftarrow \mathcal{T}_l$
  **else** $\mathcal{T} \leftarrow \mathcal{T}_r$
  **end if**
**end while**
Return the output of $\mathcal{T}$

## 3.8 Neural fitted $Q$-iteration

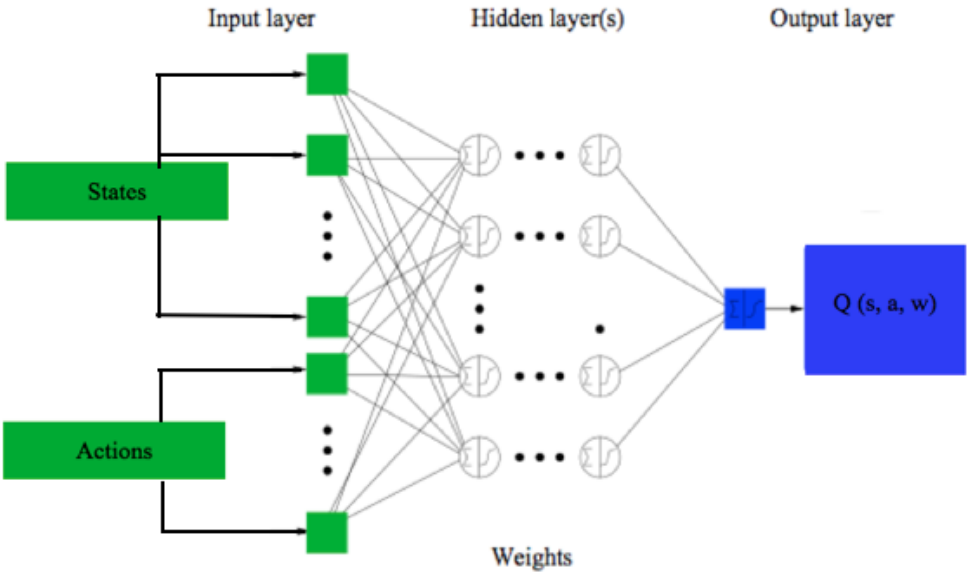



Figure 3.2: Multilayer perceptron structure for neural fitted $Q$-iteration (Figure adapted from Riedmiller (2010)).

Neural fitted $Q$-iteration (NFQ) is a batch reinforcement learning technique based on the fitted value iteration algorithm due to Ormoneit & Sen (1999). Like other batch learning techniques, neural fitted $Q$-iteration is based on the principle of storing and reusing transition experiences; in particular, this method may be viewed as a modified version

of the experience-replay technique since value iteration is executed on all the transition samples collected at a time (Riedmiller, 2005b). In Section 3.7, we introduced the idea of fitting training data to a value function using ensembles of regression trees. Neural fitted $Q$-iteration attempts to solve the same approximation problem using the multilayer perceptron as opposed to regression trees to perform non-parametric regression on a set of input-output pairs. A *multilayer perceptron* (MLP) is a feedforward artificial neural network with one or more hidden layers that maps a set of input data to a set of corresponding outputs. Here, the aim of an MLP is to learn an optimal $Q$-function successfully from a relatively sparse set of samples without requiring explicit knowledge of the system's dynamics (Riedmiller, 2005a). That is, the neural fitted $Q$-iteration algorithm attempts to provide a model-free data-efficient alternative to approximating the $Q$-function. The optimal $Q$-function can in turn be used to derive an optimal policy for the domain of interest.

Although Riedmiller (2005a) demonstrates that neural networks may be applied directly to the online $Q$-learning problem by introducing an error measure that can be minimized using gradient descent techniques, this procedure is slow and is not suited to domains where the agent and system cannot interact directly during learning. Instead, the neural fitted $Q$-iteration algorithm uses a set of transition experiences $\mathcal{F}$ collected through prior agent-environment interaction. The $Q$-function update is then performed offline on the basis of these transitions. Riedmiller (2005b) observes that the use of previously collected transition samples enables advanced supervised learning techniques to be used to derive an approximate $Q$-function; this ultimately leads to faster convergence than applying standard gradient-descent techniques to the online $Q$-learning framework.

### 3.8.1 The multilayer perceptron as a function approximator

In order to model a set of observational data and determine an adequate approximation of the value function, non-linear regression models need to be used. Non-linear regression is a form of regression analysis in which a set of training data is modelled as a non-linear combination of input variables and parameters. The basic idea is to adequately express the relationship between a set of input and output values.

The multilayer perceptron (MLP) is a feedforward neural network which can serve as a value function approximator within the context of RL. Typically, an MLP consists of multiple layers that direct information flow (Marsland, 2009). The first layer is termed the *input layer*; the last is termed the *output layer*; layers between the input and output layers are termed *hidden layers*. Each layer is comprised of a pre-specified number of nodes that are connected by a set of weighted edges to form a weighted directed graph. The input nodes are responsible for distributing signals to the nodes of the first hidden layer. Each node at a particular hidden layer is responsible for transforming the signal from the previous layer to an output signal that can be distributed across the next layer. Changing a signal from one hidden layer to the next involves using an *activation function*. That is, the output from a hidden node is calculated as a weighted sum of the

signals from the previous layer that is transformed using an activation function. Examples of activation functions include the sigmoid function, the Heaviside step function and the Gaussian function (Marsland, 2009). Training a neural network involves comparing the output values and targets for a given set of inputs to determine an error value; this error can be used to determine how to update the weights of the directed edges within the MLP. A variety of training algorithms exist, most of which apply some modified form of gradient-descent to determine the weights of the edges within the network; popular examples include the backpropagation algorithm and the resilient backpropagation algorithm. The interested reader should refer to Rumelhart et al. (1986) and Riedmiller (1994) or Riedmiller & Braun (1993) respectively, for detailed descriptions of each of these training techniques.

Within the RL framework, it is possible to use the multilayer perceptron to represent a $Q$-function. Here, a simple backpropagation neural network would suffice with one input node corresponding to each dimension of the state space and action space and one output node corresponding to the approximate $Q$-value. The number of hidden layers and nodes would depend on the problem being solved. Using MLPs in this context offers certain advantages over other function approximation techniques: in particular, the MLP makes no assumptions about how the data is distributed. As a result, the MLP can be used to model highly non-linear functions and is capable of adapting to new circumstances or suitably generalizing unseen data. Figure 3.2 has been adapted from Riedmiller (2010) and shows the structure of an MLP when applied to the neural fitted $Q$-iteration algorithm. In particular, the set of inputs used by the neural network consists of the state-action pairs made available from the sample set; the output consists of the corresponding $Q$-values.

### 3.8.2 The algorithm

The neural fitted $Q$-iteration procedure is given by Algorithm 6. We observe that the algorithm largely follows the fitted $Q$-iteration algorithm presented in Section 3.6. Here the non-parametric regression step is realized by the multilayer perceptron (Riedmiller, 2005a). In particular, the multilayer perceptron is trained on the set of input-output pairs, $(x, y)$, consisting of state and action information for the transition, and our current estimate of the $Q$-function respectively. The structure of the multilayer perceptron used is largely dependent on the problem at hand; training from the set of patterns occurs repeatedly for a specified number of epochs or until the training pattern is learned (Riedmiller, 2005b). We use the notation $\mathcal{B}$ to represent any MLP training algorithm of the user's choice.

**Algorithm 6** Pseudocode of the neural fitted $Q$-iteration algorithm (Riedmiller, 2005a).

*Input:* A set $\mathcal{F} = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i) | i = 1, \ldots, |\mathcal{F}|\}$ of four-tuples, discount factor $\gamma$, maximum number of iterations $N_{max}$ and an algorithm $\mathcal{B}$ to train the multilayer perceptron.

*Output:* $\widehat{Q}^* = \widehat{Q}_N$.
**Neural_fitted_Q-iteration($\mathcal{F}, \gamma, N_{max}$):**
Initialize number of iterations $N$ to 0
Initialize $\widehat{Q}_N$ everywhere on $\mathcal{S} \times \mathcal{A}$ (i.e. initialize multilayer perceptron.)
**repeat**
$N \leftarrow N + 1$
Build a training set $\mathcal{TS}$ from $\mathcal{F}$ where $\mathcal{TS} = \{x^i, y^i), l = 1, 2, \ldots, |\mathcal{F}|\}$ based on $\widehat{Q}_{N-1}$ and on the set of four-tuples $\mathcal{F}$. Here $(x^i, y^i)$ are given by:

$$x^i = (s_t^i, a_t^i) \tag{3.8.1}$$

$$y^i = r_t^i + \gamma \max_{a \in \mathcal{A}} \widehat{Q}_{N-1}(s_{t+1}^i, a) \tag{3.8.2}$$

Apply algorithm $\mathcal{B}$ to train neural network with the pattern set $\mathcal{TS}$ to deduce $\widehat{Q}_N(s, a)$
**until** $\widehat{Q}_N$ is satisfactory or $N_{max}$ has been reached.
**return** updated $\widehat{Q}_N$ values

## 3.9 Least Squares methods for approximate policy evaluation

An alternative to the class of approximate value iteration algorithms presented in the previous sections is the class of approximate policy iteration methods. Traditional policy iteration methods work by iteratively evaluating and improving policies on the basis of computing either the value function or state-action value function directly (Buşoniu et al., 2012). Here, the state-action space is finite and hence exact representations of the value function and the policy under consideration, are possible. The generalization problem in most practical applications of reinforcement learning means that explicit representation of the policy and value function is not feasible. In these cases, approximate policy iteration methods must be used. Usually this requires the Bellman equation for the value function to be solved approximately.

Existing approximate policy iteration algorithms typically exploit the linearity of the Bellman equation for a particular value function. By doing so, it is possible for the value function to be represented using linear architectures. Consider the approximation for the $Q$-value function under a linear approximation architecture: here, a single $Q$-value for the state-action pair $(s, a)$, may be expressed as a linear parametric combination of $k$ basis features or functions using,

$$\widehat{Q}^{\pi}(s,a;w) = \sum_{j=1}^{k} \phi_j(s,a) w_j. \tag{3.9.1}$$

Here, $\phi_j(s,a)$ are the basis functions of the state-action pairs and $w_j$ are the weights associated with these functions, $\forall j = 1, 2, \ldots, k$. The basis functions may be viewed as arbitrary non-linear functions of state-action pairs that are used to capture the underlying structure of the $Q$-function in an attempt to reduce the dimensionality of large state-action space to $\mathbb{R}^k$ where $k \leq |\mathcal{S}||\mathcal{A}|$ in general. Note that selection of appropriate basis functions is critical for good performance of approximate policy iteration methods; these basis functions must be linearly independent to prevent redundant parameters (Lagoudakis & Parr, 2003). Common choices for basis functions include Gaussian radial basis functions and polynomial bases of varying degrees (Lagoudakis & Parr, 2003).

Assume $\widehat{Q}^{\pi}$ is a vector of approximate state-action values determined using a linear approximation with basis functions $\phi_j$, $\forall j = 1, 2, \ldots, k$ and corresponding parameters weighting the basis functions $w_j$. We can define $\phi(s,a)$ to be column vector of length $k$ where the $j^{th}$ entry coincides with the basis function $\phi_j$ at the same state-action pair $(s,a)$. That is, $\phi(s,a)$ is given by

$$\phi(s,a) = \begin{pmatrix} \phi_1(s,a) \\ \phi_2(s,a) \\ \cdots \\ \phi_k(s,a) \end{pmatrix}. \tag{3.9.2}$$

We can then rewrite $\widehat{Q}^{\pi}$ in vector form for all state-action pairs using,

$$\widehat{Q}^{\pi} = \mathbf{\Phi} w^{\pi}, \tag{3.9.3}$$

where $w^{\pi} = (w_1, w_2, \ldots, w_k)$ is the set of weights for the set of basis functions under a policy $\pi$, and $\mathbf{\Phi}$ is the matrix obtained by considering all the basis functions for every state-action pair $(s,a)$. That is,

$$\mathbf{\Phi} = \begin{pmatrix} \phi_1(s_1,a_1) & \phi_2(s_1,a_1) & \cdots & \phi_k(s_1,a_1) \\ \phi_1(s_2,a_2) & \phi_2(s_2,a_2) & \cdots & \phi_k(s_2,a_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) & \phi_2(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) & \cdots & \phi_k(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \end{pmatrix}. \tag{3.9.4}$$

In particular, the rows of the matrix $\mathbf{\Phi}$ correspond all the basis functions used for representing the value function for a particular state-action pair; the columns correspond to the value of a particular basis function for all state-action pairs (Lagoudakis & Parr, 2003). The resulting system of equations of parameters 3.9.3 can then be solved using least squares methods. We can distinguish between two classes of least squares methods on the basis of their approach used to approximate a solution to the Bellman equation for a value function. These are (i) projected policy evaluation and (ii) Bellman Residual Minimization (Buşoniu et al., 2012). Projected policy evaluation methods seek to

determine an approximation for the value function by examining the projection of the value function under the Bellman operator onto the space of representable value functions (Buşoniu et al., 2012); Bellman Residual Minimization methods attempt to solve the Bellman equation directly by calculating a Bellman residual (Buşoniu et al., 2012). Informally, this residual can be seen as a minimization of the difference between the approximation of a value function and its approximation under the Bellman operator. Lagoudakis & Parr (2003) and Munos (2003) demonstrate at length that projected policy evaluation methods have certain advantages over Bellman Residual Minimization methods and are hence a preferred method for approximating the value function. As a result, we restrict our study to projected policy evaluation methods in this chapter.

## 3.10 Projected policy evaluation

We refer to Lagoudakis & Parr (2003) for the material in this section.

Recall the Bellman operator $B^\pi(\cdot)$ that was introduced in Section 3.2.3 of this chapter. A fundamental property of the value function $Q^\pi$ that enables the existence of the recursive Bellman Equation 3.2.11 is that it is a fixed point under the Bellman operator. Formally, $Q^\pi$ may be written as,

$$B^\pi(Q^\pi) = Q^\pi. \tag{3.10.1}$$

Hence if we are to determine a sufficient approximation for the value function, $\widehat{Q}^\pi$, the approximation should be a fixed point under the Bellman operator and lie in the space of representable $Q$-functions. That is,

$$B^\pi(\widehat{Q}^\pi) \approx \widehat{Q}^\pi. \tag{3.10.2}$$

Intuitively, this means that for the approximation of the value function to be easily representable, it must be forced to lie on the space spanned by the basis functions $\mathbf{\Phi}$. This requires projecting the approximation of the value function onto the space of representable value functions using the orthogonal projection $(\mathbf{\Phi}(\mathbf{\Phi}^\top\mathbf{\Phi})^{-\mathbf{1}}\mathbf{\Phi}^\top)$ that minimizes the distance between the approximate value function and the projected approximation of the value function under the Bellman operator. That is, the projected approximation of the value function under the Bellman operator should be as close to approximate value-function as possible to prevent loss of information. By using Equation 3.2.13 and applying it to the approximate $Q$-function, this can be written in matrix form as,

$$\widehat{Q}^\pi = \mathbf{\Phi}(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top(B^\pi(\widehat{Q}^\pi)) = \mathbf{\Phi}(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top(R + \gamma\mathbf{P}\mathbf{\Pi}_\pi\widehat{Q}^\pi), \tag{3.10.3}$$

Substituting Equation 3.9.3 for $\widehat{Q}^\pi$ and rearranging the terms makes it possible for the

solution of the system of equations to be represented as,

$$\begin{aligned} \mathbf{\Phi}(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top(R+\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi}w^\pi) &= \mathbf{\Phi}w^\pi \\ \Rightarrow \mathbf{\Phi}((\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top(R+\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi}w^\pi)-w^\pi) &= 0 \\ \Rightarrow (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top(R+\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi}w^\pi) &= w^\pi \\ \Rightarrow \mathbf{\Phi}^\top(R+\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi}w^\pi) &= \mathbf{\Phi}^\top\mathbf{\Phi}w^\pi \\ \Rightarrow \mathbf{\Phi}^\top(\mathbf{\Phi}-\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi})w^\pi &= \mathbf{\Phi}^\top R. \end{aligned} \tag{3.10.4}$$

Hence we can deduce,

$$w^\pi = \Big(\mathbf{\Phi}^\top(\mathbf{\Phi}-\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi})\Big)^{-1}\mathbf{\Phi}^\top R. \tag{3.10.5}$$

It is possible to replace the standard orthogonal projection with a weighted projection that determines how the error of the overall approximation is distributed. Assuming $\rho$ represents the matrix of the projection weights for each state-action pair, then the solution to the system of equations can be rewritten as,

$$w^\pi = \Big(\mathbf{\Phi}^\top\rho(\mathbf{\Phi}-\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi})\Big)^{-1}\mathbf{\Phi}^\top\rho R. \tag{3.10.6}$$

In this way, projecting the approximation of the value function leads to an easily representable solution to the Bellman equation.

### 3.10.1 Least Squares Temporal Difference learning for $Q$-values

Recall the problem of determining a reasonable approximation $\widehat{Q}^\pi$ for the state-action value function under a particular policy using a set of previously collected samples. The approximation may be expressed in terms of a linear architecture using $k$ linearly independent basis functions each weighted according to the weight vector $w^\pi$ using Equation 3.9.3. Least squares methods for approximate policy iteration typically make use of *parametric function approximators* in the form of basis functions to determine an optimal policy. Parametric approximators are mappings from a parameter space to the space of functions that they attempt to represent (Buşoniu et al., 2010). The choice of basis functions required is independent of the data for a particular problem and is decided in advance. This means that determining a suitable approximation for the state-action value function only requires us to learn the parameters $w^\pi$ from Equation 3.9.3. In particular, the values of $w^\pi$ may be deduced by solving the linear equations,

$$\mathbf{A}w^\pi = b, \tag{3.10.7}$$

where the matrices $\mathbf{A}$ and $b$ can be calculated according to

$$\mathbf{A} = \mathbf{\Phi}^\top\rho(\mathbf{\Phi}-\gamma\mathbf{P}\mathbf{\Pi}_\pi\mathbf{\Phi}), \tag{3.10.8}$$

$$b = \mathbf{\Phi}^\top\rho R. \tag{3.10.9}$$

In general, it is not possible for $\mathbf{A}$ and $b$ to be computed directly since this requires advance knowledge of the transition probability function and reward structure. However,

the values for $\mathbf{A}$ and $b$ can be computed using a set of samples. This computation in turn, makes it possible for the linear system 3.10.7 to be solved to produce a learned estimate of parameters $\tilde{w}^{\pi}$ and ultimately learn the value function.

By expanding $\mathbf{A}$ and $b$ over the sum of all states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$, Lagoudakis & Parr (2003) demonstrate that $\mathbf{A}$ and $b$ have special structures; given a set, $\mathcal{F}$, of pre-collected samples, it is possible to deduce learned estimates, $\tilde{\mathbf{A}}$ and $\tilde{b}$, using,

$$\tilde{\mathbf{A}} = \frac{1}{|\mathcal{F}|} \sum_{i=1}^{|\mathcal{F}|} \phi(s_t^i, a_t^i) \left(\phi(s_t^i, a_t^i) - \gamma \phi(s_{t+1}^i, \pi(s_{t+1}^i))\right)^{\top}, \tag{3.10.10}$$

$$\tilde{b} = \frac{1}{|\mathcal{F}|} \sum_{i=1}^{|\mathcal{F}|} \phi(s_t^i, a_t^i) r_t^i. \tag{3.10.11}$$

For computational purposes the factor $\frac{1}{|\mathcal{F}|}$ may be dropped without changing the solution of the system. Moreover, the system of equations 3.10.10 - 3.10.11 can easily be transformed into update rules for $\tilde{\mathbf{A}}$ and $\tilde{b}$ respectively. For a particular sample $(s_t, a_t, r_t, s_{t+1})$, these are,

$$\tilde{\mathbf{A}}^{(t+1)} = \tilde{\mathbf{A}}^{(t)} + \phi(s_t, a_t) \left(\phi(s_t, a_t) - \gamma \phi(s_{t+1}, \pi(s_{t+1}))\right)^{\top}, \tag{3.10.12}$$

$$\tilde{b}^{(t+1)} = \tilde{b}^{(t)} + \phi(s_t, a_t) r_t, \tag{3.10.13}$$

where $\tilde{\mathbf{A}}^{(t)}$ and $\tilde{b}^{(t)}$ are the current learned estimates for $\mathbf{A}$ and $b$ under a certain policy $\pi$.

Using the incremental update rules 3.10.12 - 3.10.13, Lagoudakis & Parr (2003) construct an algorithm that learns a value-function approximation. The *Least Squares Temporal Difference learning method for Q-values* (LSTD-$Q$) learns an approximation for the $Q$-function under a fixed policy $\pi$ using a batch of pre-recorded samples $\mathcal{F}$. The algorithm takes as input the set of samples used for learning, a discount factor $\gamma$, the initial policy to be evaluated $\pi$ and the set of basis functions $\phi$ used to approximate the state-action values. In return, it produces a vector of weights $\tilde{w}$ corresponding to each basis function. This set of weights, when multiplied by the set of basis functions can be used to determine $\widehat{Q}$. Selecting the action that maximizes the value $\widehat{Q}$ results in an improved policy. When $\pi(s_{t+1})$ is available for each resulting state $s_{t+1}$ in the sample set, the approximate value function can be determined simply by determining which basis function $\phi(s_{t+1}, \pi(s_{t+1}))$ to add to the matrix $\tilde{\mathbf{A}}$. The implications of this for sample-efficiency are tremendous: we can compute the approximate value function for all policies considered in a single iteration of the algorithm using only one sample set. Like the fitted $Q$-iteration method presented in Section 3.6, LSTD-$Q$ makes no assumptions about the way in which these samples are collected from an actual process.

The complete LSTD-$Q$ algorithm is given by Algorithm 7. The key step of the algorithm consists of updating the matrix $\tilde{\mathbf{A}}$ upon encountering a new sample $(s_t, a_t, r_t, s_{t+1})$.

**Algorithm 7** Pseudocode of the LSTD-$Q$ algorithm (Lagoudakis & Parr, 2003).

*Input:* A set $\mathcal{F} = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i) | i = 1, \ldots, |\mathcal{F}|\}$ of four-tuples, discount factor $\gamma$, policy $\pi$, number of basis functions $k$, and basis functions $\phi$.
*Output:* Vector of weight parameters $\tilde{w}$.
**LSTD_Q($\mathcal{F}, \gamma, \pi, k, \phi$):**
Initialize $\tilde{\mathbf{A}}$ to the $k \times k$ matrix of zeros.
Initialize $\tilde{b}$ to a $k \times 1$ column vector of zeros.
**for** $i = 1$ to $|\mathcal{F}|$ **do**
  $\tilde{\mathbf{A}} \leftarrow \tilde{\mathbf{A}} + \phi(s_t^i, a_t^i) \left(\phi(s_t^i, a_t^i) - \gamma\phi(s_{t+1}^i, \pi(s_{t+1}^i))\right)^\top$
  $\tilde{b} \leftarrow \tilde{b} + \phi(s_t^i, a_t^i) r_t^i$
**end for**
$\tilde{w}^\pi \leftarrow \tilde{\mathbf{A}}^{-1} \tilde{b}$
**return** $\tilde{w}^\pi$

### 3.10.2 Least Squares Policy Iteration

The *Least Squares Policy Iteration* (LSPI) technique makes use of LSTD-$Q$ to determine an improved policy. The algorithm takes as input a set of samples $\mathcal{F}$, a discount factor $\gamma$, a set of basis functions $\phi$ and some initial policy to evaluate $\pi_0$. Repeated calls are made to the LSTD-$Q$ method discussed in the previous section using the chosen set of basis functions and the policy under consideration. This produces an improved weight vector for the selected basis functions at each iteration. The process continues until the difference between the weight vectors produced by successive calls to the LSTD-$Q$ method is significantly small; this is representative of the fact that the policy can no longer be adequately improved. The final policy is returned in the form of a weight vector which, using the basis functions, can be used to produce an approximation of the value function.

The LSPI algorithm is given by Algorithm 8. The major steps involved in policy iteration namely, policy evaluation and improvement, are shown.

**Algorithm 8** Pseudocode of the LSPI algorithm (Lagoudakis & Parr, 2003).

*Input:* A set $\mathcal{F} = \{(s_t^i, a_t^i, r_t^i, s_{t+1}^i) | i = 1, \ldots, |\mathcal{F}|\}$ of four-tuples, discount factor $\gamma$, initial policy $\pi_0$, number of basis functions $k$ and basis functions $\phi$.
*Output:* Policy $\pi$.
**LSPI($\mathcal{F}$, $\gamma$, $\pi_0$, $k$, $\phi$):**
Initialize policy $\pi'$ to initial policy $\pi_0$
**repeat**
  $\pi \leftarrow \pi'$
  $\pi' \leftarrow$ **LSTD_Q($\mathcal{F}$, $\gamma$, $\pi$, $k$, $\phi$)**
**until** $\pi \approx \pi'$
**return** $\pi$.

## 3.11 Benchmark domains

This section describes the benchmarks that are used for testing and comparing algorithmic performance in Chapter 5. Although the purpose of this research is to apply batch reinforcement learning techniques specifically to the problem of HIV drug scheduling, testing on benchmark domains can give us some indication of whether the implementations of the algorithms are working correctly in domains exhibiting similar properties where larger data sets are available.

### 3.11.1 The swing-up acrobot

The swing-up acrobot is a two-link underactuated robot arm analogous to a double-inverted pendulum or acrobat swinging on a bar. Like other underactuated systems, the acrobot has fewer control inputs or actuators than degrees of freedom (Spong, 1998). The first joint corresponding to the shoulder of the acrobot has no actuator; the second joint at the elbow has an actuator and hence exerts a torque (Spong, 1995). The acrobot's state can be described in terms of four continuous state variables: two joint positions, $\theta_1$ and $\theta_2$, and two corresponding joint velocities, $\dot{\theta}_1$ and $\dot{\theta}_2$ (Sutton & Barto, 1998). The aim of the agent is to swing the tip of the pendulum above the first joint by an amount equal to the length of one of the links as quickly as possible (Sutton & Barto, 1998). The actions of the agent can be described in terms of the torque. The torque, $\tau$, applied to the second joint can either have a fixed positive magnitude ($a$ = 1), a fixed negative magnitude ($a$ = -1) or no magnitude at all ($a$ = 0). A reward of -1 is given for every time step, $t$ = 0.05, until the goal state is reached. The swing-up acrobot control problem is illustrated in Figure 3.3 (Boone, 1997). The reader should refer to Appendix A.1 for the system dynamics of the swing-up acrobot.

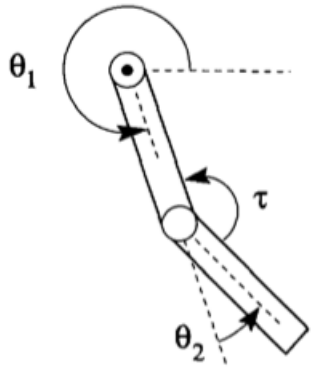



Figure 3.3: The swing-up acrobot (Boone, 1997).

### 3.11.2 The mountain car

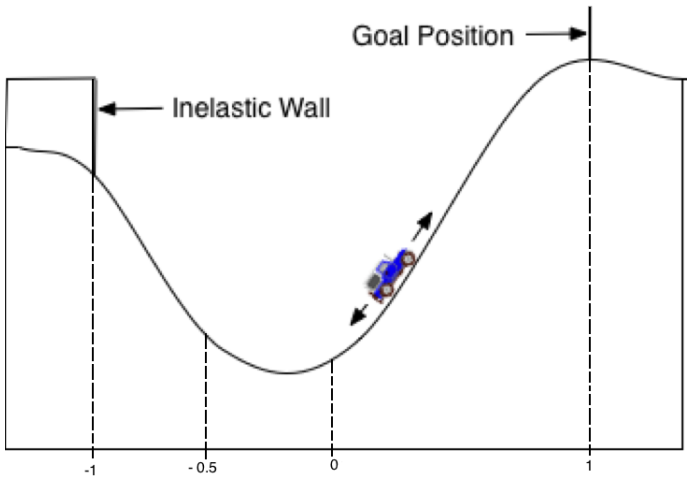



Figure 3.4: Illustration of the mountain car problem (Figure modified from Tanner (2009)).

The mountain car task is a two-dimensional task where an underpowered car must accelerate to the top of one side of a valley in a minimum amount of time. In many cases, the acceleration of the car alone is not enough to move the car directly to the top of the hill so the car has to move in the opposite direction first to gain enough potential energy. Figure 3.4 illustrates the mountain car task (Tanner, 2009).

Within the reinforcement learning context, the car can be modelled as a point travelling on a hill whose state can be expressed as a two-tuple ($p_t$, $v_t$) containing the car's position, $p_t$, and its speed, $v_t$, at some time $t$. At each time step, $t = 0.1$, the agent is given a choice of 2 possible actions namely full throttle ($a_t = 4$) or reverse full throttle ($a_t$ = -4) (Sutton & Barto, 1998). That is, actions are restricted to the values $\{-4, 4\}$. The initial velocity of the car is set to 0 while its position is fixed to -0.5. During the course of learning, the velocity of the car is drawn from the interval [-3, 3]. The agent's goal is to reach the position 1 at the top of the valley. The reader should refer to Appendix A.2 for the system dynamics of the mountain car task.

## 3.12 Conclusion

In short, this chapter has among other things, (i) introduced the reinforcement learning paradigm using MDPs, (ii) discussed the need for batch reinforcement learning for certain domains, (iii) presented the fitted $Q$-iteration algorithm (within the class of value iteration methods) using both extremely randomized trees and neural networks as function approximators, and (iv) presented the least squares policy iteration method for batch learning based on policy iteration. The algorithms discussed in this chapter are the core of our research and have been implemented for learning within the HIV

domain. The next chapter examines exactly how the HIV drug scheduling task has been formulated as an MDP and how the techniques presented here have been applied to this problem.

# Chapter 4

# Research Methodology

## 4.1 Introduction

In this chapter, we present the methodology that was used to conduct this research. In particular, the research problem described in Chapter 1 will be formalized in Section 4.2. Using the material provided from Chapters 2 and 3 as a basis along with the problem definition, we formulate a series of specific research questions in Section 4.3. Thereafter, we provide an outline of the methodology that has been used to answer these questions. Specifically, we discuss the data that is used to test the performance of our implementation in Section 4.5. This data is of two forms: the first is a set of simulated data of a large scale that is used to compare relative performances of the algorithms and test whether the outcomes of the learning techniques can potentially improve patient wellness. The second is a set of real data that has been used to determine whether RL techniques can provide suitable drug scheduling strategies for existing patients and whether these strategies are consistent with those currently in place. Section 4.6 gives a detailed description of how the HIV drug scheduling problem may be formulated as an MDP; we examine MDP construction for both real and simulated data sets. We conclude the chapter in Section 4.7 where we provide a summary of how each batch RL technique was implemented to give the results presented in Chapter 5.

## 4.2 Aim of this research

Current HIV treatment regimens are focused at reducing pill burden for patients in an attempt to reduce side-effects and improve overall patient adherence to drug therapy. Typically, HIV drug therapy involves use of multiple drugs for treatment in order to prevent the development of drug-resistant HIV strains. For these purposes, suitable drug combinations need to be determined. Traditionally, this would involve conducting numerous lengthy and expensive clinical trials. We propose modeling this drug-scheduling problem as an MDP and using RL techniques to determine effective drug combinations. The use of such techniques could be coupled with clinical trial investigations in the future.

## 4.3 Research questions

We can delineate the scope of this research by formulating the following research questions:

1. Is it possible to model the HIV drug scheduling problem as an MDP? If so, how can this modelling be adapted to include newly developed drugs?
2. How do batch RL algorithms compare in terms of outcomes when applied to the HIV drug scheduling problem? Are the drugs recommended by the batch RL algorithms consistent with the strategies currently in place for a particular real patient?
3. Does applying a particular batch RL algorithm improve patient wellness for simulated patients?
4. How do the batch RL algorithms under consideration compare in performance when applied to benchmark domains?

## 4.4 Research methodology

This research has been completed in a number of phases. The initial phase focused primarily on gathering real HIV patient data from hospitals in Johannesburg and preparing this data for testing purposes. The next phase involved simulating HIV patient data using a mathematical model in MATLAB. We discuss the details of this model and the simulated data in the next section. After collecting and generating the necessary data, the HIV drug scheduling problem was formulated as an MDP. This involved constructing a suitable reward function for the real data case and choosing relevant state variables from the data available. This reward function should ultimately reward those situations in which a patient's health shows an adequate improvement under a particular drug combination and penalize those that do not. For the simulated case, we are restricted to constructing the MDP in the same way as Adams et al. (2004). We discuss the details of this MDP construction in Section 4.5.2. Once the construction of the MDP was completed, each of the batch RL techniques under consideration was implemented in MATLAB. A brief description of the implementation is provided in the final section of this chapter. The code was tested on both the real and simulated data sets. These results and the resources used for testing purposes are given in Chapter 5. Finally, data sets for the mountain car and acrobot domains were generated using the dynamics provided in Appendix A.1 and A.2. These data sets were also used to test relative performances of the learning techniques.

## 4.5 Data collection and simulation

This research required us to use both real and simulated data for training and testing purposes. Simulated data can be generated to determine algorithmic performance on

large scales. We can also apply the policies learned by the algorithms back to the model used for data generation to generate more samples and see whether these samples exhibit any improvement. That is, we can determine whether the policies learned by the algorithms result in improved health among the patients generated according to the model. Availability of real data is limited hence resulting in a much smaller sample set. However, this real data contains much more specific drug information for each patient. This information is crucial for determining the suitability of specific drug combinations in a real context rather than a generalized prediction of what combinations of drug classes can be useful. We can draw comparisons between the drug combinations suggested by the algorithms and the drug combinations currently in place. The subsequent sections discuss how data has been collected and simulated for this research.

### 4.5.1 HIV patient data collection

We would like to express our thanks to the staff of the Charlotte Maxeke Johannesburg Academic Hospital and Dr Minakshi Jivan for assisting us in the collection of patient data. All patient data has been collected having obtained ethics clearance and is in compliance with the rules stipulated by the Human Research Ethics Committee at the University of the Witwatersrand.[1]

The data of 250 HIV-infected patients comprising of $CD4^+$ counts, viral loads and drug therapy was collected from a period of up to ten years to produce a set of 2 560 samples in total. Patient selection was entirely random and based on limited data sources. Data preprocessing and reasons for examining these specific variables are discussed in Section 4.6.2 where we formulate an MDP for the real-world HIV drug scheduling problem.

### 4.5.2 Simulating HIV patient data using a mathematical model

We introduce the mathematical model we used to generate artificial HIV data for experimentation. In general, modelling HIV infection and its impact on the immune system requires many factors to be considered; despite the existence of complex interactions between many biological components, only a small subset of these biological indicators can be chosen for modelling. For the purposes of this research generating suitable HIV patient data requires using a model of the HIV infection dynamics that includes patient wellness indicators that adequately describe a patient's condition at a particular time. These indicators can in turn be used to model the HIV domain as an MDP and constitute the state space for the reinforcement learning problem. Perhaps the most obvious choice of a wellness indicator for an HIV-infected individual is the viral load; this is a direct measure of the number of HIV particles contained in the blood. The HIV viral load is measured as the number of copies of RNA per millilitre of blood. Other indicators that can be used to determine the health of an HIV-infected individual include the numbers of infected and uninfected $CD4^+$ cells and macrophages. Immune

[1] These rules specify the maintenance of strict patient confidentiality

effectors such as $CD8^+$ cells, can be used as an indicator of the body's immune response to the presence of infected T-cells and the pathogen itself. Ideally, a suitable HIV data generation model should at minimum take these wellness indicators into account.

A model used to simulate HAART in HIV-infected patients should also include the action of commonly used antiretrovirals and allow for the use of multiple drug combinations at each time. Ideally, we wish to use a model that accounts for specific drug combinations such as the popular EFV, FTC, tdf combination (now being sold as Atripla), however this is unrealistic, so we settle for a model that simulates action of the major classes of antiretroviral drugs instead. Unfortunately this means that the action of one member of a particular drug class is considered to be identical to another member of the same class which is not always the case and is something we would ideally like to determine.

The model we used to simulate HIV data under HAART is based on work from Callaway & Perelson (2002). The model itself may be found in Adams et al. (2004) and Adams et al. (2005). While it was originally constructed to take into account possible treatment interruptions in HAART, here we ignore the case when patients are completely removed from medication. The complete dynamics of the model are described by the set of Equations 4.5.1 - 4.5.6.

$$\frac{\mathrm{d}T_1}{\mathrm{d}t} = \lambda_1 - d_1 T_1 - (1-\epsilon_1) k_1 V T_1 \tag{4.5.1}$$

$$\frac{\mathrm{d}T_2}{\mathrm{d}t} = \lambda_2 - d_2 T_2 - (1-f\epsilon_1) k_2 V T_2 \tag{4.5.2}$$

$$\frac{\mathrm{d}T_1^*}{\mathrm{d}t} = (1-\epsilon_1) k_1 V T_1 - \delta T_1^* - m_1 E T_1^* \tag{4.5.3}$$

$$\frac{\mathrm{d}T_2^*}{\mathrm{d}t} = (1-f\epsilon_1) k_2 V T_2 - \delta T_2^* - m_2 E T_2^* \tag{4.5.4}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = (1-\epsilon_2) N_T \delta (T_1^* + T_2^*) - cV - \left[(1-\epsilon_1)\rho_1 k_1 T_1 + (1-f\epsilon_1)\rho_2 k_2 T_2\right] V \tag{4.5.5}$$

$$\frac{\mathrm{d}E}{\mathrm{d}t} = \lambda_E + \frac{b_E (T_1^* + T_2^*)}{(T_1^* + T_2^*) + K_b} E - \frac{d_E (T_1^* + T_2^*)}{(T_1^* + T_2^*) + K_d} E - \delta_E E \tag{4.5.6}$$

Here, $T_1$ ($T_1^*$) denotes the number of non-infected (respectively infected) $CD4^+$ T-lymphocytes (in cells/$ml$), $T_2$ ($T_2^*$) the number of non-infected (respectively infected) macrophages (in cells/$ml$), $V$ the number of free HI viruses (in copies/$ml$) and $E$ the number of cytotoxic T-lymphocytes (in cells/$ml$). The values of the various parameters of the model are taken directly from Adams et al. (2004) and Bonhoeffer et al. (2000). These are listed in Table 4.1.

| Parameters | Value | Units | Description |
|---|---|---|---|
| $\lambda_1$ | 10 000 | $\frac{cells}{ml.day}$ | production rate of $CD4^+$ cells |
| $d_1$ | 0.01 | $\frac{1}{day}$ | death rate of $CD4^+$ cells |
| $\epsilon_1$ | $\in [0, 1)$ | - | efficacy of RTI |
| $\epsilon_2$ | $\in [0, 1)$ | - | efficacy of PI |
| $k_1$ | $8.0 \times 10^{-7}$ | $\frac{ml}{virions.day}$ | infection rate of $CD4^+$ cells |
| $\lambda_2$ | 31.98 | $\frac{cells}{ml.day}$ | production rate of macrophages |
| $d_2$ | 0.01 | $\frac{1}{day}$ | death rate of macrophages |
| $f$ | 0.34 | - | reduction of treatment efficacy for macrophages |
| $k_2$ | $1.0 \times 10^{-4}$ | $\frac{ml}{virions.day}$ | infection rate of macrophages |
| $\delta$ | 0.7 | $\frac{1}{day}$ | death rate of infected cell |
| $m_1$ | $1.0 \times 10^{-5}$ | $\frac{ml}{cells.day}$ | immune-induced clearance rate for $CD4^+$ cells |
| $m_2$ | $1.0 \times 10^{-5}$ | $\frac{ml}{cells.day}$ | immune-induced clearance rate for macrophages |
| $N_T$ | 100 | $\frac{virions}{cell}$ | virions produced per infected cell |
| $c$ | 13 | $\frac{1}{day}$ | natural death rate of virus |
| $\rho_1$ | 1 | $\frac{virions}{cell}$ | average number of virions infecting a $CD4^+$ cell |
| $\rho_2$ | 1 | $\frac{virions}{cell}$ | average number of virions infecting a macrophage |
| $\lambda_E$ | 1 | $\frac{cells}{ml.day}$ | production rate of immune effector/cytotoxic T-cell |
| $b_E$ | 0.3 | $\frac{1}{day}$ | maximum birth rate for cytotoxic T-cell |
| $K_b$ | 100 | $\frac{cells}{ml}$ | saturation constant for cytotoxic T-cell birth |
| $d_E$ | 0.25 | $\frac{1}{day}$ | maximum death rate for cytotoxic T-cell |
| $K_d$ | 500 | $\frac{cells}{ml}$ | saturation constant for cytotoxic T-cell death |
| $\delta_E$ | 0.1 | $\frac{1}{day}$ | natural death rate of cytotoxic T-cells |

Table 4.1: Parameters used in Equations 4.5.1 – 4.5.6 (Adams et al., 2004).

The model consists of two populations of target cells representing $CD4^+$ T-cells and macrophages respectively (Adams et al., 2004). In particular, the classes of RTIs and PIs are modelled. The manner in which NRTIs and NNRTIs operate is largely similar hence they are grouped together as one class of drug. Like most other mathematical models of HIV-infection under HAART, the model includes parameters for the drug efficacy of each class of ARV under consideration. These parameters are $\epsilon_1$ and $\epsilon_2$. They describe how effective the RTI and PI classes of drugs are in reducing infection respectively (Adams et al., 2004). The model assumes that the RTI class of drugs is more effective in the $CD4^+$ population of cells than in macrophages where the efficacy is reduced by a factor of $f$, $f \in [0, 1]$. The PI class of drugs is only included in equations describing the change in the viral load under HAART, since these drugs operate by directly interfering with the formation of viral proteins that are necessary for viral production. Infected T-cells result from direct interaction between uninfected T-cells and free virus particles; these cells are removed from the system via natural death or through the action of cytotoxic T-cells. The model assumes both T-cells and macrophages have the same death rates, $d_1 = d_2$. The immune effector cells are produced in response to the presence of infected

cells and existing immune effectors. The action of these immune effector cells triggers lysing of infected T-cells and macrophages which results in their removal from the system of equations at the rates of $m_1$ and $m_2$ respectively. The rate at which the virus infects both types of cells is assumed to be different and is given by the parameters $k_1$ and $k_2$ respectively. Free virus particles are produced by both infected macrophages and infected T-cells; the model assumes these are produced at the same rate.

Adams et al. (2004) demonstrate that when both $\epsilon_1$ and $\epsilon_2$ are zero, the dynamic model has three physical equilibrium points where all the variables are non-negative. These equilibria are:

$$(T_1, T_2, T_1^*, T_2^*, V, E) = (10^6, 3\ 198, 0, 0, 0, 10), \tag{4.5.7}$$

$$(T_1, T_2, T_1^*, T_2^*, V, E) = (967\ 839, 621, 76, 6, 415, 353\ 108), \tag{4.5.8}$$

$$(T_1, T_2, T_1^*, T_2^*, V, E) = (163\ 573, 5, 11\ 945, 46, 63\ 919, 24). \tag{4.5.9}$$

Note that Equation 4.5.7 is an unstable equilibrium point representing an uninfected individual; Equations 4.5.8 and 4.5.9 are the stable equilibrium points representing an infected individual. Specifically, Equation 4.5.8 represents an individual with good immune control over the virus. This individual has a low viral load and high $CD4^+$ and $CD8^+$ counts; a patient is considered to be in a healthy steady state or stationary equilibrium here. Equation 4.5.9 represents an individual in an unhealthy steady state whose viral load is considerably elevated and T-cells are in short-supply in the absence of treatment.

Using this dynamic model, we generate HIV data in the same way as Ernst et al. (2006). That is, we assume the variables $T_1$, $T_2$, $T_1^*$, $T_2^*$, $V$ and $E$ are measured every five days and each patient is monitored for a period of 1 000 days. We consider 50 patients in the infected unhealthy steady state and randomly select medication for the patient every five days. The values of $\epsilon_1$ and $\epsilon_2$ are modified according to whether the associated class of drug is in use. Using Euler's method to solve the system of equations, the resulting T-cell counts, macrophage counts and viral load are recorded. In this way, each patient produces 200 samples. For a total of 50 patients, 10 000 samples are generated. In Chapter 5, we consider a number of experiments using this simulated data. Some of these experiments require larger sample sets to be used, or require us to generate more samples of data iteratively after applying RL to the problem domain; we discuss this iterative data generation procedure in Chapter 5 where necessary.

## 4.6 Modelling the HIV drug scheduling problem as an MDP

In this section we formulate the HIV drug scheduling problem as an MDP to which the RL techniques under consideration can be applied for learning purposes. We distinguish between different MDPs for the simulated data and real data cases respectively as some of the variables modelled by Equations 4.5.1 - 4.5.6 are not directly observable in reality.

### 4.6.1 An MDP formulation for the simulated case

For the case of simulated data, we have modelled the HIV drug scheduling problem as a MDP in a manner almost identical to Ernst et al. (2006). Here our state space $\mathcal{S}$ consists of six state variables corresponding to the number of healthy CD4$^+$ T-cells ($T_1$), healthy macrophages ($T_2$), infected CD4$^+$ T-cells ($T_1^*$), infected macrophages ($T_2^*$), free virus particles ($V$) and cytotoxic T-cells ($E$) respectively. Our action space $\mathcal{A}$ is comprised of three actions representing the possible drug combinations available. That is, at each time step, we consider one of three on-off drug combinations. These are:

1. Use both RTI and PI class drugs
2. Use only an RTI
3. Use only a PI

Note that we have ignored the case when both classes of drugs are turned off since this is the equivalent of cycling a patient off drugs completely and constitutes a treatment interruption. As mentioned in Section 2.6.3 of Chapter 2, intermittent therapy has proven dangerous under the SMART trial. Unfortunately we are limited to examining only two classes of drugs here because of the manner in which HIV-infection has been modelled.

Like Ernst et al. (2006), we are looking for those drug strategies that minimize the cumulative instantaneous costs of taking various drug combinations over an infinite time horizon. Here the instantaneous cost of taking a particular drug combination at a time $t$ is calculated according to

$$c(s_t, a_t) = QV_t + R_1{\epsilon_{1t}}^2 + R_2{\epsilon_{2t}}^2 - SE_t \tag{4.6.1}$$

where $Q = 0.1$, $R_1 = 20\ 000$, $R_2 = 2\ 000$ and $S = 1\ 000$. The variables $V_t$ and $E_t$ represent the number of free virus particles (copies/ml) and cytotoxic T-cells (copies/ml) at time $t$ respectively. The parameters $\epsilon_{1t}$ and $\epsilon_{2t}$ correspond to the efficacies of the RTI and PI under consideration at time $t$ respectively. When an RTI is being used for treatment, $\epsilon_{1t} = 0.7$; when a PI is being used, $\epsilon_{2t} = 0.3$; otherwise, these values are 0. Intuitively, a combination of drugs that results in a raised viral load increases the cost function; conversely, taking a combination of drugs that results in a better immune effector response, decreases the cost function. The cost function may be viewed as an inverse reward function. That is, a higher cost would be the equivalent of a lower reward in the context of RL and would result in the agent being penalized for taking the associated action. The interested reader should refer to Adams et al. (2004) and Adams et al. (2005) for the precise reasoning behind this cost function for the system of Equations 4.5.1 - 4.5.6.

### 4.6.2 An MDP formulation for the real data case

Moving from the simulated case to dealing with real data sets requires us to consider a number of issues pertaining to the data available. Firstly, when simulating how HIV

interacts with a group of patients, each patient is assumed to have the same dynamics. When using real data, this is not necessarily the case since the dynamics may vary significantly from patient to patient. Reasons for differing dynamics can often be attributed to existence of different HIV types and strains or differences in patients' immune systems. A possible means of addressing this issue is to consider including an attribute such as the HIV type or specifics regarding a patient's case in the state space. In our research, we have tried to eliminate differences that result from varying types of HIV by considering only those patients infected by HIV-1. We restrict our study to HIV-1 since our sources of HIV data are limited and HIV-2 is not prevalent in South Africa. However, because of the large number of clades in HIV-1, this is not necessarily the best solution as variation between individual strains of HIV is still unaccounted for. In an ideal situation, it might be useful to consider adding a state variable for the HIV strain to the Markov model. With restrictions to the data we have access to, this was not possible.

One of the issues associated with using real data is defining a time step for modelling an MDP. In the simulated case, we can assume that the values of the state variables determined according to the system of ODEs 4.5.1 - 4.5.6 are recorded at regular time steps that are equidistant from one another. For real data, patients are typically required to visit the hospital for blood tests every six months however, this does not always happen. To overcome this issue, we have defined a single time step to be a period of six months from the previous date of visit and interpolated the values of the state variables from the data where regular six-month data was unavailable. In addition, we have normalized all state variables using a logarithmic transformation. Reasons for normalizing the real data are based on the fact that it is easier to define a reward function on reduced data range. This problem does not exist when using simulated data since we use the same cost function as defined by Ernst et al. (2006). There are also several cases in the real data set where the values of the state variables are larger than those in the simulated data set, despite being expressed in the same units. Normalization is useful for these cases. The values of the state variables are all plotted on a logarithmic scale in both the experimentation using simulated and real data sets. This is for easier visualization and interpretation.

Perhaps the most significant challenge that arises when using real data is the problem of partial observability (Ernst et al., 2006). In the simulated case, we assumed that all the state variables can be observed and recorded directly with current technology. In reality however, this is not the case: current blood tests conducted by hospitals for HIV-infected individuals do not measure macrophage counts and existing technology cannot distinguish between infected and uninfected $CD4^+$ T-lymphocytes; in addition, $CD8^+$ lymphocytes are only recorded once in a few years if at all. With these limitations in available data, we are forced to consider only the $CD4^+$ count and viral load data when constructing a state space for the MDP.

Having discussed the major difficulties when using real data for training, we can formally define how we have constructed an MDP for the real data case. Our state space $\mathcal{S}$ consists of a patient's $CD4^+$ T-cell count ($T$) and viral load ($V$) normalized by

applying a logarithmic transform. Unlike when using simulated data that was generated according to a model that could not take into account specific drugs, for real data we consider seven major drugs that are currently used in South Africa in ARV treatment. These are EFV, 3TC, tdf, d4T, LPV/r, NVP and AZT.[2] Reasons for restricting our study to these seven drugs are based on the fact that these drugs are widely available in the country and are commonly used; another major reason is the real patient data we have accessed makes use of combinations of these drugs. We can represent the seven drugs under consideration as a seven-tuple vector of binary values where each value corresponds to whether the associated drug is being used at a certain time or not. This seven-tuple vector represents the action taken by a patient at a particular time. Theoretically, our action space $\mathcal{A}$ consists of 128 possible actions based on the drugs being used in a multi-drug combination. Practically, certain combinations of drugs are not taken together because of the side-effects they may produce when combined, and since this research does not consider the use of STIs, we do not allow for the possibility of a patient not receiving any treatment. By taking this into consideration, we can eliminate a few actions from the action space. While we have only considered the use of these seven drugs, it is possible to extend the action space to include more drugs or newly developed drugs for a clinical trial setting. In the case of clinical trials, the patient data obtained from a particular phase of the trials can be used to train the batch RL algorithms we consider in this research; here, the action space would include the developed drug. Results from using batch RL algorithms between phases of a clinical trial could provide medical researchers with insight into the efficacy of the new drug(s) fairly quickly before the subsequent phase begins. This is particularly useful since phases of a clinical trial are usually very lengthy and typically last a number of years. We are aware that the action space is exponential in the number of drugs considered, however there are ways in which this problem can be overcome. Subtle variants of the fitted $Q$-iteration method exist that demonstrate how the algorithm can be extended to large and/or continuous action spaces. These procedures are discussed by Antos et al. (2007) and briefly by Ernst et al. (2005). The same can be said for LSPI (see Pazis & Lagoudakis (2011) for details).

Like Zhao et al. (2009), we use a composite function to define the reward the agent receives at each time step during learning. This reward function is comprised of two functions, $r_{t1}$ and $r_{t2}$, based on the changes observed in a patient's T-cells and viral loads respectively. We penalize an increase in the viral load or decrease in the $CD4^+$ count. We give a large positive reward when a patient's viral load is below detectable limits i.e. less than 40 copies/ml ($V_t$ = -1). If a patient's T-cell count neither decreases nor increases, we do not give a reward since there is no change in immune response. However, if the viral load stays the same, we give a penalty of -1. This assists in making the algorithms learn to choose those actions which improve the viral load as quickly as possible. The values for $r_{t1}$ and $r_{t2}$ are given by,

[2]We consider LPV and r together since LPV must be taken in conjunction with r.

$$r_{t1} = \begin{cases} 5, & \text{if } T_t > T_{t-1} \\ -5, & \text{if } T_t < T_{t-1} \\ 0, & \text{otherwise} \end{cases} \tag{4.6.2}$$

$$r_{t2} = \begin{cases} 5, & \text{if } V_t < V_{t-1} \text{ and } V_t \neq -1 \\ -5, & \text{if } V_t > V_{t-1} \text{ and } V_t \neq -1 \\ 25, & \text{if } V_t = -1 \\ -1, & \text{otherwise.} \end{cases} \tag{4.6.3}$$

Here, the time step $t$ is a period of 6 months between patient visits to the hospital. The final reward is given by the following equation:

$$r_t = r_{t1} + r_{t2}. \tag{4.6.4}$$

In this form, we place equal emphasis on both state variables. It would, however, be possible to introduce weights to this sum if we wanted one variable to have more influence on the actions selected than the other.

## 4.7 Implementation of batch reinforcement learning techniques

For this research, all algorithms and data generation methods have been implemented in MATLAB. For comparative purposes all learning algorithms need to be implemented in the same language. We chose MATLAB primarily because of its Neural Network Toolbox that allows us to create and customize neural networks easily for the neural fitted $Q$-iteration algorithm. Data visualization is also fairly straightforward since MATLAB has an extensive set of graphics options available to the user. Furthermore, matrix manipulation is important for methods such as LSPI; MATLAB is particularly good at this.

For the fitted $Q$-iteration with extra trees algorithm, we chose to represent each tree as a four-tuple $(f_1, f_2, f_3, f_4)$ where the first element of the tuple corresponds to the attribute selected at the root node to split on; the second element corresponds to the value of the split, and the third and fourth elements represent the index of the left and right subtrees respectively. Each of the subtrees produced were represented in the same way. In this way, a tree could easily be represented recursively.

Coding the neural fitted $Q$-iteration method consisted of executing the standard fitted $Q$-iteration algorithm using neural networks to perform the regression step. Our specific implementation used the resilient backpropagation algorithm (RPROP) (Riedmiller & Braun, 1993) to train the network. RPROP is an improved version of classical backpropagation method that is used to train neural networks. The main advantage of using RPROP is the performance gain. This gain is largely a result of the manner in which weights are updated during training: weights are updated by combining the current gradient and the gradient from the previous step in training (Hatzigeorgiou &

Megraw, 2006). Network structures were varied according to the specific experiment performed.

Implementation of the LSPI algorithm mainly involved coding methods for calculating suitable basis functions for the problem at hand. We have used Gaussian radial basis functions which we mention in the next chapter. These basis functions were used by the LSPI algorithm to perform standard matrix multiplication and derive a suitable policy for the domain of interest. While it is possible for tilings to be used for the same purposes, preliminary testing revealed that Gaussian radial basis functions performed better across all the domains we considered.

## 4.8 Conclusion

In this chapter, we presented the methodology that was carried out to complete this research. This involved formulating the drug scheduling task as an MDP, collecting and generating patient data, cleaning the data, and applying the learning techniques under consideration to these data sets. The next chapter presents the results obtained from each of the experiments conducted and a discussion of what these results mean for drug scheduling in the HIV domain.

# Chapter 5

# Results and Discussion

## 5.1 Introduction

In this chapter we present the major results of this research. We begin by providing the specifications of the machine we have used for testing purposes in Section 5.2. In Section 5.3, we examine the performance of each of the batch reinforcement learning techniques discussed in Chapter 3 on the benchmark domains of mountain car and the swing-up acrobot. Specifically the results from the benchmark domains can give us some insight as to how the algorithms perform on well-behaved domains. In Section 5.3, we apply the same techniques to the set of simulated data that was discussed in Chapter 4. We use each of the policies determined on this set of simulated data to generate more patient samples for learning in Section 5.4; these samples allow us to ascertain whether the learned policies result in improved patient outcomes for the simulated case. When experimenting using real data, it is not possible to implement the policies suggested by the algorithms on real patients and observe the outcomes. For these reasons, we perform a match test, to observe how many times the suggested policy matches those policies currently in place from the patient data available. These results are provided in Section 5.5. Section 5.6 concludes the chapter by discussing how the results obtained in this chapter can be used to answer the research questions formulated in Chapter 4.

## 5.2 System specifications

We have used MATLAB 7.11 (R2010b) to implement all of our code for this research.[1] All experimentation has been performed on a MacBook Pro '11 with the following specifications:

- Mac OS X Lion Version 10.7.5 (11G63b)
- 2.66 Ghz Intel Core 2 Duo
- 4GB 1067 MHz DDR3

[1] All the code written for this research is available upon request at sonali.parbhoo@students.wits.ac.za.

- 3MB Cache

## 5.3 Benchmark domain experimentation

In this section, we present the results of the performances of each batch RL technique when applied to the benchmark domains of the mountain car and acrobot task respectively. Specifically, we examine how the algorithms compare in terms of their run times for data sets of differing sizes as well as their outcomes in terms of the policies they determine.

### 5.3.1 Performance assessment metrics

In order to compare the performances of each batch RL technique under consideration when applied to the benchmark domain tasks, we require certain metrics to assess the quality of the policies determined. In our case, we use (i) the number of times a successful optimal policy[2] is generated and (ii) how quickly the learning task at hand is fulfilled under a successful optimal policy. The former requires us to run a number of trials on different sample sets and determine what percentage of these trials produce a policy that accomplishes the task. The latter requires us to assess how many steps are required to fulfill a certain task under a successfully computed optimal policy. The learning technique that reaches the goal state most often and/or requires the smallest number of steps to reach the goal state exhibits the best performance overall.

### 5.3.2 Comparison of algorithmic outcomes for mountain car

In this experiment we compare the outcomes of each batch RL technique under consideration when applied to the mountain car task. Specifically, we divide the experiment into three parts. Part (i) is based on comparing the run time performances of each algorithm when applied to sets of mountain car samples of varying sizes. We acknowledge using code by Fonteneau (2009) for generating these mountain car samples. In particular, we have generated sample sets of sizes 5 000, 10 000, 15 000, 20 000, 25 000 and 30 000 respectively. Each episode within the set of samples begins at the initial state $(p, v)$ = (-0.5, 0) representing a stationary car at the bottom of a hill; an episode ends once the car reaches the top of the hill or if either $p <$ -1 or $v$ leaves the interval [-3, 3]. We execute 50 iterations of each learning technique and measure the time it takes for each method to reach completion. This is repeated over a set of five trials from which we can obtain an average run time. While run time is not an indicator of the quality of the solution obtained, it is of practical significance if the batch of samples available is quite large. It can also give us an indication about the performance of each method relative to the training time required. In part (ii), we determine the optimal policy from the start state at the bottom of the hill for each algorithm using a sample

[2]Hereafter, we use the term optimal policy to refer to the approximate optimal policy after $N$-steps, $\widehat{\pi}^*_N$.

set of 10 000 samples after 50 iterations and illustrate these results. In part (iii), we first execute each algorithm on 30 sample sets each containing 10 000 samples. For each of the algorithms, we determine an optimal policy after 50 iterations. From the optimal policies determined, we calculate the percentage of policies that are successful in reaching the goal state. Thereafter, we select 10 sample sets for which all three learning methods are able to produce successful policies. We record the number of steps taken to reach the goal under $\widehat{\pi}^*_{50}$ from the learning techniques across each of the 10 sample sets. Each part of this experiment has been performed using sample sets that are generated according to a random policy. We use the parameters $M = 50$ and $n_{min} = 2$ for fitted $Q$-iteration with extra trees. A neural network consisting of 4 input nodes, a layer of 50 hidden nodes and 1 output node are used for neural fitted $Q$-iteration.This neural network structure has been chosen after significant testing. For LSPI, we use 100 randomly centered Gaussian radial basis functions with $\sigma^2 = 5$. We present the results for (i) - (iii) hereafter.

**(i) Run time comparison of learning techniques using sample sets of varying sizes**

Tables 5.1 - 5.3 show the running times of the three batch RL techniques under consideration for sample sets of varying sizes. The average of these times are computed over 5 trials.

| $\lvert\mathcal{F}\rvert$ | 1 | 2 | 3 | 4 | 5 | Average time (mins) |
|---|---|---|---|---|---|---|
| 5 000 | 12.38 | 15.81 | 16.19 | 16.73 | 14.62 | 15.15 |
| 10 000 | 55.76 | 51.87 | 46.38 | 49.29 | 52.71 | 51.20 |
| 15 000 | 55.83 | 56.92 | 48.27 | 44.02 | 62.50 | 53.51 |
| 20 000 | 61.46 | 48.97 | 57.91 | 63.82 | 67.25 | 59.89 |
| 25 000 | 66.70 | 68.79 | 89.12 | 59.32 | 64.81 | 69.75 |
| 30 000 | 69.71 | 63.14 | 78.72 | 78.93 | 83.95 | 74.89 |

Table 5.1: Run times of neural fitted $Q$-iteration on mountain car sample sets of varying sizes over 5 trials.

| $\lvert\mathcal{F}\rvert$ | 1 | 2 | 3 | 4 | 5 | Average time (mins) |
|---|---|---|---|---|---|---|
| 5 000 | 98.72 | 102.49 | 116.27 | 99.36 | 108.58 | 105.08 |
| 10 000 | 126.34 | 120.10 | 132.74 | 121.67 | 121.23 | 121.42 |
| 15 000 | 136.00 | 138.21 | 131.29 | 131.03 | 137.96 | 134.90 |
| 20 000 | 138.71 | 142.11 | 139.43 | 139.82 | 139.41 | 139.67 |
| 25 000 | 148.33 | 141.71 | 144.00 | 148.39 | 148.52 | 146.19 |
| 30 000 | 145.67 | 152.71 | 148.88 | 149.72 | 147.19 | 148.83 |

Table 5.2: Run times of fitted $Q$-iteration using extra trees on mountain car sample sets of varying sizes over 5 trials.

| $\|\mathcal{F}\|$ | 1 | 2 | 3 | 4 | 5 | Average time (mins) |
|---|---|---|---|---|---|---|
| 5 000 | 3.18 | 3.19 | 3.24 | 3.11 | 3.07 | 3.16 |
| 10 000 | 6.36 | 6.42 | 6.79 | 7.02 | 6.21 | 6.56 |
| 15 000 | 10.11 | 9.59 | 9.71 | 10.28 | 9.02 | 9.74 |
| 20 000 | 12.45 | 12.92 | 11.63 | 12.22 | 11.73 | 12.19 |
| 25 000 | 18.31 | 18.76 | 19.20 | 16.72 | 18.19 | 18.24 |
| 30 000 | 24.26 | 24.19 | 24.18 | 23.72 | 24.84 | 24.24 |

Table 5.3: Run times of LSPI on mountain car sample sets of varying sizes over 5 trials.

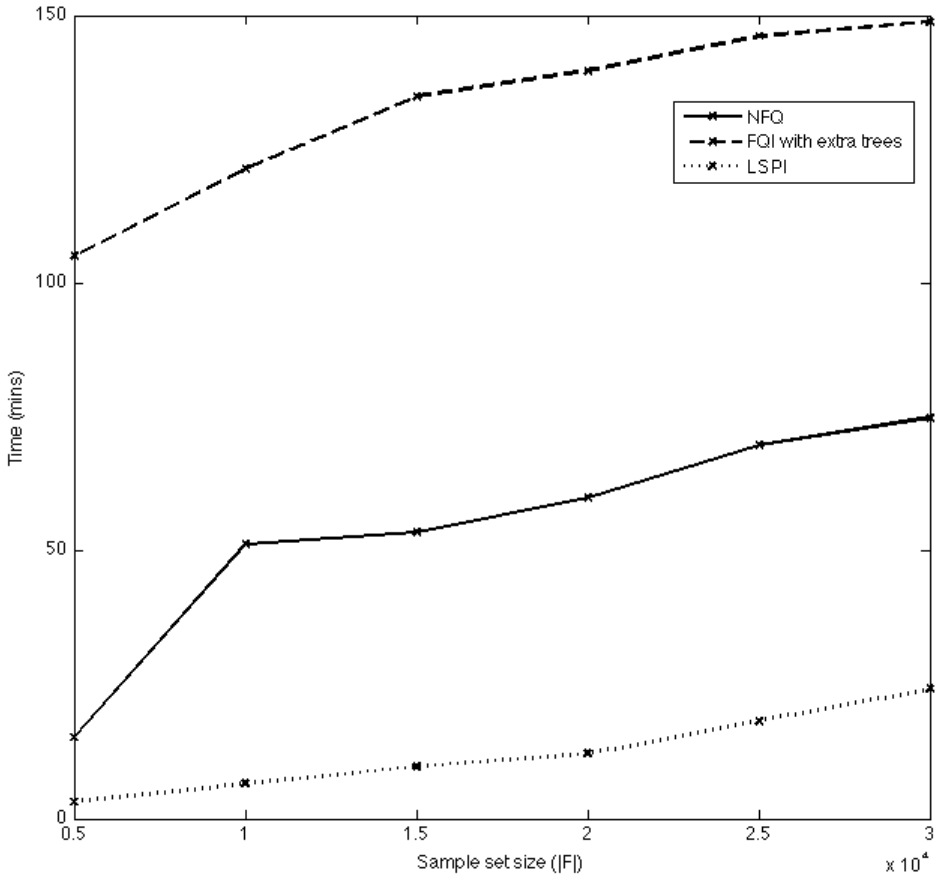



Figure 5.1: Average run times of each algorithm when applied to the mountain car task using sample sets of varying sizes.

The run times of each algorithm increase with an increase in the size of the sample sets used for learning as is expected. For both versions of fitted $Q$-iteration, the majority of the time is spent on the regression step to determine a $Q$-function of approximation.

That is, for the fitted $Q$-iteration algorithm with extra trees, we spend the most time actually constructing the ensembles of trees recursively; for neural fitted $Q$-iteration, most time is spent training the neural network. The major computational step for LSPI involves computing the basis functions for each pair state $(p, v)$. Overall, the extra trees implementation is extremely slow in comparison to the other two methods. LSPI outperforms the other two methods significantly in terms of its run time. This is based on the fact that once the basis function calculation step of the algorithm is complete, the $Q$-values can be determined from a straightforward matrix multiplication step. Figure 5.1 shows a summary of these times for each algorithm as the size of the sample set used for learning increases. While changing the parameters $M$ and $n_{min}$ would result in significantly different performances, Ernst et al. (2005) demonstrate that $M = 50$ is large enough to ensure that the accuracy of the models produced for this domain cannot be improved further by increasing the number of trees. Hence we have chosen the same parameter values for this experiment.

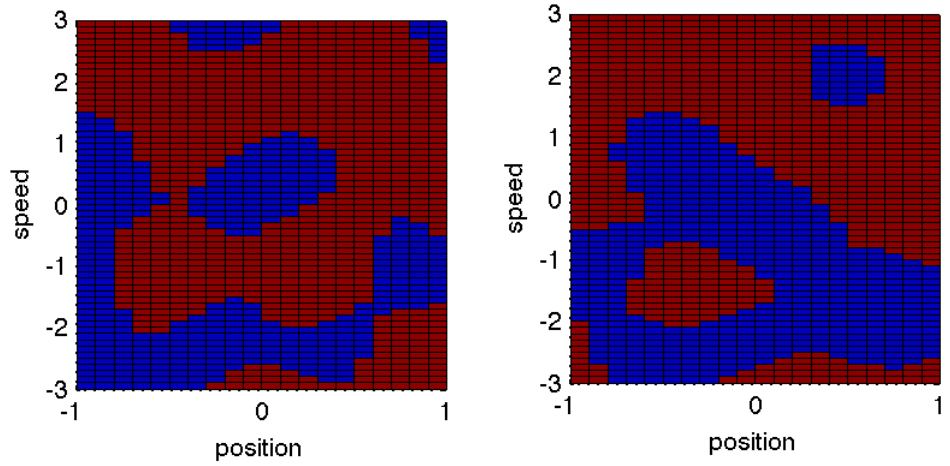
**(ii) Optimal policies determined by batch RL techniques for mountain car task.**

In this part of the experiment, we determine an optimal policy from the start state $(p, v)$ = (-0.5, 0) for the mountain car task. To do so, we run 50 iterations of each learning technique on a sample set of 10 000 randomly generated mountain car samples. Figures 5.2, 5.3 and 5.4 give a representation for the optimal policy after a different number of iterations, $N$. The points in blue correspond to those states where $\widehat{Q}_N((p, v), -4) > \widehat{Q}_N((p, v), 4)$; the points in red represent states where $\widehat{Q}_N((p, v), -4) < \widehat{Q}_N((p, v), 4)$; points in green correspond to states where $\widehat{Q}_N((p, v), -4) = \widehat{Q}_N((p, v), 4)$.

We observe that for the particular sample set used, each algorithm is able to generate a successful policy however, the policies obtained differ slighty. Results from NFQ show that the policy changes quite remarkably between iterations (see Figure 5.2 (a) - (e)). The policies obtained using fitted $Q$-iteration with extra trees and LSPI tend to stabilize and change less frequently at some point (see Figures 5.3 and 5.4 (a) - (e)). The trajectory obtained from applying the optimal policy from LSPI to the start state $(p, v)$ = (-0.5, 0) is longer than the trajectories obtained from applying fitted $Q$-iteration and neural fitted $Q$-iteration to the same sample set. Both versions of fitted $Q$-iteration are able to determine similar optimal policies that are more efficient since they require fewer and smaller steps to reach the goal state. For the particular sample set used here, neural fitted $Q$-iteration produces the most effective trajectory to the goal state using only 21 steps in comparison to the 24 steps and 40 steps required by fitted $Q$-iteration with extra trees and LSPI respectively. This is shown in Figure 5.2 (f).

We note that applying the learning techniques to this sample set has produced a successful optimal policy for each case. However, since an episode in the mountain car task ends either when the goal state is reached, or when $p <$ -1 or $v$ leaves the interval [-3, 3], it is possible to obtain an unsuccessful policy when using other sample sets. Figure 5.5 shows an example of a trajectory produced by an unsuccessful policy where

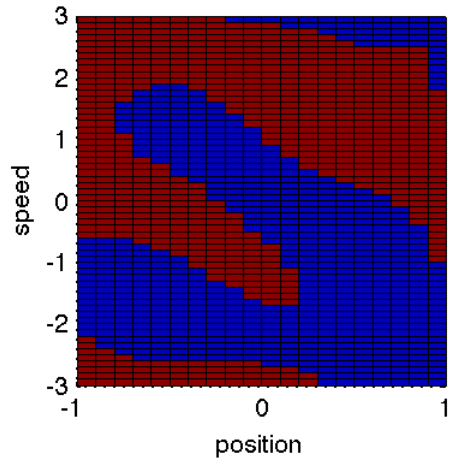
the position of the car, $p < -1$. In addition, it is possible that one learning method outperforms the others on a particular sample set, but not in general. For these reasons, we perform part (iii) of this experiment.



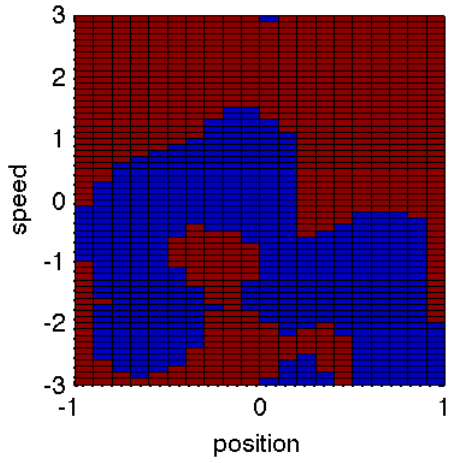

(a) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{10}(s,a)$



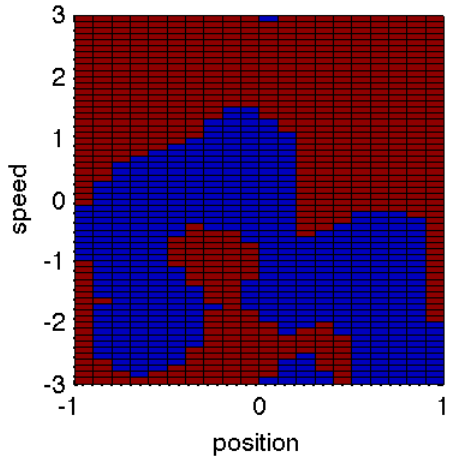

(b) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{20}(s,a)$



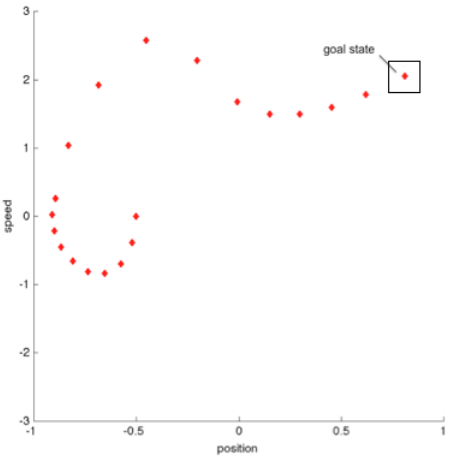

(c) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{30}(s,a)$




(d) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{40}(s,a)$

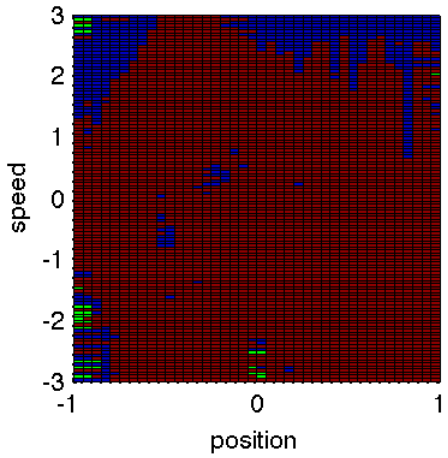



(e) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{50}(s,a)$

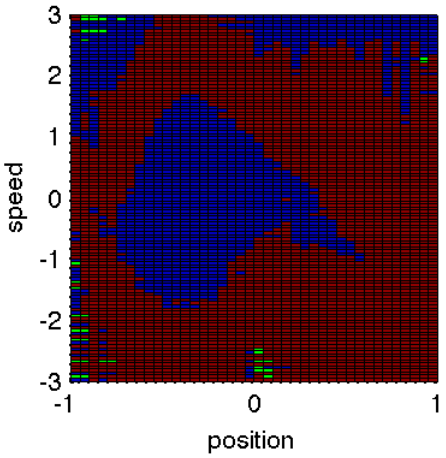



(f) Trajectory from initial state $s_0 = (-0.5, 0)$

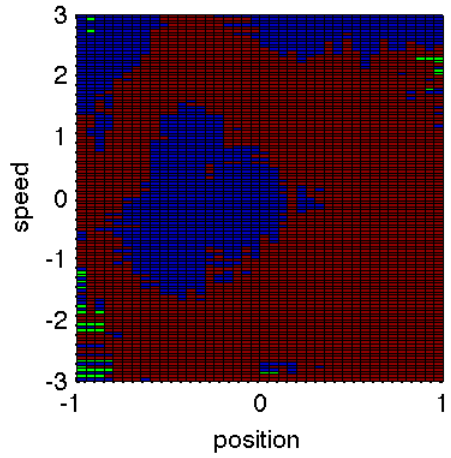
Figure 5.2: (a) - (e): Graphical representation of optimal policy, $\widehat{\pi}^*_N$, after $N$ steps of neural fitted $Q$-iteration, where $N = 10$, 20, 30, 40 and 50 respectively. (f): Trajectory from $s_0 = (-0.5, 0)$ under policy $\widehat{\pi}^*_{50}$.



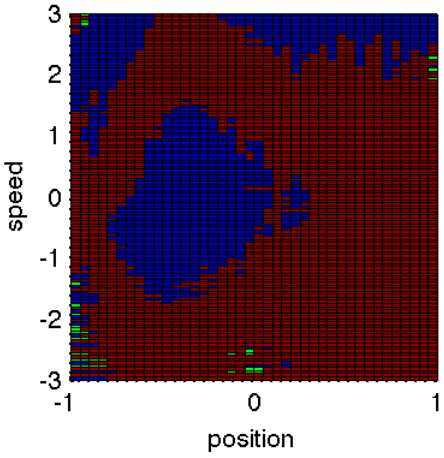

(a) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{10}(s,a)$



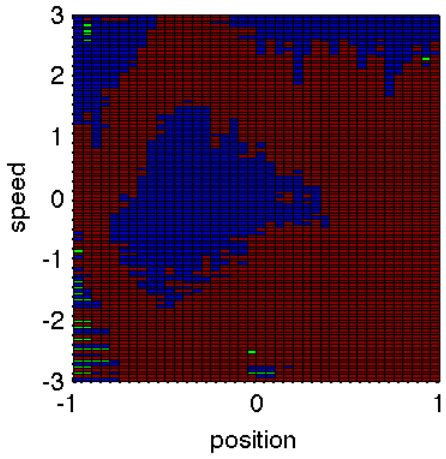

(b) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{20}(s,a)$



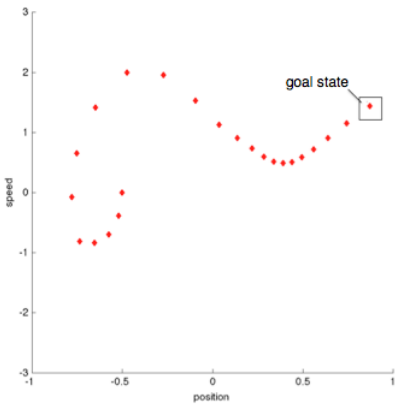

(c) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{30}(s,a)$




(d) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{40}(s,a)$

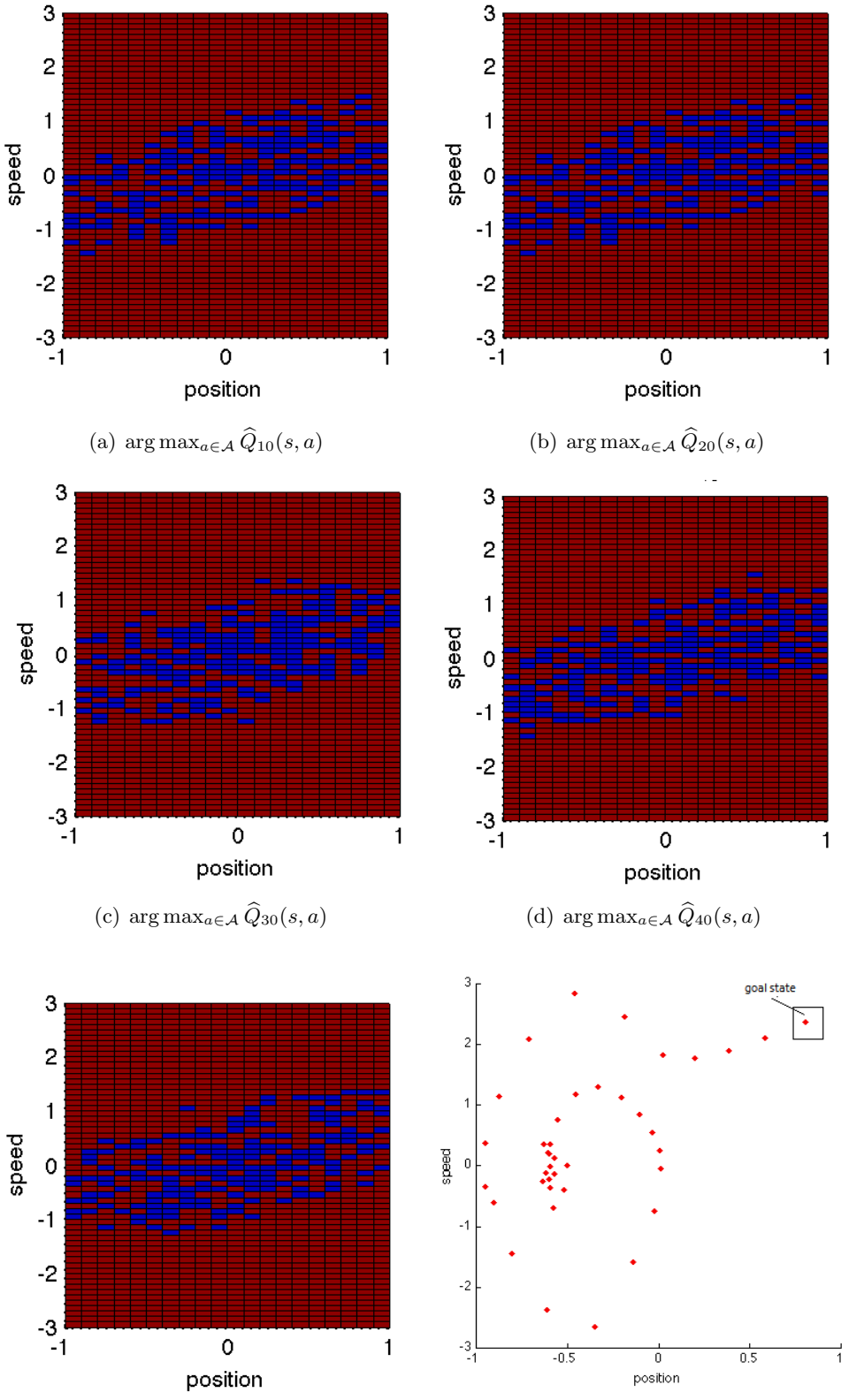



(e) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{50}(s,a)$




(f) Trajectory from initial state $s_0 = (-0.5, 0)$

Figure 5.3: (a) - (e): Graphical representation of optimal policy, $\widehat{\pi}^*_N$, after $N$ steps of fitted $Q$-iteration with extra trees, where $N = 10$, 20, 30, 40 and 50 respectively. (f): Trajectory from $s_0 = (-0.5, 0)$ under policy $\widehat{\pi}^*_{50}$.




(a) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{10}(s,a)$

(b) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{20}(s,a)$

(c) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{30}(s,a)$

(d) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{40}(s,a)$

(e) $\arg\max_{a\in\mathcal{A}} \widehat{Q}_{50}(s,a)$

(f) Trajectory from initial state $s_0 = (-0.5, 0)$

Figure 5.4: (a) - (e): Graphical representation of optimal policy, $\widehat{\pi}^*_N$, after $N$ steps of LSPI, where $N = 10$, 20, 30, 40 and 50 respectively. (f): Trajectory from $s_0 = (-0.5, 0)$ under policy $\widehat{\pi}^*_{50}$.

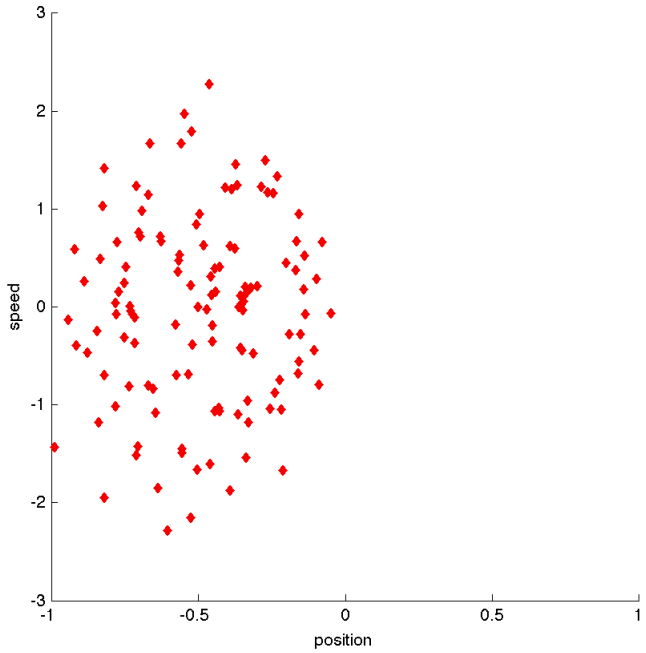



Figure 5.5: Example of the trajectory resulting from an unsuccessful policy on the mountain car domain.

### (iii) Number of steps to goal state of mountain car task for each learning technique

For the final part of this experiment, we record the number of successful policies produced by each learning technique after 50 iterations when applied to 30 sample sets of 10 000 samples each. From these 30 sample sets, we select 10 sample sets which allow for a successful optimal policy to be computed across all three learning techniques. We compute the average number of steps to reach the goal state from the initial state for each method.

Table 5.4 gives the number of times the learning technique considered produces a successful policy over 30 sample sets.

| | NFQ | FQI | LSPI |
|---|---|---|---|
| **Number of successful policies** | 21 | 28 | 23 |
| **% success** | 70.00 | 93.33 | 76.67 |

Table 5.4: Number of successful policies computed under each learning technique over 30 sample sets.

In Table 5.5 we record the number of steps required to reach the goal state for each batch learning method over 10 sample sets or trials and compute an average.

| Trial | Steps taken by NFQ | Steps taken by FQI | Steps taken by LSPI |
|---|---|---|---|
| 1 | 23 | 20 | 36 |
| 2 | 19 | 35 | 40 |
| 3 | 174 | 21 | 92 |
| 4 | 40 | 16 | 61 |
| 5 | 39 | 32 | 67 |
| 6 | 27 | 29 | 69 |
| 7 | 138 | 21 | 78 |
| 8 | 273 | 22 | 105 |
| 9 | 46 | 22 | 73 |
| 10 | 20 | 24 | 40 |
| **Average** | 79.90 | 24.20 | 66.10 |

Table 5.5: Number of steps taken by each algorithm to reach goal state under optimal policy over 10 trials.

### 5.3.3 Discussion of results for mountain car domain

Results from experimentation on the mountain car task demonstrate that each learning method has its own associated advantages and disadvantages. In terms of speed, LSPI and neural fitted $Q$-iteration are both significantly faster than the extra trees version of fitted $Q$-iteration. This is a direct consequence of the function approximation structure that is used for each technique. For the particular set of samples used in part (ii) of this experiment, the trajectories obtained as a result of the optimal policy demonstrate similar performance between both versions of fitted $Q$-iteration. The resulting trajectory produced by applying LSPI to the same set of samples, is almost double in length. This means that for the sample set used in part (ii), it takes LSPI significantly longer to reach the goal state than neural fitted $Q$-iteration and fitted $Q$-iteration with extra trees. From part (ii), we would expect similar results when applying each of the learning methods to different sample sets in part (iii). However, this is not necessarily the case. While fitted $Q$-iteration with extra trees and neural fitted $Q$-iteration are essentially variations of the same learning technique, the extra trees implementation is far superior in terms of the number of successful policies it is able to produce. In fact, despite the results from part (ii), LSPI is able to generate an optimal policy with a higher success rate than neural fitted $Q$-iteration, regardless of the length of the resulting trajectory. These results are reinforced by the fact that when we apply all three learning methods to sets of samples that generate successful policies in each case, neural fitted $Q$-iteration tends to produce trajectories that vary considerably in length. In some instances, the trajectories are much shorter than those produced by LSPI and FQI with extra trees whereas in other cases, the opposite is true. This means that neural fitted $Q$-iteration produces trajectories that are on average longer than those produced by both other techniques. Overall, the performance of neural fitted $Q$-iteration is somewhat unpredictable and

largely dependent on the sample set used. A possible explanation is that the neural fitted $Q$-iteration algorithm may require more iterations to be run to refine its approximation of the $Q$-function adequately enough. In our experimentation, we have run 50 iterations using each method however, the number of iterations required to converge to an optimal $Q$-function may be significantly different for each method. This is further demonstrated by (a) - (e) in Figure 5.2 where the policy differs significantly between iterations. In contrast, the optimal policies for LSPI and fitted $Q$-iteration with extra trees remain largely stable after 20 iterations (see Figures 5.3 and 5.4 (a) - (e)). It is also possible that a better neural network structure is required. In general, despite its speed, fitted $Q$-iteration using extra trees tends to produce the most effective optimal policy that results in a trajectory with the shortest number of steps.

### 5.3.4 Comparison of algorithmic outcomes for swing-up acrobot

This experiment is more or less identical to the experiment performed on the mountain car task that was discussed in the previous section. We have generated samples starting from the initial state $(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2}) = (0,\ 0,\ 0,\ 0)$ representing the acrobot lying in the vertical position using a random policy. In part (i), we compare the run time performances of neural fitted $Q$-iteration, fitted $Q$-iteration and LSPI on sample sets of sizes 5 000, 10 000, 15 000, 20 000, 25 000 and 30 000 respectively. At each time step in an episode within the set of samples, we choose a random action from the set of 3 possible actions $\{-1, 0, 1\}$. These actions correspond to the torque applied to the second joint of the acrobot. An episode reaches completion once the acrobot is able to swing its tip above the first joint by an amount equal to the length of its links which in this case is 1. That is, the acrobot reaches its goal state when its tip lies above a height of 1. As with the mountain car task, we execute 50 iterations of each learning technique and measure the time it takes for each method to reach completion. The test is repeated over five trials from which an average run time is calculated. In part (ii), we use a sample set of 10 000 samples to calculate the $50^{th}$ step optimal policy, $\widehat{\pi}^*_{50}$ under each learning technique, and compare the outcomes. Finally, in part (iii), we repeat part (ii) over ten different sample sets each containing 10 000 samples. Here, we obtain a policy $\widehat{\pi}^*_{50}$, for each method across each sample set. We apply these policies to the initial state and calculate the number of steps required to reach the goal state for each sample set. We use the parameters $M = 50$ and $n_{min} = 2$ for fitted $Q$-iteration with extra trees. The number of trees is once again adequate to obtain a suitable model of the data. A neural network consisting of 7 input nodes, one hidden layer of 100 hidden nodes and 1 output node are used for neural fitted $Q$-iteration. For LSPI, we use 1 050 randomly centered Gaussian radial basis functions (350 basis functions per action) with $\sigma^2 = 5$. We present the results for (i) - (iii) hereafter.

### (i) Run time comparison of learning techniques using acrobot sample sets of varying sizes

Tables 5.6 - 5.8 show the running times of the three batch RL techniques under consideration for sample sets of varying sizes. The average of these times are computed over 5 trials.

| $\|\mathcal{F}\|$ | 1 | 2 | 3 | 4 | 5 | Average time (mins) |
|---|---|---|---|---|---|---|
| 5 000 | 25.71 | 22.19 | 28.27 | 24.85 | 24.93 | 25.19 |
| 10 000 | 43.15 | 49.81 | 54.72 | 46.11 | 46.82 | 48.12 |
| 15 000 | 42.71 | 48.92 | 51.47 | 51.33 | 51.67 | 49.22 |
| 20 000 | 49.93 | 56.49 | 53.19 | 53.83 | 59.71 | 54.63 |
| 25 000 | 57.96 | 59.21 | 63.94 | 63.60 | 58.13 | 60.57 |
| 30 000 | 61.97 | 65.74 | 66.81 | 62.39 | 65.00 | 64.38 |

Table 5.6: Run times of neural fitted $Q$-iteration on acrobot sample sets of varying sizes over 5 trials.

| $\|\mathcal{F}\|$ | 1 | 2 | 3 | 4 | 5 | Average time (mins) |
|---|---|---|---|---|---|---|
| 5 000 | 67.68 | 72.19 | 73.47 | 64.44 | 71.26 | 69.81 |
| 10000 | 121.71 | 121.84 | 125.92 | 127.88 | 123.32 | 124.13 |
| 15 000 | 129.33 | 134.45 | 132.29 | 132.25 | 141.73 | 134.01 |
| 20 000 | 137.21 | 139.48 | 145.60 | 157.83 | 137.74 | 143.57 |
| 25 000 | 149.77 | 149.91 | 145.38 | 148.20 | 148.34 | 148.32 |
| 30000 | 146.79 | 141.24 | 148.17 | 156.69 | 169.92 | 152.56 |

Table 5.7: Run times of fitted $Q$-iteration using extra trees on acrobot sample sets of varying sizes over 5 trials.

| $\|\mathcal{F}\|$ | 1 | 2 | 3 | 4 | 5 | Average time (mins) |
|---|---|---|---|---|---|---|
| 5 000 | 78.87 | 73.29 | 73.81 | 76.95 | 68.47 | 74.28 |
| 10 000 | 105.38 | 112.31 | 108.64 | 108.77 | 108.90 | 108.80 |
| 15 000 | 126.98 | 124.01 | 124.73 | 121.35 | 112.51 | 121.92 |
| 20 000 | 137.67 | 139.42 | 139.28 | 141.74 | 140.06 | 139.63 |
| 25 000 | 148.98 | 149.20 | 149.27 | 149.43 | 151.72 | 149.71 |
| 30 000 | 152.29 | 152.34 | 150.88 | 157.46 | 155.53 | 153.70 |

Table 5.8: Run times of LSPI on acrobot sample sets of varying sizes over 5 trials.

The run times of each algorithm increase with an increase in the size of samples sets used for learning. As before, both versions of fitted $Q$-iteration spend the most time calculating the regression step to obtain a suitable $Q$-function approximation. The

main computational step for LSPI consists of computing the basis functions for each state $(\theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2})$. We observe that the extra trees implementation is very slow even across a different domain. The higher dimensionality of the state space in this case may contribute to the fact that run times are slower than when using the mountain car domain as there are more attributes to test and potentially split when using fitted $Q$-iteration with extra trees. Neural fitted $Q$-iteration outperforms both other methods in terms of speed. LSPI performs significantly slower when applied to the swing-up acrobot task as opposed to its performance on the mountain car domain. This is a direct consequence of using a larger number of basis functions to approximate the $Q$-values. It is interesting to observe that the average run times of LSPI on the smallest set of 5 000 samples and the largest set of 30 000 samples are higher than those of fitted $Q$-iteration with extra trees. Figure 5.6 shows a summary of these times for each algorithm as the size of the sample set used for learning increases.

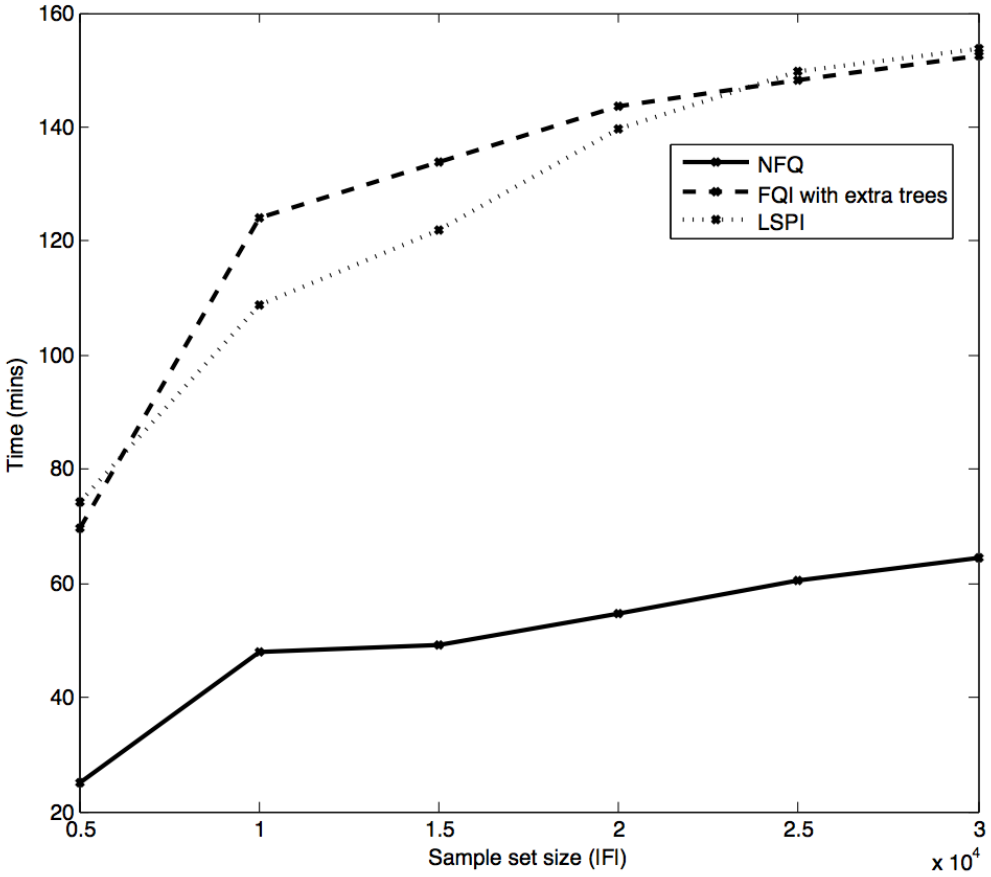



Figure 5.6: Average run times of each algorithm when applied to the swing-up acrobot task using sample sets of varying sizes.

**Optimal policies determined by batch RL techniques for the swing-up acrobot task**

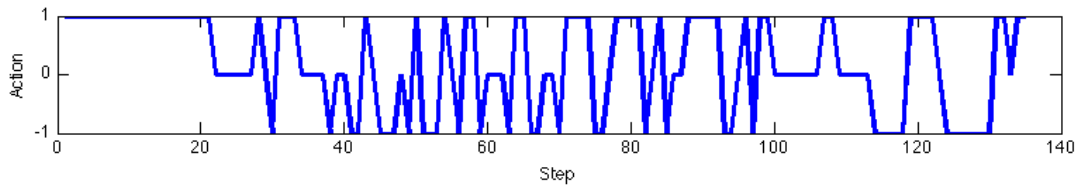



Figure 5.7: Policy $\widehat{\pi}^*_{50}$ obtained from applying neural fitted $Q$-iteration to the swing-up acrobot task.

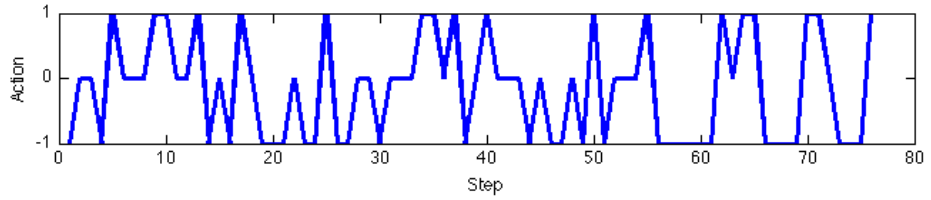



Figure 5.8: Policy $\widehat{\pi}^*_{50}$ obtained from applying fitted $Q$-iteration with extra trees to the swing-up acrobot task.

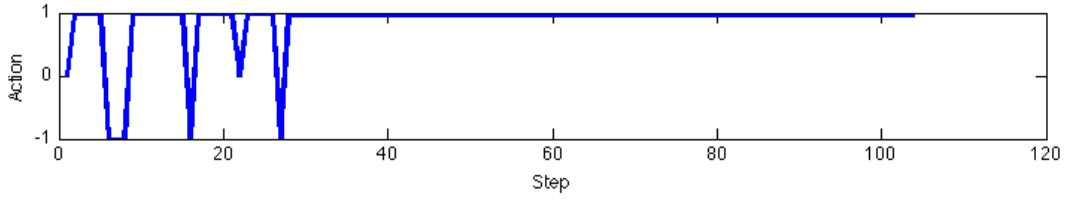



Figure 5.9: Policy $\widehat{\pi}^*_{50}$ obtained from applying LSPI to swing-up acrobot task.

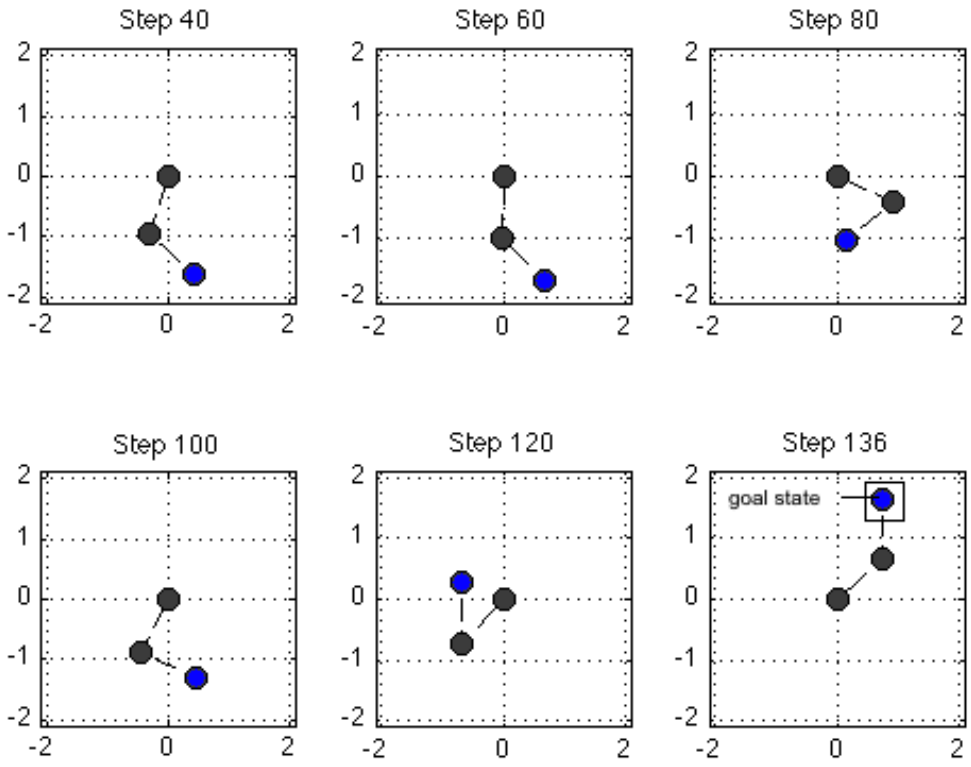



Figure 5.10: Positioning of acrobot at various steps under $\widehat{\pi}^*_{50}$ using neural fitted $Q$-iteration.

In this part of the experiment, we determine the $50^{th}$ step optimal policy from the start state for the swing-up acrobot task. To do so, we run 50 iterations of each learning technique on a sample set of 10 000 randomly generated acrobot samples. Because of the dimensionality of the state space, it is impossible to graphically represent $\arg\max_{a\in\mathcal{A}} \widehat{Q}(s,a)$ for each pair $(s,a)$ as we did for the mountain car task. Instead, we provide figures for the positioning of the acrobot at different steps having followed the policy $\widehat{\pi}^*_{50}$ from the initial state. In particular, Figures 5.10, 5.11 and 5.12 demonstrate the acrobot at different positions in an episode following learning under neural fitted $Q$-iteration, fitted $Q$-iteration with extra trees and LSPI respectively. The goal states are indicated. The corresponding sequence of actions taken are given in Figures 5.7, 5.8 and 5.9 respectively. We observe that for this particular sample set, it takes the acrobot 136 steps to reach the goal state using the policy $\widehat{\pi}^*_{50}$ obtained from neural fitted $Q$-iteration. For the same sample set, it takes the acrobot 77 steps and 105 steps to reach the goal state using $\widehat{\pi}^*_{50}$ from fitted $Q$-iteration with extra trees and LSPI respectively.

If we compare the policies $\widehat{\pi}^*_{50}$ obtained from each learning technique by looking at the sequence of actions the acrobot takes to reach the goal state, we can see that all

three policies are very different. The sequence of actions taken by the acrobot having applied $\widehat{\pi}^*_{50}$ from both versions of the fitted $Q$-iteration algorithm involve many more oscillations between applying a torque and applying a torque in the opposite direction. As a result, we see the acrobot swinging back and forth more rapidly. This is not the case when using the LSPI learning technique; here, a positive torque is continuously applied after the $28^{th}$ step. There are also more instances where zero torque is applied in both versions of fitted $Q$-iteration than in the policy obtained from using LSPI.

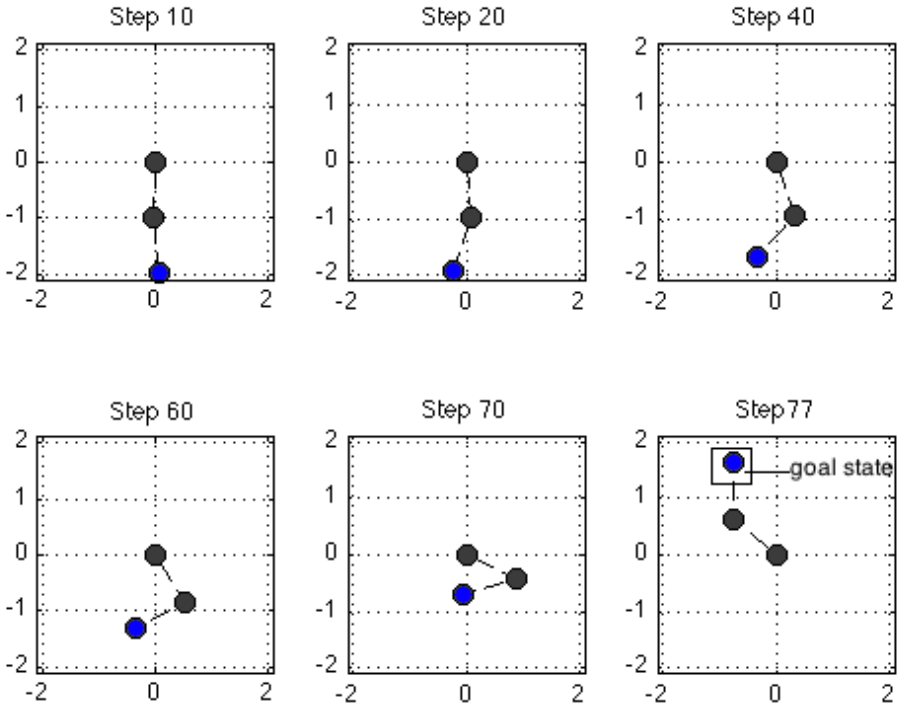



Figure 5.11: Positioning of acrobot at various steps under $\widehat{\pi}^*_{50}$ using fitted $Q$-iteration with extra trees.

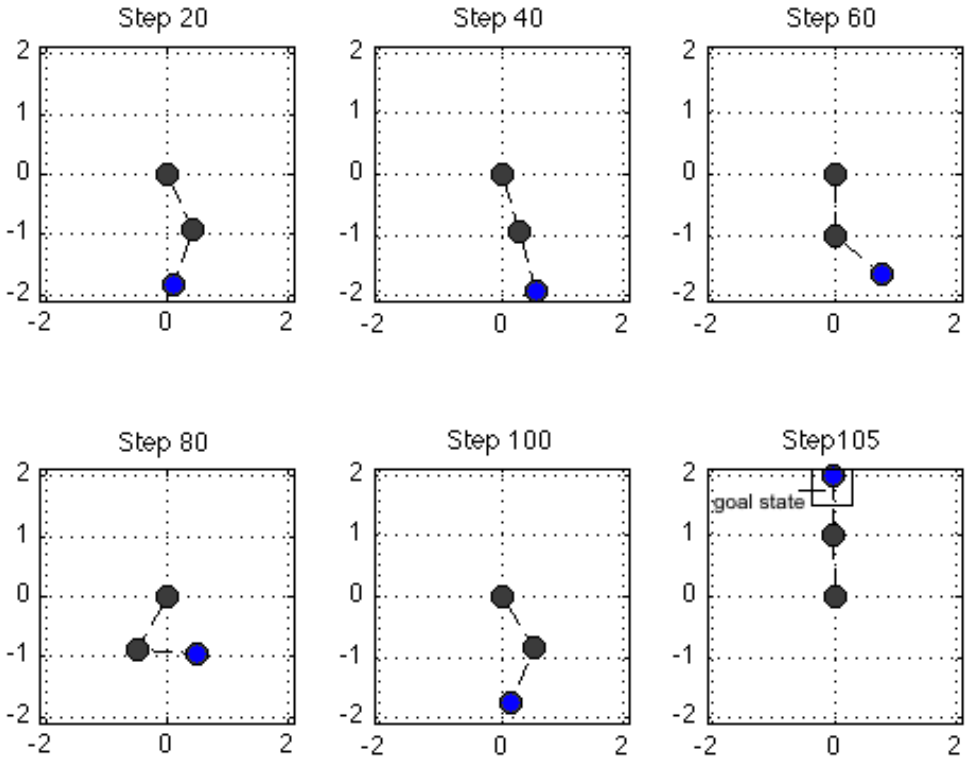



Figure 5.12: Positioning of acrobot at various steps under $\widehat{\pi}^*_{50}$ using LSPI.

**Number of steps to goal state of acrobot task for each learning technique**

As in the final part of our experimentation on the mountain car domain, we compare the average number of steps taken by each algorithm to reach the goal state from $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0,\ 0,\ 0,\ 0)$ under their respective optimal policies. Note that since the acrobot task only has one terminal state i.e. when the goal state is reached, we omit calculating the percentage of successful policies generated which we did for the mountain car task. For this experiment, we conduct 10 trials using 10 sample sets of 10 000 samples each. The samples are generated according to the random policy as before. We determine the optimal policy after 50 iterations from the initial state for each of these sets for each learning technique considered. Again, we compute the number of steps each algorithm takes to reach the goal state from the initial state under the optimal policies obtained. Table 5.9 shows these results.

| Trial | Steps taken by NFQ | Steps taken by FQI | Steps taken by LSPI |
|---|---|---|---|
| 1 | 48 | 39 | 148 |
| 2 | 36 | 35 | 180 |
| 3 | 42 | 42 | 105 |
| 4 | 54 | 27 | 135 |
| 5 | 51 | 29 | 94 |
| 6 | 136 | 80 | 105 |
| 7 | 37 | 31 | 83 |
| 8 | 58 | 32 | 129 |
| 9 | 21 | 48 | 101 |
| 10 | 62 | 37 | 98 |
| **Average** | 54.50 | 40.00 | 117.80 |

Table 5.9: Number of steps taken by each algorithm to reach goal state of acrobot task under optimal policy over 10 trials.

### 5.3.5 Discussion of results for the swing-up acrobot domain

Results from experimentation on the swing-up acrobot control task reveal some differences in the performances of each learning technique in comparison to the results from the mountain car task. In terms of speed, we observe once again that the extra trees version of fitted $Q$-iteration is very slow. However, the speed advantage demonstrated by LSPI when applied to the mountain car task, is not observed here. On the contrary, there are instances where LSPI takes longer to run than fitted $Q$-iteration with extra trees. This is because of the fact that a substantial number of basis functions are calculated. Ultimately, it is the method used to calculate these basis functions that largely dictates the speed of the LSPI algorithm. Neural fitted $Q$-iteration is significantly faster than both other learning techniques. In terms of the optimal policies obtained when applying each technique to sample sets of 10 000 samples, both versions of fitted $Q$-iteration have similar performance with a similar number of average steps required to reach the goal state. LSPI takes on average double the number of steps to reach the same goal. This is suggestive of the fact that more basis functions should possibly be used to improve the quality of the $Q$-function approximation. Alternatively, it could be worthwhile investigating the use of different basis functions, other than randomly centered Gaussian radial basis functions, for $Q$-function approximation on this task. Overall, the policy determined by the extra trees version requires the fewest steps to reach the goal state. In doing so, the acrobot swing-up task is accomplished sooner than when using neural fitted $Q$-iteration or LSPI.

## 5.4 Experimentation on simulated data

In this experiment, we compare the outcomes of each batch RL algorithm when applied to HIV data simulated using the model discussed in Section 4.5.2 of the previous chapter. In particular, we make observations about whether or not a patient's health status improves after taking the actions recommended by each of the learning techniques following training.

To perform this experiment, we use an iterative data generation and testing procedure analogous to Ernst et al. (2006). We assume that patients are monitored every five days for a period of 1 000 days. At the first iteration, we generate the data for fifty patients all in the unhealthy steady state. Thereafter, every five days the patient's health status is assessed according to the quantities of the state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ and we select a random treatment action from the set of actions {RTI on and PI on, RTI on and PI off, RTI off and PI on}. By recording a patient's state of health every five days, we obtain a trajectory $(s_0, a_0, s_1, a_1, \ldots, s_{200})$ corresponding to the course of action taken on the patient over 1 000 days. This trajectory produces 200 samples of the form $(s_t, a_t, s_{t+1})$. Hence, by the end of the first iteration, a total of 10 000 samples are available. At this point we run each of the batch RL algorithms on the set of 10 000 samples for 50 iterations to produce a policy $\widehat{\pi}_{50}^*(s_0)$ where $s_0$ is the unhealthy steady start state.

During the second iterative step, we generate the data for 10 new patients again from unhealthy stationary equilibrium and record their state data every five days for exactly 1 000 days. Instead of randomly selecting the medication taken by a patient every five days, we consider the optimal action suggested by the policy $\widehat{\pi}_{50}^*$ that was obtained from the original set of 10 000 samples. We take this optimal action 70% of the time and choose a random action for the remaining 30%. Once again we apply each of the batch algorithms to the new set of 12 000 samples for 50 iterations and obtain a new optimal policy for the larger sample set.

At the third iterative step, we generate the data for another set of 10 new patients from the unhealthy steady state and record the necessary state data. We consider the action suggested by the optimal policy obtained from the set of 12 000 samples and take this action 85% of the time; for the remaining 15%, we choose a random action. We repeat the iterative procedure ten times to produce a total of 30 000 samples. After the tenth iteration, we use the optimal policy $\widehat{\pi}_{50}^*$ that was obtained from the set of 30 000 samples to generate the data for one patient (once again starting at unhealthy stationary equilibrium). This time, we take only the action suggested by $\widehat{\pi}_{50}^*$. The resulting trajectory obtained represents the evolution of a patient's health condition under the policy $\widehat{\pi}_{50}^*$ from a 30 000 sample set. We determine such a trajectory for each of the learning methods under consideration.

By alternating between sample generation and testing using the batch RL algorithms, we are able to determine whether the policies learned by the RL techniques eventually improve a patient's state of health overall. For the fitted $Q$-iteration algorithm using extra trees, we build a set of 50 trees at each iteration like Ernst et al.

(2006); for the neural fitted $Q$-iteration algorithm, we use a neural network with 9 input nodes corresponding to the state and action variables, 100 hidden nodes and 1 output node. When applying LSPI, we use 9 000 randomly centered Gaussian radial basis functions with $\sigma^2 = 4$.

Figure 5.16 represents the changes in the state variables for a patient being treated from unhealthy stationary equilibrium for a period of 1 000 days following $\widehat{\pi}^*_{50}$ from using neural fitted $Q$-iteration. We note that over the course of treatment an improvement in the CD4$^+$ count and a decrease in the overall viral load are observed. We also observe decrease in both infected macrophages and infected T-cells with a reduced viral load. Figure 5.13 shows the corresponding course of action taken for the patient when neural fitted $Q$-iteration was applied. We observe that initially extensive drug cycling occurs between using RTIs and PIs. This is followed by a period of using only PIs. After 730 days of treatment, the patient begins using RTIs again but this time without the use of any PIs.

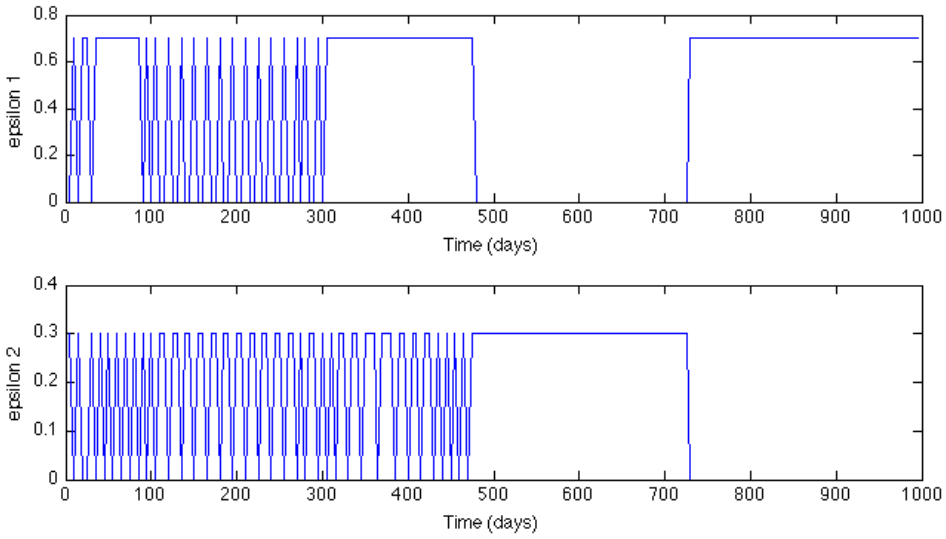



Figure 5.13: Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using neural fitted $Q$-iteration.

Figures 5.17 and 5.14 show these results for a patient being treated from unhealthy stationary equilibrium for a period of 1 000 days under $\widehat{\pi}^*_{50}$ from fitted $Q$-iteration with extra trees. Once again, we observe an improvement in overall health in terms of the CD4$^+$ count, viral load, macrophage counts and cytotoxic T-cell counts; however, the course of action in this case involves many more periods of drug cycling than in the previous case. Similarly, Figures 5.18 and 5.15 show the results for a patient being treated from unhealthy stationary equilibrium for 1 000 days under $\widehat{\pi}^*_{50}$ from applying LSPI to a set of 30 000 samples. The courses of action obtained using LSPI and fitted

$Q$-iteration with extra trees are quite similar in the sense that more drug cycling occurs with fewer periods of drug stability in comparison to the results from using neural fitted $Q$-iteration. Nonetheless, all three methods allow for an improvement in a patient's health condition.

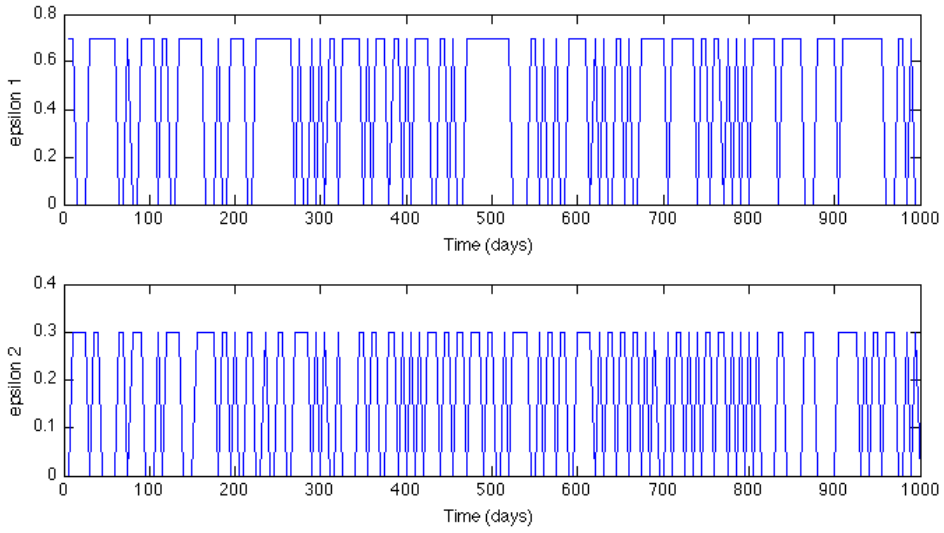



Figure 5.14: Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using fitted $Q$-iteration with extra trees.

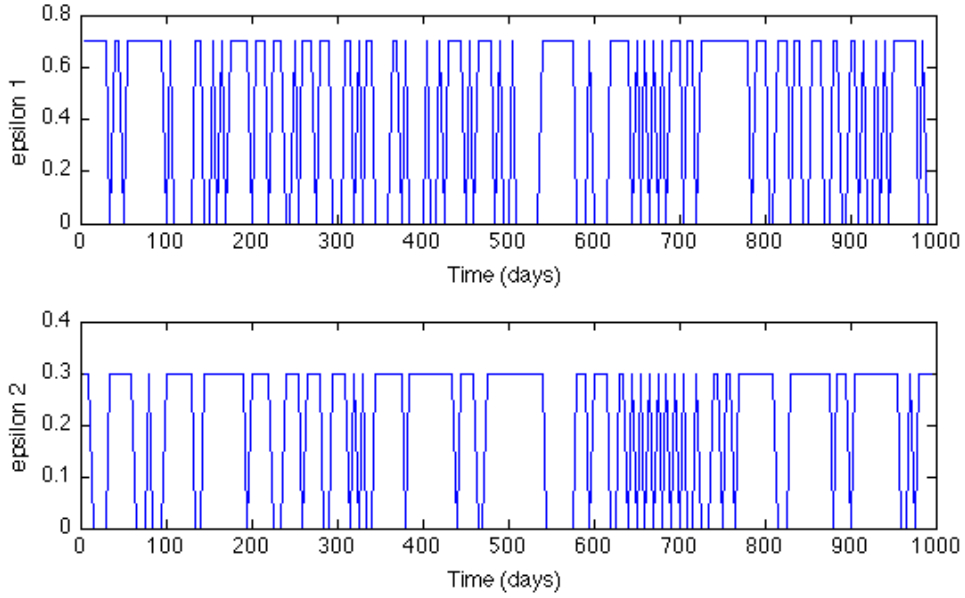



Figure 5.15: Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using LSPI.

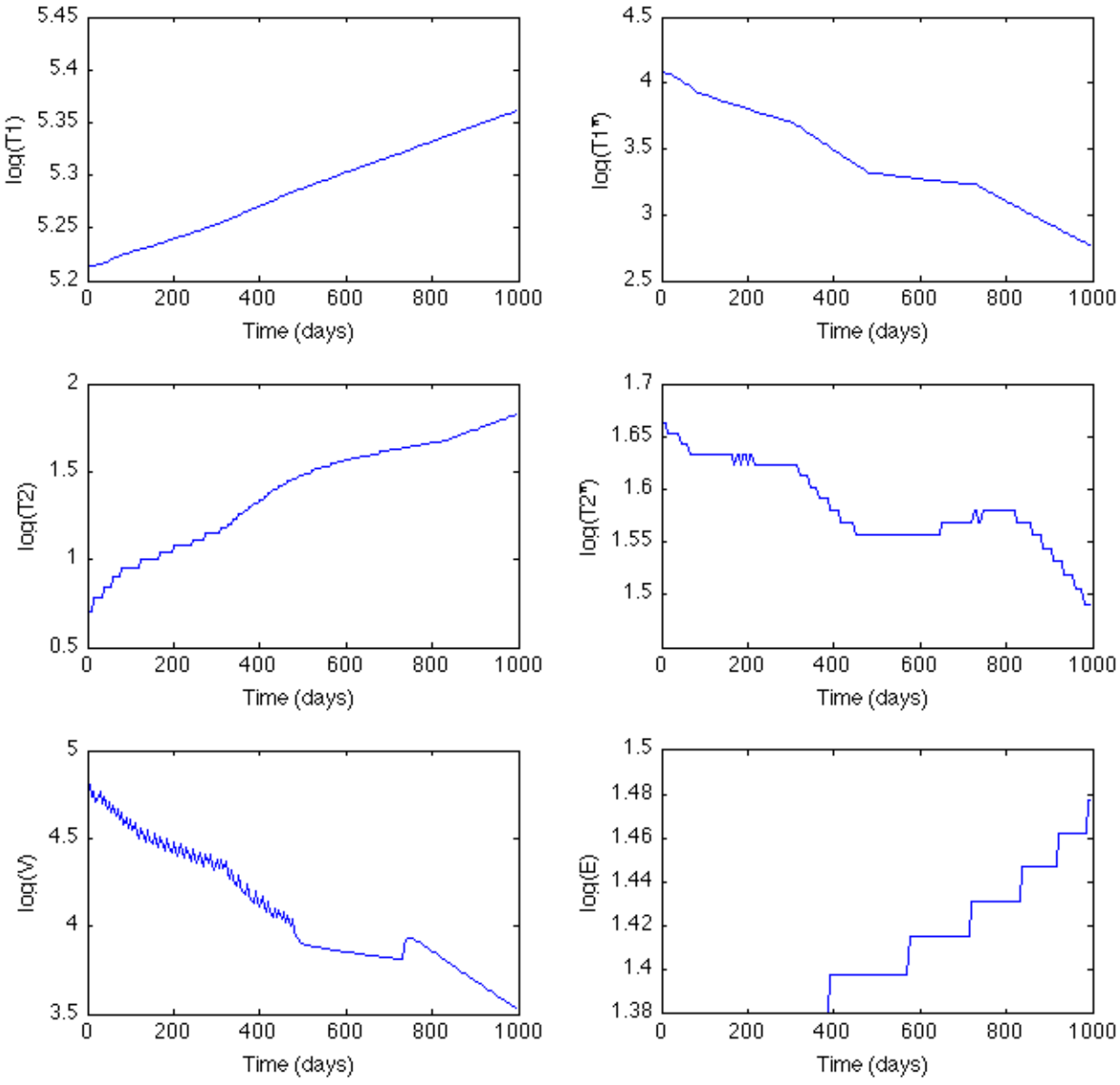



Figure 5.16: Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying neural fitted $Q$-iteration.

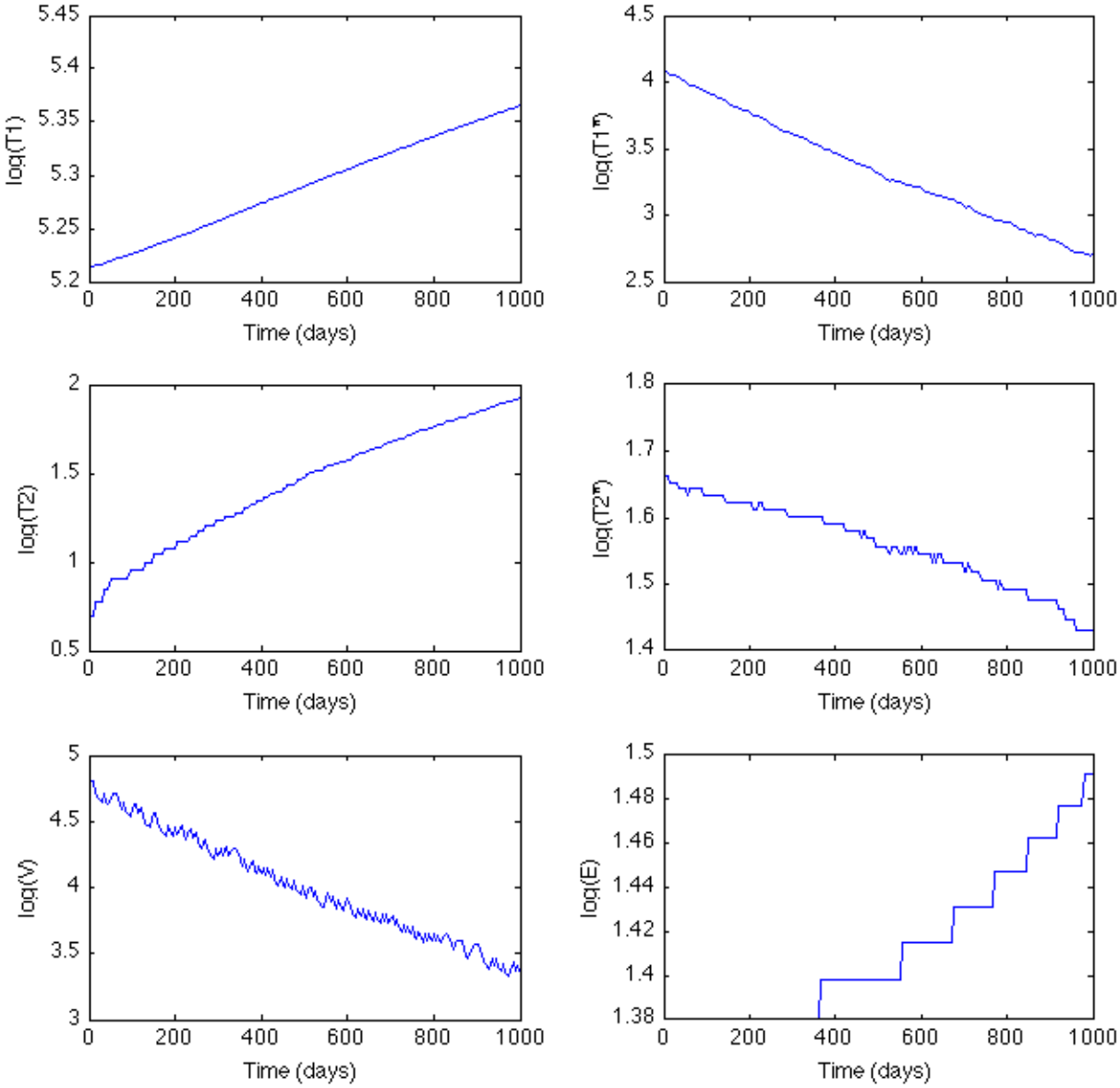



Figure 5.17: Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying fitted $Q$-iteration with extra trees.

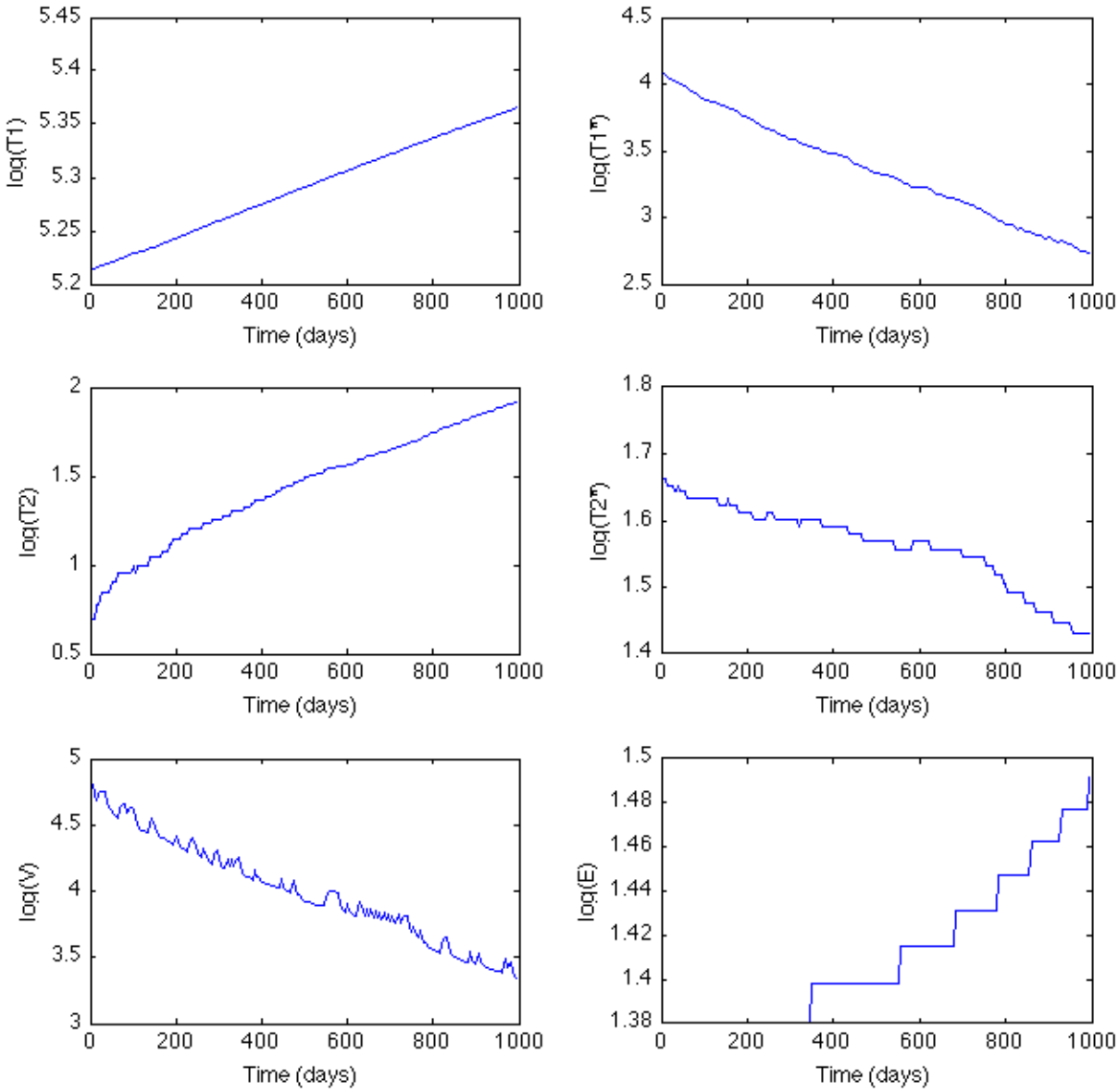



Figure 5.18: Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying LSPI.

### 5.4.1 Discussion of results from testing batch RL on simulated HIV patient data

In general, all three methods of learning produce policies after 50 iterations that demonstrate an improvement in the immune outcomes for an infected individual. The results from using neural fitted $Q$-iteration show a more gradual change in the state variables than the results from the other two methods. The policy obtained using neural fitted $Q$-iteration has larger periods of stability where no drug cycling occurs. This seems realistic since it allows the patient periods where they can acclimatize to using a particular drug (assuming only one drug from the particular drug class is used). However, despite the improvement in a patient's condition after 1 000 days, in comparison to the results from using LSPI and fitted $Q$-iteration with extra trees, the overall condition of a patient remains slightly weak. This is evident from the fact that after 1 000 days, a patient following $\widehat{\pi}^*_{50}$ from neural fitted $Q$-iteration has a lower $CD8^+$ T-cell count, a higher infected $CD4^+$ count and a higher infected macrophage count than a patient following $\widehat{\pi}^*_{50}$ from LSPI and fitted $Q$-iteration with extra trees after the same period of time. There is also a period within the 1 000 days where a patient under $\widehat{\pi}^*_{50}$ from neural fitted $Q$-iteration experiences a blip in their viral load. This occurs after a consistent decrease in their viral load between 400 and 700 days (see Figure 5.16). While this is unlike the results obtained from applying both other learning techniques, it is not unlike situations in real life. Often, a patient's viral load may fluctuate despite an improvement in their overall health status. In these situations, it is more useful to look at other state variables as an indication of the patient's condition. Overall, results from using neural fitted $Q$-iteration suggest that using the learning technique to schedule RTIs and PIs can result in improved patient outcomes in the simulated case. However, within a period of 1 000 days, a patient is still unable to reach a healthy steady state.[3]

Similar observations can be made from the results obtained from using fitted $Q$-iteration with extra trees and LSPI. In general, fitted $Q$-iteration with extra trees is able to produce a policy $\widehat{\pi}^*_{50}$ that results in the largest improvement in a simulated patient's health status overall after a period of 1 000 days. This is evident from the steadily decreasing infected cell counts and decreasing viral load in a patient simulated from unhealthy stationary equilibrium. In both these cases however, drug cycling occurs very frequently (see Figures 5.14 and 5.15). If this drug cycling involves switching between classes of drugs as well as different drugs within a particular class, it is not entirely realistic nor advisable since frequent switching can have many side-effects in the real world. If the drug cycling involves switching between drug classes but does not necessarily involve changing the drugs used from a particular class, this may not be the case. In these instances, switching between drug classes would involve taking a previously used drug but from a different class. By using the model provided in Chapter 4 to simulate patient data, it is impossible to distinguish between these cases. Other issues that arise from using this model to simulate patient data include the fact that every

[3]Subsequent experimentation has shown that by extending this period to 2 000 days, a patient is able to reach a state that is slightly closer to the healthy steady state.

patient is simulated from the same initial state, which is not realistic. It is debatable that the efficacy parameters for the classes of RTIs and PIs are fixed to the values of either 0 or 0.7, and 0 or 0.3 respectively, throughout a period of treatment for each individual. These values are specified before before performing a simulation however, in reality we would like to determine these values to be able to ascertain which drugs are more suitable for treatment at a particular time than others. Whether the efficacy of a particular drug should be fixed to a particular range of values is questionable; it may be the case that a drug's efficacy in fact changes over time in accordance to a patient's infection, viral strain etc.

Ultimately, if we are to address the efficacy of frequent switching and some of the other points raised, a more detailed mathematical model that considers variations within each drug class, would be required. Alternatively, modelling the MDP for reinforcement learning in a slightly different way and applying it to real data could allow us to gain some insight about switching between individual drug types as opposed to drug classes. All in all, the results from this experiment suggest that the model used for data simulation is too general and should incorporate more drug classes or individual drugs to produce more telling results. For these reasons, we perform experimentation using batch reinforcement learning techniques on real data in the next section.

### 5.4.2 A note about the size of the sample set used for simulated data testing

In this experiment we have used a large sample set consisting of the data from several patients. This is motivated by the fact that using a larger sample set of more patients allows for more opportunities for variation to exist between samples. This variation is especially interesting for the purposes of learning and is reasonably important if we consider the fact that the model for data simulation uses only three actions. However, in reality it is possible that a significantly smaller sample of the population is available as a result of restricted data sources. For these purposes, it may also be worth investigating how the learning techniques perform on a significantly smaller sample set to be able to make a fair comparison between performances on both real and simulated data sets. The results of this experiment may be found in Appendix B.

## 5.5 Experimentation on real patient data

In this section, we extend our experimentation using batch reinforcement learning techniques further to a real HIV data setting. Unlike the sets of generated HIV data used in the previous section, real HIV data does not conform to a specific model and behaves significantly more unpredictably.

As described in Section 4.6.2 of Chapter 4, using real data introduces a number of issues that need to be carefully considered. In addition, the application of batch RL techniques to real data requires a different MDP formulation. For this experiment, we make use of our complete set of HIV data consisting of data from 250 random HIV-

infected individuals corresponding to 2 560 samples. The data has been preprocessed and prepared according to the discussion in Section 4.6.2. This is smaller than the data sets used in all previous experimentation based on our limited access to real data sources.

The outcome of this experiment corresponds to one or more action(s) in terms of drug therapy that the patient can take at a certain time step. Since we have no way of implementing these actions in real life, it is impossible to predict what will happen to the patient as a result of the action and use this information to determine subsequent actions. Hence for this experiment, we perform a ten-fold cross validation test in which the actions suggested by the learning techniques for a certain sample are validated against those actions or drugs used by the individual at that particular time. While this does not necessarily guarantee the best course of treatment, it allow us to determine the number of times the outcomes of the algorithm are consistent with the treatments followed by doctors.

In particular, we divide our data into ten validation and training sets respectively. Validation sets are constructed by selecting 256 samples from the original sample set and using the remaining samples as corresponding training data for an individual test. That is, we partition the sample set into ten sets of size 256 samples each. For a single trial, we use one of the sets as validation data while training on the remaining 2 304 samples comprising the other sets. The process is repeated over ten trials, each time using a different set of 256 samples to validate against. Training sets are used to train the batch algorithms while validation sets are used for purposes of measuring the number of times a match occurs between the outcomes of a particular learning technique following training, and the current course of treatment for a patient. The results are averaged over the ten sets to produce the average number of matches that occur under each learning method. Using ten-fold cross validation is advantageous since it allows for the possibility of selecting each individual sample from the original set once for testing. We note it is possible that the learning techniques suggest more than one suitable action. Here, the approximate $Q$-values for these actions is the same. In these cases, if a match occurs between one of the suggested actions and the originally prescribed action, we regard this as a successful match. While using matching accuracy as a performance measure assumes a symmetric loss in getting a prescription wrong which is most certainly not the case in reality, the results from using these learning algorithms are intended to serve as clinical-decision support, rather than a means of automating decision-making in clinical practice.

We have performed this experiment using $M = 50$ and $n_{min} = 2$ for fitted $Q$-iteration with extra trees, however preliminary testing revealed similar results when using larger values of $M$ for this domain. Using neural fitted $Q$-iteration required us to construct a much larger neural network for training and testing purposes. Specifically, we use 9 input nodes (corresponding to the two state variables and seven dimensional binary actions), 350 hidden nodes in the first hidden layer, 300 hidden nodes in the second hidden layer and 1 output node corresponding to the $Q$-value of a particular state-action pair $(s, a)$. For LSPI, we use 2 560 randomly centered Gaussian radial basis

functions with $\sigma^2 = 2$. We have run 50 iterations of each learning technique for training purposes. We present our results for each validation set in Table 5.10.

| Trial | Matches using NFQ | Matches using FQI | Matches using LSPI |
|---|---|---|---|
| 1 | 190 | 211 | 203 |
| 2 | 181 | 209 | 174 |
| 3 | 213 | 199 | 192 |
| 4 | 220 | 174 | 227 |
| 5 | 96 | 158 | 145 |
| 6 | 179 | 191 | 198 |
| 7 | 102 | 111 | 82 |
| 8 | 203 | 207 | 141 |
| 9 | 194 | 194 | 215 |
| 10 | 178 | 143 | 86 |
| **Average** | 175.60 | 179.7 | 166.30 |
| **% consistency** | 68.60 | 70.20 | 64.96 |

Table 5.10: Average consistency between learned actions and actions taken by clinicians using FQI with extra trees, NFQ and LSPI respectively.

### 5.5.1 Discussion of results in a real HIV setting

Results from this experiment have demonstrated that batch reinforcement learning techniques can be applied to real HIV data with reasonable success. We observe that despite the use of a relatively straightforward reward function and a state space consisting of only two state variables, all three learning techniques are able to produce outcomes that are consistent with the actions prescribed by clinicians a large number of times. This is quite interesting since medical professionals usually perform a number of different tests to assess a patient's health before prescribing a particular treatment. The information gained from these assessments is vastly more specific than what we use to train the learning methods for this experiment. On average, fitted $Q$-iteration with extremely randomized trees produces an outcome that matches the action taken by a clinician most frequently. The results from performing the same experiment using neural fitted $Q$-iteration and LSPI are fairly similar but less successful.

It is evident that choosing drugs for a patient involves considering their health status. This is dictated by time. That is, it may be reasonable to take certain combinations of drugs at one particular point in time but not at another. For these reasons, it is not necessarily the case that drugs that are used frequently among most patients are the best choice for every infected individual. This means that we cannot use the frequency at which a particular drug combination is suggested by the learning methods as an overall indicator of the quality of learning. We can however, compare the combinations of drugs recommended to those commonly used in treating HIV to see how realistic the learning is in terms of the suggested actions.

In this particular experiment, the most frequently occurring combination of drugs suggested by FQI with extra trees is EFV, 3TC, tdf. This is perfectly reasonable given the data used for training the algorithm and the fact that this combination is one that is regularly prescribed by medical professionals to HIV-infected individuals. Consider the validation set used in trial 5 in Table 5.10. This set of data produces a poor number of matches when used with neural fitted $Q$-iteration. Figure 5.19 shows the frequency at which combinations of actions are recommended. Figure 5.20 shows the actual frequency of drug combinations used in the same validation set. It is evident that the results are vastly different which ultimately means that a poor number of matches occur. It is obvious that neural fitted $Q$-iteration does not identify all the drug combinations prescribed for the data set. In particular, the NFQ algorithm fails to suggest the most frequently used drug combination for this validation set - AZT, tdf, LPV/r. One possible explanation for this is that having extracted the data used in this validation set from the original sample set, there are not enough samples using this action in the training set. There is only one instance where the action prescribed almost certainly produces a match using NFQ. This occurs when using the 3TC, LPV/r combination. NFQ suggests using this action 23 times in the validation set; in reality, this action was prescribed 20 times. It is also possible that there may be more than one suitable action available to a clinician at a particular time which could account for the number of mismatches that occur here. Based on the smaller number of matches that occur when using NFQ and LSPI on the validation set in trial 5, one would presume that the combinations of drugs suggested by these methods are not standard. However, this is certainly not the case. Instead, the actions that are suggested are not unusual in terms of the combinations of drugs used in reality. Most combinations of drugs are not recommended at all (demonstrated by the gaps between actions that are selected in Figure 5.19). We observe that the most frequently occurring drug combination suggested by neural fitted $Q$-iteration is EFV, 3TC, d4T. This is a standard drug combination that is prescribed 21 times in the validation set used for trial 5. Similar results are observed in the other validation sets where matching is poor too. Despite observing a poor number of matches in these validation sets, the actions recommended by the learning techniques are realistic in comparison with the drug combinations typically used by doctors every day. This is extremely promising if we are to consider using RL techniques in treatment design in the future. From the perspective of treatment simplification, these combinations of drugs may be suitable candidates for re-formulation to reduce patient pill burden and improve adherence to medication.

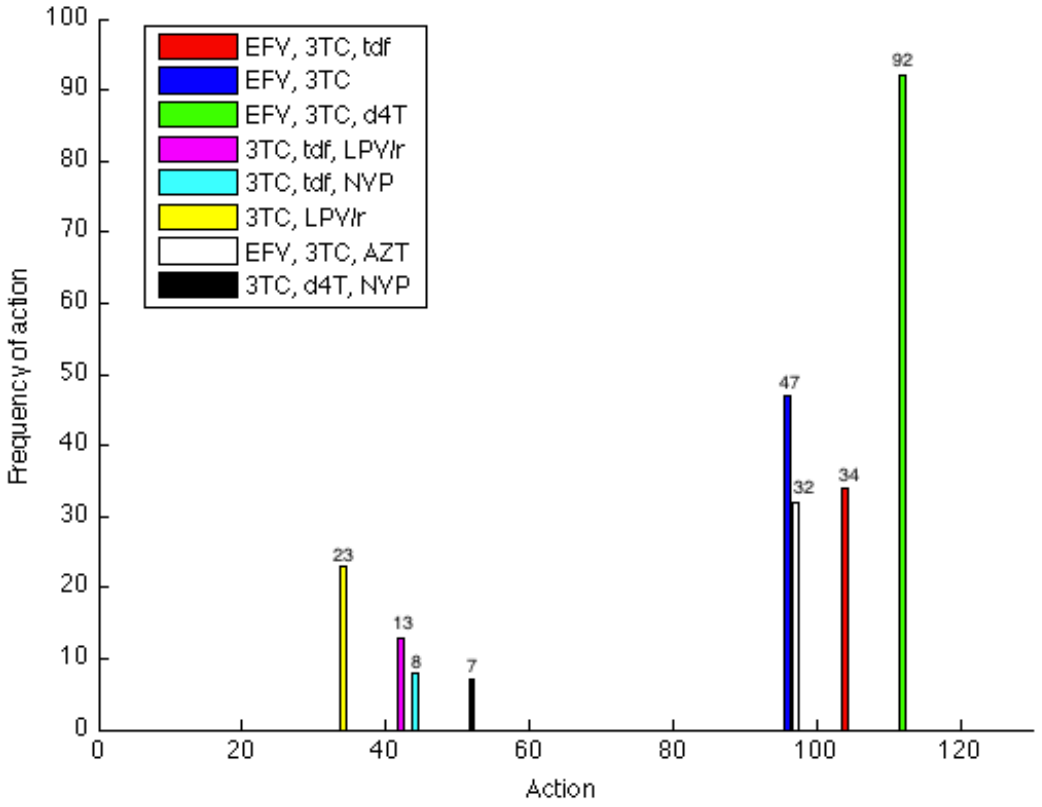



Figure 5.19: Frequency of recommended drug combinations when applying neural fitted $Q$-iteration to the validation set used for trial 5.

We have used a fairly sparse set of HIV samples to train and test each of the learning techniques in this thesis. For this reason, it may be more practical to perform leave-one-out cross validation instead of ten-fold cross validation in order to train the learning methods using as many samples as possible. It may also be interesting to train using all 2 560 samples and observe the outcomes produced when testing using samples comprising the actual training set. We have done this and tested the outcomes on the samples that comprised the validation set used in trial 1 previously. In this case, FQI using extra trees produces a match 219 times; NFQ produces a match still only 190 times, and LSPI produces a match 211 times. While one would expect the number of matches to be improved by training using more samples and testing on samples comprising the training set, the improvement is only a slight one. In fact, the number of matches produced by NFQ is identical to when the validation data was separated from the original set of samples in trial 1. This is very interesting since it suggests the performances of the learning methods from training using a complete sample set are very similar to the performances from training using a partial sample set. Perhaps this is indicative of the fact that to observe an improvement in the number of matches, a significantly larger set

of samples is required. Alternatively, it suggests that better neural network structures or basis functions should be considered in NFQ and LSPI to improve training. It is also possible that the parameters used in FQI with extra trees could be better tuned to produce an improved number of matches.

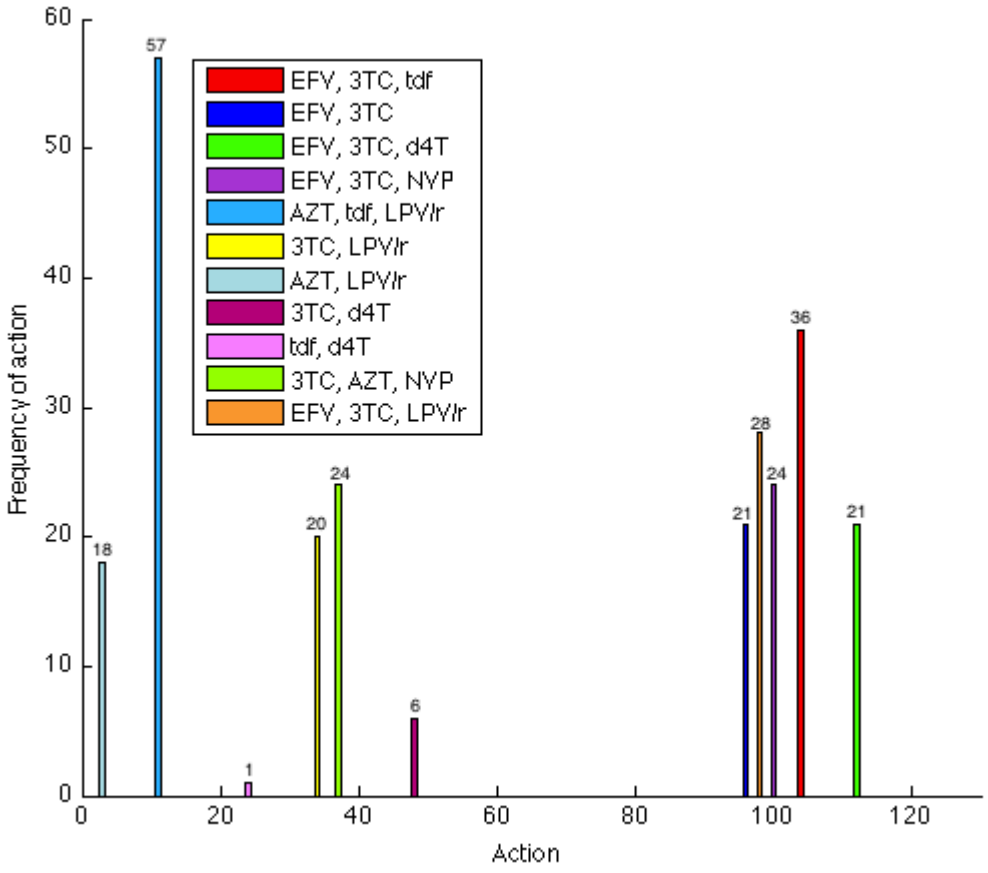



Figure 5.20: Frequency of prescribed drug combinations in the validation set used for trial 5.

### 5.5.2 A note about supervised learning

A natural question that arises after investigating the use of batch RL techniques to identify suitable treatments for HIV-infected individuals at specific times, is whether or not using supervised learning techniques alone would be able to produce similar results. To investigate this, we consider comparing the outcomes produced by a standard neural network with those produced by the reinforcement learning techniques under consideration. Here, our neural network is structured to take as input a patient's wellness indicators in the form of a log-normalized $CD4^+$ count and viral load. The output of

the neural network corresponds to the action identified as a suitable treatment for a patient for a particular sample. Hence our neural network structure consists of 2 input nodes, one hidden layer with 300 nodes and an output layer of 7 nodes. These 7 nodes correspond to whether a particular drug is selected for treatment or not. In this experiment, we use the data that was used for validation in trial 1 of the previous experiment as test data, and observe the number of times the output of the network corresponds to a match in treatment. That is, we consider the 256 samples used in trial 1 of our real data experimentation as test data for the neural network. The remaining 2 304 samples are used for training the neural network. We train the neural network using the resilient backpropagation technique that was also used for neural fitted $Q$-iteration for 20 000 epochs. We observe that the neural network produces an outcome that is consistent with prescribed treatments only 82 times. This corresponds to a matching accuracy of 32%. Similar results are obtained when using the data from trials 2 - 10 from the previous experiment as test data. These results are shown in Table 5.11. There are also a few cases where the neural network produces a single drug as output. This is unusual as most HIV-infected individuals are prescribed combinations of antiretrovirals to prevent the development of drug-resistant HIV strains.

| **Trial** | **Matches using an artificial neural network** |
|---|---|
| 1 | 82 |
| 2 | 97 |
| 3 | 64 |
| 4 | 75 |
| 5 | 79 |
| 6 | 78 |
| 7 | 94 |
| 8 | 62 |
| 9 | 64 |
| 10 | 86 |
| **Average** | 78.1 |
| **% consistency** | 30.51 |

Table 5.11: Average consistency between learned actions and actions taken by clinicians using neural networks alone.

Upon closer investigation, a number of comparisons can be made between using supervised learning alone and using RL methods. Perhaps the most interesting of these comparisons arises when we examine the case where a patient suddenly transitions from a very good state of health to a very bad one or vice versa. Consider the specific example of a patient with a $CD4^+$ count of 350 cells/mm$^3$ and a viral load of 40 copies/ml currently being treated with EFV and 3TC. The patient's condition under this treatment deteriorates rapidly resulting in a new $CD4^+$ count of 120 cells/mm$^3$ and a viral

load of 310 064 copies/ml.[4] At this point the the patient's treatment is altered to a new combination of drugs 3TC, tdf and NVP. The fact that a patient's treatment is altered at this point is not unusual as the viral load changes from being virtually undetectable to being reasonably high. While it is not always the case, a change of this magnitude in the viral load, is usually the first indication that a patient is developing drug resistance to one or more of the prescribed drugs.[5] Because there are several examples in the training data where a patient is under the treatment of EFV and 3TC, the neural network method is able to detect this and hence suggests using the same combination of drugs that was initially prescribed, as opposed to switching to the new combination of drugs prescribed. The same patient is suggested a combination of 3TC, tdf and d4T when using neural fitted $Q$-iteration and fitted $Q$-iteration with extra trees, while we observe an exact match in the treatment suggested by LSPI. Similar results can be seen when dealing with longer patient trajectories. This example highlights a key difference between classical supervised learning techniques and RL methods for HIV drug scheduling: supervised learning methods have no way of identifying what happens to a patient after taking a particular course of action since each sample of data is treated separately without any reference to another. Conversely, RL methods use state and subsequent state information to learn the value of making a particular transition between these states. This is crucial since a patient's history may be embedded into their state information. The fact that RL methods make use of a long-term horizon means that at each iteration of the learning techniques, we can look further and further into the future to choose actions appropriately. This is a crucial piece of information as an action that is beneficial in the immediate future may not necessarily be beneficial in the long run. It is not possible to make such an assessment using standard supervised learning techniques.

## 5.6 Discussion in relation to research questions

Having completed experimentation, we can now answer the research questions posed in Section 4.3 of Chapter 4.

1. It is possible to model the problem of HIV drug scheduling in terms of an MDP where the state space $\mathcal{S}$ is comprised of variables that serve as indicators of a patient's health at a particular time. In this research, we have chosen $CD4^+$ T-lymphocyte counts and viral loads as our variables however, it is also possible to extend the state space to include several other wellness indicators. The action space $\mathcal{A}$ consists of the drugs used to treat a patient at a particular time. We have constructed a seven-tuple indicating whether the drugs EFV, 3TC, d4T, TDF, NVP, LPV/r and AZT are at use at a certain time in treatment. Like the state space, these drugs may be exchanged for other drugs. The inclusion of newly developed drugs into the action space would require the dimensionality of

[4] These are the values of the state variables without normalization.

[5] At this point a doctor may choose to run one or more drug resistance tests to ascertain whether such resistance exists.

the action vector to be increased. The introduction of more state variables to the MDP may mean that features must be extracted from the set of data first before applying the batch algorithms to these features.[6] Alternatively, dimensionality reduction techniques could be applied to the state space if necessary. A reward function for this MDP is based on rewarding those instances where actions result in improved T-cell counts or lowered viral loads and penalizing those that do not.

2. The performances of the learning techniques vary when applied to the problem of determining suitable drug combinations for a real patient. By performing ten-fold cross validation on sets of patient data, we observe that all three learning methods are capable of producing outcomes that are consistent with strategies that are currently in place. Fitted $Q$-iteration using extremely randomized trees produces these outcomes the most frequently in comparison to the other two learning methods. However, strategies to improve overall learning among all three methods can be introduced. All three learning methods demonstrate the ability to suggest drug combinations that are not unusual and are commonly used in the real world. This highlights their potential to be coupled with clinical research to enhance treatment design.

3. When we apply the learning techniques to simulated patient data, we observe that all three methods are successful at improving patient wellness. The strategies determined by LSPI and fitted $Q$-iteration using extra trees are fairly similar as opposed to the strategy determined using neural fitted $Q$-iteration. Despite the overall improvement observed, the strategies determined by LSPI and fitted $Q$-iteration involve a lot of switching between using RTIs and PIs with few periods of stability. This may not be feasible since frequent switching can have adverse effects on real patients. Overall, the model used to simulate patient data is too general in terms of the actions it considers, and would need to be modified to include more drugs or classes of drugs for more meaningful results.

4. The batch RL techniques considered in this research have demonstrated good performance, in general, when applied to the benchmark domains. All three methods are able to produce optimal policies that succeed in reaching the goal state for the mountain car task. Experimentation on the acrobot task demonstrated the performances of the learning methods in a higher-dimensional state space. Results revealed similar performance between both versions of fitted $Q$-iteration. In terms of speed, the extra trees implementation is very slow. The speeds of the other learning methods vary with the size of neural network and number of basis functions used. For large data sets, it may be worth considering using better hardware to run these learning methods on. The use of benchmark domains in this research has not only helped compare the batch RL techniques on relatively well-behaved data sets, but also allowed us to gain a better understanding of learning in continuous state spaces.

[6]The interested reader should refer to Lange et al. (2012) for these details.

## 5.7 Conclusion

This chapter presented the major results of this research. Results reveal that batch reinforcement learning methods vary in their performances across different domains. For the case of simulated data, all three methods have shown the ability to improve a patient's outcomes by scheduling appropriate drug use. In a real medical setting, results suggest that batch learning methods have the potential of assisting clinicians in decision-making however, more testing using larger data sets and inclusion of more state and action variables is recommended.

# Chapter 6

# Conclusions and Future Work

Discovering effective treatment strategies for HIV-infected individuals remains one of the main challenges in medical research. Existing treatments combine anti-HIV drugs under HAART to inhibit the development of drug-resistant HIV strains. Despite the fact that eradicating the virus using HAART may not be possible, improved strategies for choosing suitable drug combinations and scheduling drugs are necessary for better control of viral infection. Such strategies may also enable treatments to be simplified, and drug combinations to be re-formulated to reduce a patient's pill burden and improve adherence to medication.

In this dissertation, we have demonstrated the application of batch reinforcement learning to the problem of drug scheduling and HIV treatment design. Specifically, we have used the techniques of fitted $Q$-iteration with extremely randomized trees, neural fitted $Q$-iteration and LSPI to address this problem. In particular, we have shown that it is possible to formulate the problem of HIV drug scheduling as an MDP by using a patient's wellness indicators as state variables and considering drug combinations as actions. Results from experimentation on real data suggest that each learning technique has its associated benefits and disadvantages. However, all three techniques have the potential of suggesting drug combinations that are reasonably consistent with those prescribed by medical professionals for HIV patients. The implications of this are tremendous: overall, these techniques could be coupled with the expertise of clinicians to improve treatment design.

We have also addressed the research questions formulated in Chapter 4. The performances of all three batch reinforcement learning techniques were recorded and compared when applied to the benchmark domains of mountain car and swing-up acrobot. Results demonstrate varied performance among all three techniques in terms of speed and the quality of solutions produced. Testing these methods using simulated HIV data revealed that all the techniques were suitable in determining drug strategies that improve the health status of an infected individual however, limitations in the model used for simulation mean that the strategies determined are not entirely realistic nor practical.

This research opens many avenues for future work. From a theoretical perspective, it would be worth investigating the use of a more complex reward function that takes

into account variations or perhaps smaller differences in the state variables. It would also be worth looking into using more basis functions for the LSPI technique or improve basis function selection in general. The inclusion of more suitable basis functions could allow for better function approximation particularly when applying LSPI to real data. Similar improvements could be made to neural fitted $Q$-iteration by considering changes to the neural networks used for training.

From a practical perspective, HIV is a virus that invades and infects the body using a number of mechanisms. Each of these mechanisms involves a series of complex interactions between several biological components of the immune system. Furthermore, an individual's response to HIV is dictated by a number of factors such as cytokine production, the presence of other infections and whether there is repeated exposure to the virus. For these reasons, perhaps the biggest avenue for future research in this area involves extending the MDP we have used here to include many more state variables such as complete white blood cell counts, HIV strain information, information about the occurrence of other infections such as tuberculosis, and even variables that take into account differences between individuals such as age, sex, etc. These differences may play an important role in explaining why some individuals have better responses to viral infection than others. It is more than likely that including more variables would improve quality of learning overall. We have only considered the use of seven major drugs for the treatment of HIV however, the same techniques can be applied to the complete list of HIV drugs, if larger data sets using each of these drugs are available.

Finally, we have shown the potential for batch reinforcement learning techniques to be used for improving treatments for people suffering from HIV, largely because of its prevalence in South Africa. These techniques, if used in a clinical trial setting, could provide insight into the combinations of drugs that are suitable for these individuals and identify candidates of drugs for treatment simplification. However, the application of these techniques to other medical domains such as treatment of epilepsy, types of cancers or other life-threatening illnesses is also possible, and may prove incredibly useful for treatment design.

# Appendix A

# Benchmark Domains

## A.1 The Acrobot Swing-Up Control Problem

The dynamics of the acrobot swing-up control problem may be described by the system of Equations A.1.1. In particular, the robot arm contains two links: the first rotates freely about the joint whilst the second is actuated by a torque applied to the joint. The state space can be described in terms of the four continuous state variables, $\theta_1$, $\theta_2$, $\dot{\theta}_1$ and $\dot{\theta}_2$ where $\theta_1$ is the angular position of the first link in relation to the joint, and $\theta_2$ is the angular position of the second link in relation to the first; $\dot{\theta}_1$ and $\dot{\theta}_2$ are the angular velocities of each link respectively. A reward of -1 is given at each time step, $t = 0.05$, until the goal is reached.

$$
\begin{aligned}
\ddot{\theta}_1 &= -{d_1}^{-1}(d_2\ddot{\theta}_2 + \phi_1) \qquad \text{(A.1.1)}\\
\ddot{\theta}_2 &= \left(m_2 l_{c2}^2 + I_2 - \frac{d_2^2}{d_1}\right)^{-1}\left(\tau + \frac{d_2}{d_1}\phi_1 - \phi_2\right)\\
d_1 &= m_1 l_{c1}^2 + m_2\left(l_1^2 + l_{c2}^2 + 2l_1 l_{c2}\cos\theta_2\right) + I_1 + I_2\\
d_2 &= m_2\left(l_{c2}^2 + l_1 l_{c2}\cos\theta_2\right) + I_2\\
\phi_1 &= -m_2 l_1 l_{c2}\theta_2^2\sin\theta_2 - 2m_2 l_1 l_{c2}\dot{\theta}_2\dot{\theta}_1\sin\theta_2 + (m_1 l_{c1} + m_2 l_1)\, g\cos(\theta_1 - \pi/2) + \phi_2\\
\phi_2 &= m_2 l_{c2} g\cos(\theta_1 + \theta_2 - \pi/2)
\end{aligned}
$$

Here $g = 9.8$ is the gravitational force. The joint positions $\theta_1$ and $\theta_2$ can take any value; the angular velocities $\dot{\theta}_1$ and $\dot{\theta}_2$ are restricted to $[-4\pi; 4\pi]$ and $[-9\pi; 9\pi]$ respectively. The rest of the parameters are described in Table A.1

| Parameter | Description of parameter | Value |
|---|---|---|
| $m_1$, $m_2$ | masses of links | 1.0 |
| $l_1$, $l_2$ | lengths of links | 1.0 |
| $l_{c1}$, $l_{c2}$ | lengths to mass centre | 0.5 |
| $I_1$, $I_2$ | link inertias | 1.0 |
| $\tau$ | torque | {-1.0, 0.0, 1.0} |

Table A.1: Parameters of the swing-up acrobot control problem.

## A.2 Mountain Car

The mountain car system dynamics can be expressed in terms of the following set of equations (directly as in Ernst et al. (2005)):

$$\begin{aligned} \frac{\mathrm{d}p}{\mathrm{d}t} &= v \\ \frac{\mathrm{d}v}{\mathrm{d}t} &= \frac{a}{m + Hill'(p)^2} - \frac{gHill'(p)}{1 + Hill'(p)^2} - \frac{v^2 Hill'(p) Hill''(p)}{1 + Hill'(p)^2}. \end{aligned} \tag{A.2.1}$$

Here $m = 1$, $g = 9.81$ and the function $Hill(p)$ is defined as:

$$Hill(p) = \begin{cases} p^2 + p & p < 0 \\ \frac{p}{\sqrt{1+5p^2}} & p \geq 0 \end{cases} \tag{A.2.2}$$

For simulation purposes we choose $t = 0.1s$ and generate sample sets using Euler's method with $0.001s$ as an integration step.

# Appendix B

# Additional Results

## B.1 Experimentation using a set of simulated HIV data of a smaller size

This appendix presents the results from using smaller sample sets of simulated data for learning. As in Section 5.4 of Chapter 5, we compare the outcomes of each learning algorithm when applied to HIV data simulated using the model in Section 4.5.2 in Chapter 4. We are particularly interested in whether performance of the learning methods degrades as a result of using a smaller set of samples for learning.

To perform this experiment, we use an iterative data generation and testing procedure virtually identical to the experimentation in Section 5.4. Once again we assume that patients are monitored every five days for a period of 1 000 days. At the first iteration, we generate the data for two patients in the unhealthy steady state. Thereafter, we record the status of each patient's health every five days according to the quantities of the state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$. At each time step, we select a random treatment action from the set of actions {RTI on and PI on, RTI on and PI off, RTI off and PI on}. From this, we obtain a trajectory $(s_0, a_0, s_1, a_1, \ldots, s_{200})$ corresponding to the course of action taken on the patient over 1 000 days. This trajectory produces 200 samples of the form $(s_t, a_t, s_{t+1})$. Hence, by the end of the first iteration, a total of 400 samples are available. At this point we run each of the batch RL algorithms on the set of 400 samples for 50 iterations to produce a policy $\widehat{\pi}^*_{50}(s_0)$ where $s_0$ is the unhealthy steady start state.

At the second iterative step, we generate the data for two new patients, again from the unhealthy steady state. We once again record their state data every five days for 1 000 days. Instead of randomly selecting the medication taken by a patient every five days, we consider the optimal action suggested by the policy $\widehat{\pi}^*_{50}$ that was obtained from the original set of 400 samples. We take this optimal action 70% of the time and choose a random action for the remaining 30%. Once again we apply each of the batch algorithms to the new set of 400 samples for 50 iterations and obtain a new optimal policy for the larger sample set.

At the third iterative step, we generate the data for another two new patients from

the unhealthy steady state and record the necessary state data. We consider the action suggested by the optimal policy obtained from the set of 600 samples and take this action 85% of the time; for the remaining 15%, we choose a random action. We repeat this procedure of training on the sample set, determining an optimal policy and using this policy to generate the data of 2 new patients (whose data is added to the original sample set) five times. After the fifth iteration, we use the optimal policy $\widehat{\pi}^*_{50}$ that was obtained from the set of 2 000 samples to generate the data for one patient (once again starting at unhealthy stationary equilibrium). This time, we take only the action suggested by $\widehat{\pi}^*_{50}$. The resulting trajectory obtained represents the evolution of a patient's health condition under the policy $\widehat{\pi}^*_{50}$ from a 2 000 sample set. We determine such a trajectory for each of the learning methods under consideration. For the fitted $Q$-iteration algorithm using extra trees, we build a set of 50 trees at each iteration like Ernst et al. (2006); for the neural fitted $Q$-iteration algorithm, we use a neural network with 9 input nodes corresponding to the state and action variables, 100 hidden nodes and 1 output node. When applying LSPI, we use 9 000 randomly centered Gaussian radial basis functions with $\sigma^2 = 4$ as before.

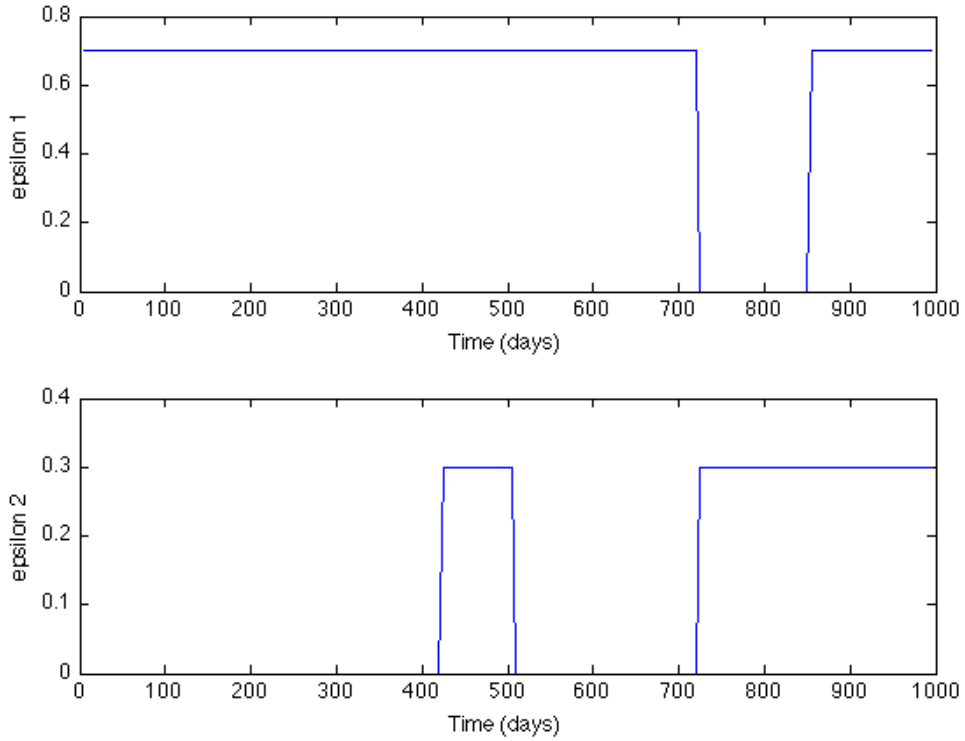



Figure B.1: Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using neural fitted $Q$-iteration with $|\mathcal{F}| =$ 2 000 samples.

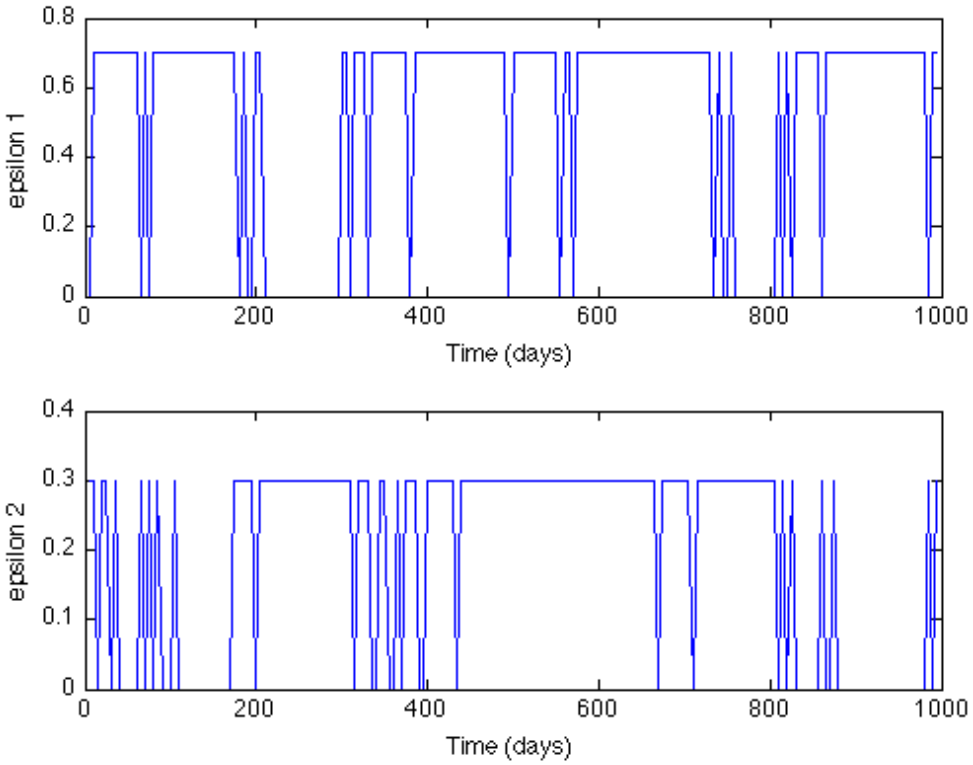



Figure B.2: Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using fitted $Q$-iteration with extra trees where $|\mathcal{F}| = 2\ 000$ samples.

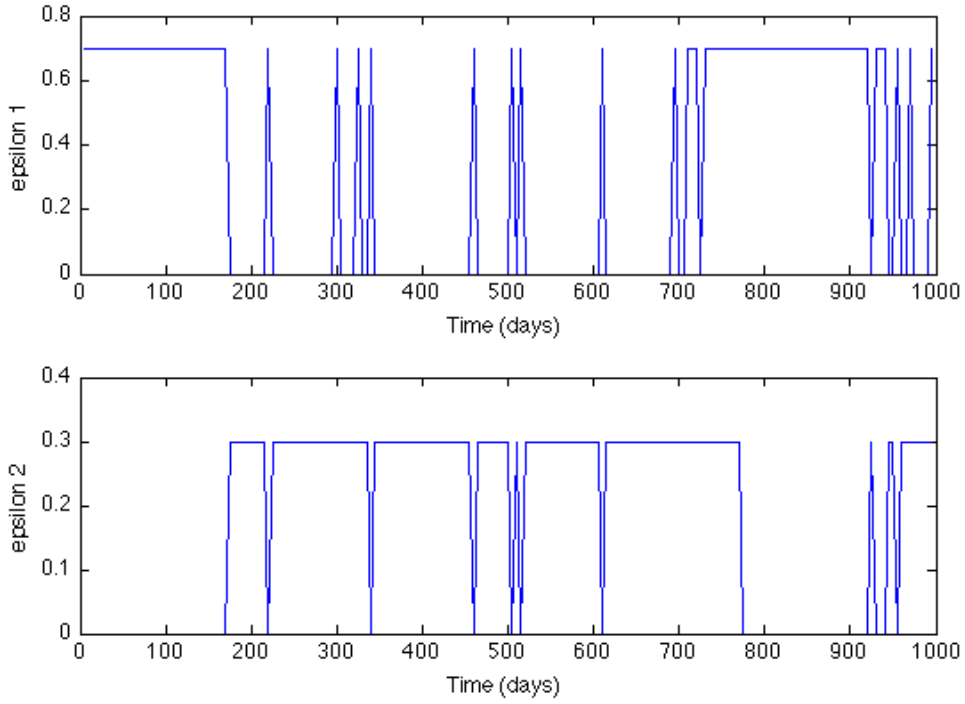



Figure B.3: Representation of the treatment strategy, $\widehat{\pi}^*_{50}$, in terms of $\epsilon_1$ and $\epsilon_2$ for a typical patient in an unhealthy steady state using LSPI and $|\mathcal{F}| = 2\ 000$ samples.

Figures B.4, B.5 and B.6 represent the changes in the state variables for a patient being treated from unhealthy stationary equilibrium for a period of 1 000 days following $\widehat{\pi}^*_{50}$ using neural fitted $Q$-iteration, fitted $Q$-iteration with extra trees and LSPI respectively. The corresponding courses of action under each learning method are provided in Figures B.1, B.2 and B.3 respectively.

Overall, the results from using a smaller sample set for learning are fairly similar to those results produced from using a significantly larger set of samples as in Chapter 5. This probably serves as evidence that there are many repetitions in the larger sample set used for experimentation in Section 5.4, as a result of a small action space. We still observe an improvement in the overall health of a patient in terms of the $CD4^+$ count, viral load, macrophage counts and cytotoxic T-cell counts across all three learning techniques. However, the courses of action suggested by each learning method are significantly less erratic. While these sort of treatment strategies are certainly more realistic, this may also be a direct result of using fewer samples with less variation between them.

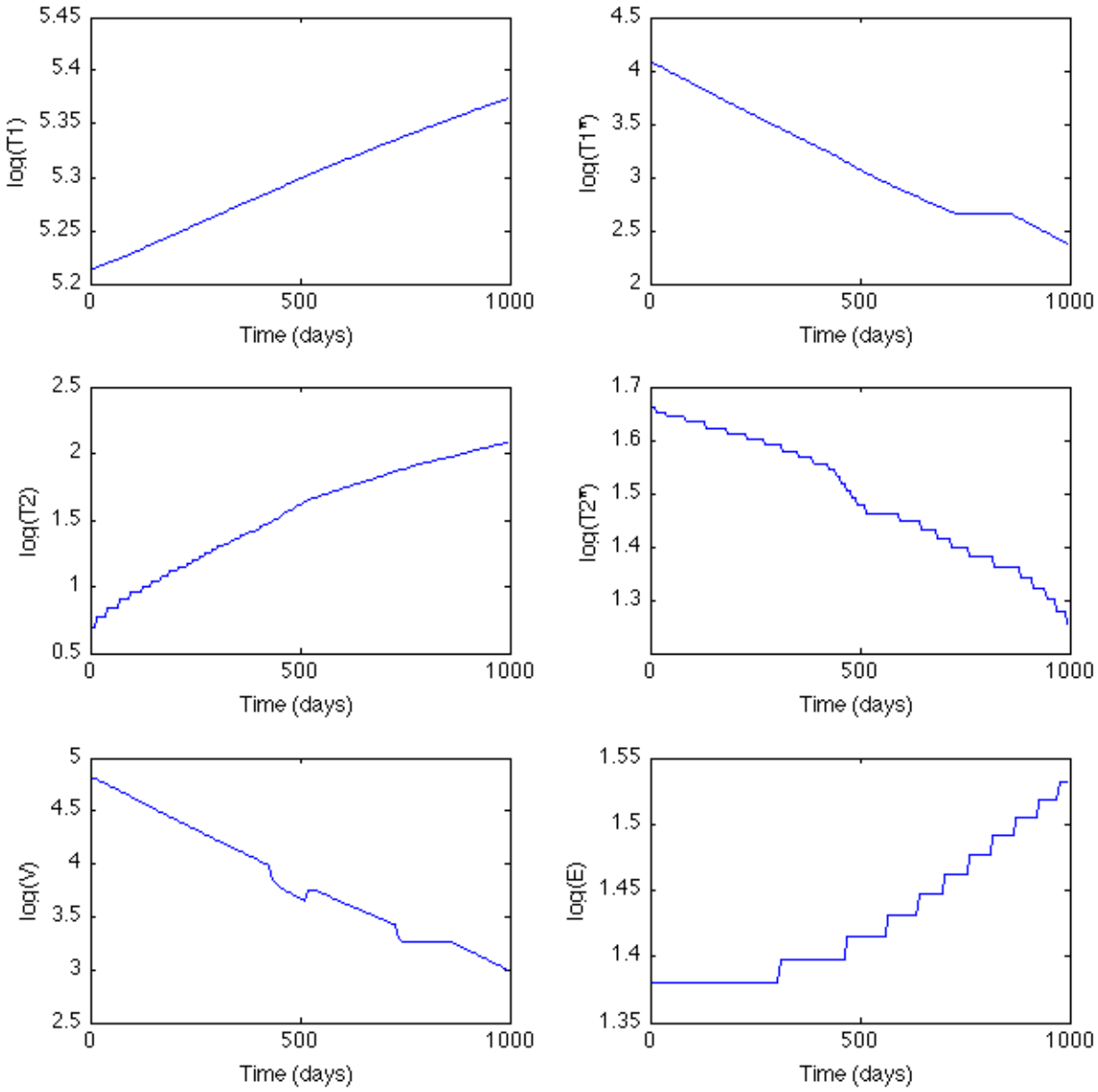



Figure B.4: Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying neural fitted $Q$-iteration across a smaller sample set of $|\mathcal{F}| = 2\ 000$ samples.

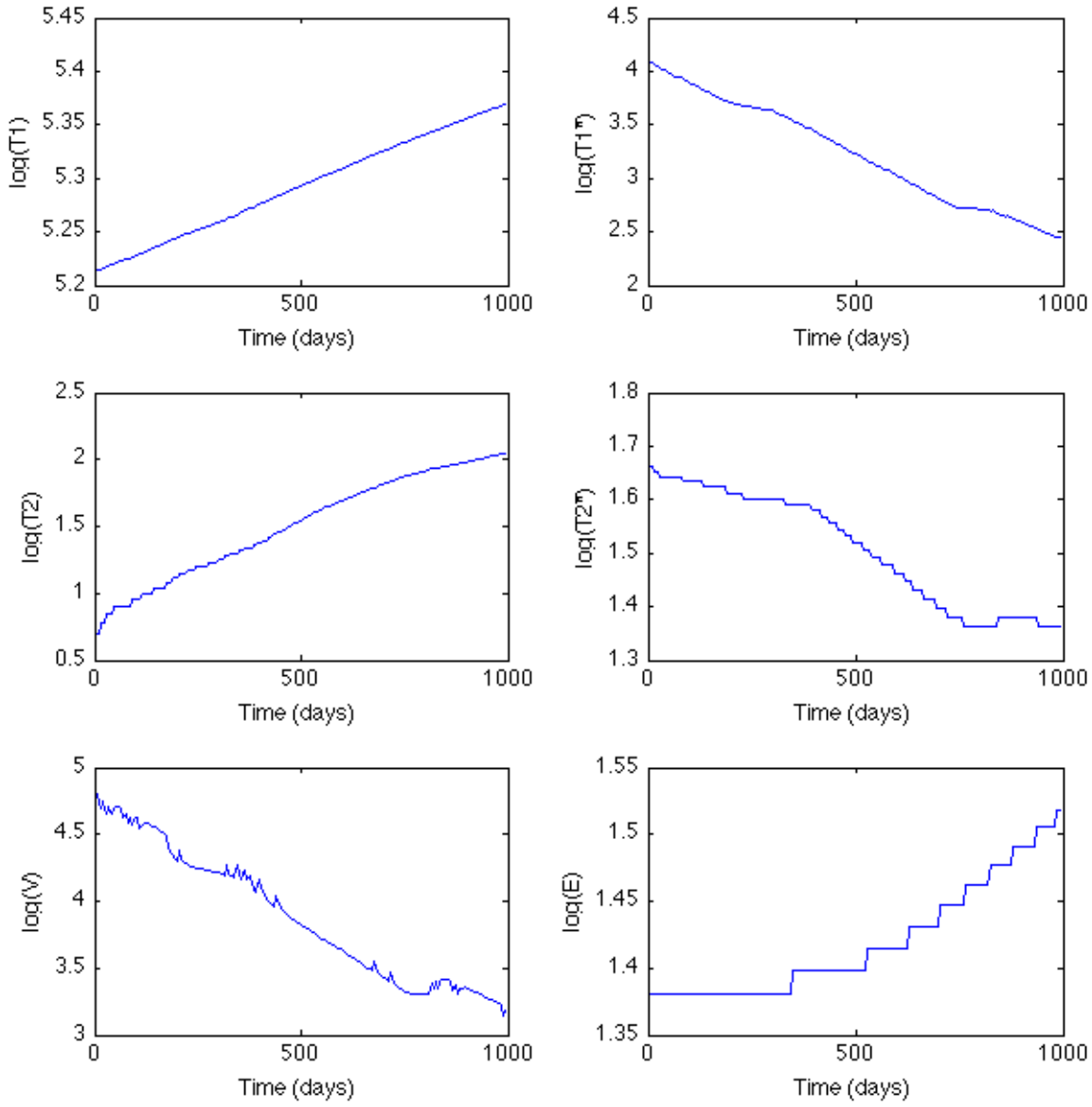



Figure B.5: Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying fitted $Q$-iteration with extra trees across a smaller sample set of $|\mathcal{F}| = 2\ 000$ samples.

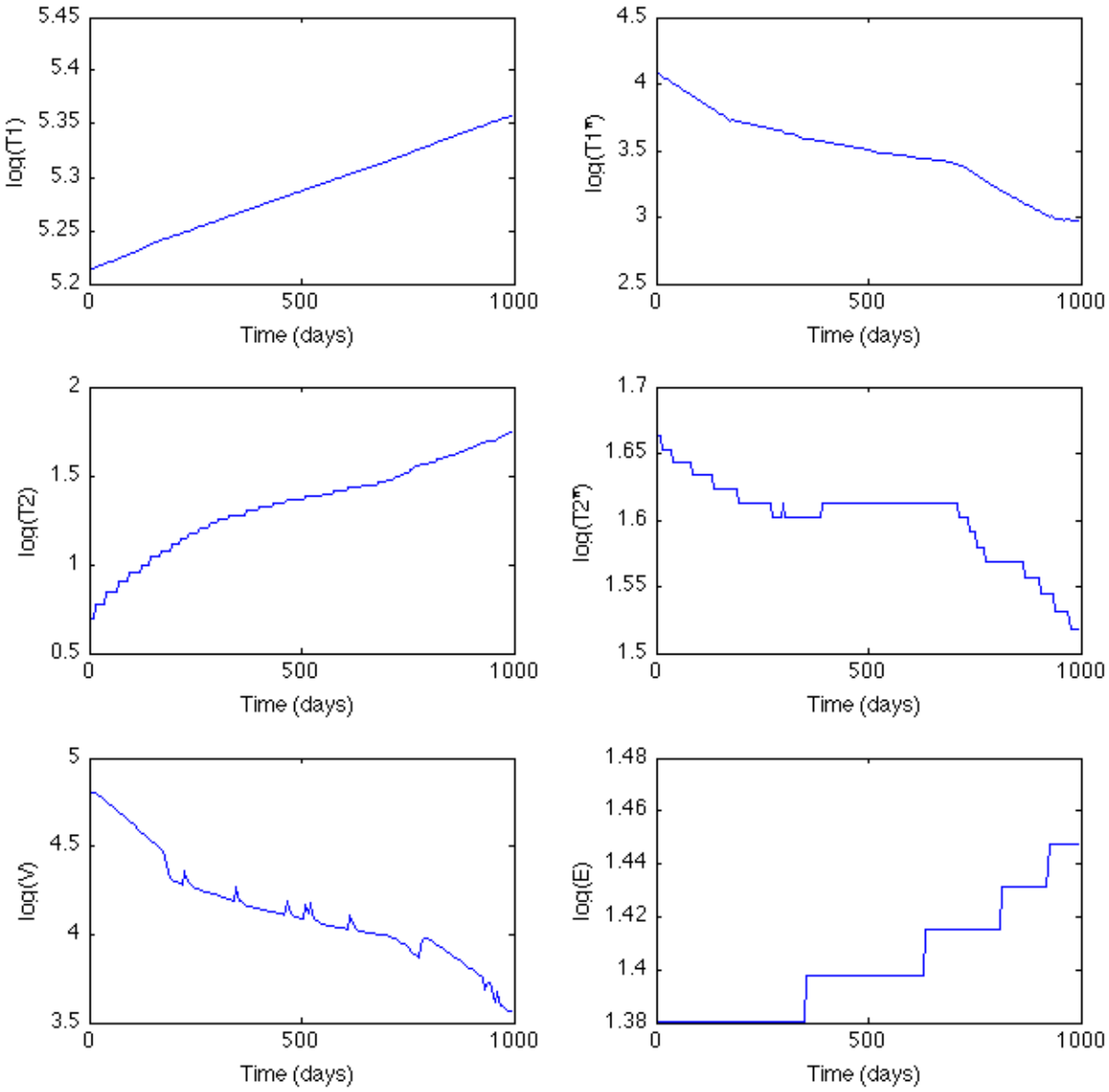



Figure B.6: Graphs representing the evolution of state variables $(T_1, T_2, T_1^*, T_2^*, V, E)$ over 1 000 days for a patient being treated from an unhealthy steady state when applying LSPI across a smaller sample set of $|\mathcal{F}| = 2\ 000$ samples.

# References


J. Abadi, M. Rosenberg, J. Dobroszycki, J. Sansary, G. Fennelly & A. Wiznia. Partial treatment interruption of protease inhibitor-based highly active antiretroviral therapy regimens in HIV-infected children. *AIDS*, 41(3):298–303, 2006.

A. Abbas & A. Lichtman. *Basic Immunology: Functions and Disorders of the Immune System.* Saunders Elsevier, Philadelphia, 3rd edition, 2009.

B. Adams, H. Banks, M. Davidian, H. Kwon, H. Tran & S. Wynne. HIV dynamics: Modeling, data analysis, and optimal treatment protocols. *Journal of Computational and Applied Mathematics*, 184(1):10–49, 2005.

B. Adams, H. Banks, H. Kwon & H. Tran. Dynamic multidrug therapies for HIV: Optimal and STI control approaches. *Mathematical Biosciences and Engineering*, 1: 223–241, 2004.

A. Antos, R. Munos & C. Szepesvari. Fitted Q-iteration in continuous action-space MDPs. *NIPS*, 2007.

J. P. Archer. *The Diversity of HIV-1.* PhD thesis, University of Manchester, 2008.

J. G. Bartlett. Ten years of HAART: Foundation for the future. The 13th Conference on Retroviruses and Opportunistic Infections, February 2006.

R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ., 1957.

B. Bethke, J. P. How & A. Ozdaglar. Kernel-based reinforcement learning using bellman residual elimination. *Journal of Machine Learning Research (to appear)*, 2008.

A. Billich. AIDSVAX. vaxgen. *Current Opinions in Investigational Drugs*, 2(9):1203–1208, September 2001.

A. Boasso & G. Shearer. Chronic innate immune activation as the cause of HIV-1 immunopathogenesis. *Clinical Immunology*, 126(3):235–242, March 2008.

S. Bonhoeffer, M. Rembiszewski, G. Ortiz & D. Nixon. Risks and benefits of structured antiretroviral drug therapy interruptions in HIV-1 infection. *AIDS*, 14:2313–2322, 2000.

G. Boone. Minimum-time control of the acrobot. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, pages 3281–3287, New Mexico, 1997.

P. Borrow. Innate immunity in acute HIV-1 infection. *Current Opinions in HIV AIDS*,

6(5):353–363, September 2011.

L. Buşoniu, R. Babuška, B. De Schutter & D. Ernst. *Reinforcement Learning and Dynamic Programming using Function Approximators*. Taylor & Francis CRC Press, Boca Raton, FL, 2010.

L. Buşoniu, A. Lazaric, M. Ghavamzadeh, R. Munos, R. Babuška & B. De Schutter. Least-squares methods for policy iteration. In *Reinforcement Learning*, pages 75–109. Springer, 2012.

D. Callaway & A. Perelson. HIV-1 infection and low steady state viral loads. *Bulletin of Mathematical Biology*, 64(1):29–64, 2002.

N. Campbell & J. Reece. *Biology*. Pearson Education Inc. as Benjamin Cummings, San Francisco, CA, seventh edition, 2005.

A. Cann & J. Karn. Molecular biology of HIV: New insights into the virus life-cyle. *AIDS*, 3:19–34, 1989.

P. Carter. Midlands technical college - specific defenses of the host: The immune response, 2011. URL: `http://classes.midlandstech.edu/carterp/Courses/bio225/chap17/`.

J. Chinen & W. T. Shearer. Molecular virology and immunology of HIV infection. *J Allergy Clin. Immunol.*, 110(2):189–198, August 2002.

V. Douce, A. Janossy, H. Hallay, S. Ali, R. Riclet, O. Rohr & C. Schwartz. Achieving a cure for HIV infection: do we have reason to be optimistic? *Journal of Antimicrobial Chemotherapy*, February 2012.

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

D. Ernst, G. B. Stan, J. Gonçalves & L. Wehenkel. Clinical data based optimal STI strategies for HIV: A reinforcement learning approach. In *Proceedings of the 45th IEEE Conference on Decision and Control*, volume 25, pages 2302–2308, 2006.

P. Fisher. UCSF Immunology Module, August 2011. URL: `http://missinglink.ucsf.edu/lm/immunology_module/prologue/objectives/obj05.html`.

R. Fonteneau. Raphael fonteneau - home page, 2009. URL: `https://sites.google.com/site/raphaelfonteneau/code`.

T. Geijtenbeek, D. Kwon, R. Torensma, S. van Vliet, G. van Duijnhoven, J. Middel, I. Cornelissen, H. Nottet, V. Kewalramani, D. Littman, C. Figdor & Y. van Kooyk. DC-SIGN, a dendritic cell-specific HIV-1-binding protein that enhances trans-infection of T-cells. *Cell*, 100(5):587–597, 2000.

P. Geurts, D. Ernst & L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, April 2006.

G. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1999.

R. M. Gulick, J. W. Mellors, D. Havlir, J. J. Eron, C. Gonzalez, D. McMahon, D. D. Richman, F. T. Valentine, L. Jonas, A. Meibohm, E. A. Emini, J. A. Chodakewitz, P. Deutsch, D. Holder, W. A. Schleif & J. H. Condra. Treatment with indinavir, zidovudine, and lamivudine in adults with human immunodeficiency virus infection and prior antiretroviral therapy. *New England Journal of Medicine*, 337(11):734–739, 1997.

S. M. Hammer, K. E. Squires, M. D. Hughes, J. M. Grimes, L. M. Demeter, J. S. Currier, J. J. Eron, J. E. Feinberg, H. H. Balfour, L. R. Deyton, J. A. Chodakewitz & M. A. Fischl. A controlled trial of two nucleoside analogues plus indinavir in persons with hiv infection and CD4 cell counts of 200 per cubic millimeter or less. *The New England Journal of Medicine*, 337:725–733, 1997.

A. Hatzigeorgiou & M. Megraw. *Global Optimization: Scientific and Engineering Case Studies*, chapter 7: Computational Analysis of Human DNA Sequences: An Application of Artificial Neural Networks, pages 172–173. Springer, New York, NY, 2006.

T. Hoffman, B. Schölkopf & A. J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

V. Kumar, A. Abbas, N. Fausto & R. Mitchell. *Basic Pathology*. Saunders Elsevier, Philadelphia, 8th edition, 2007.

M. Lagoudakis & R. Parr. Least squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

R. P. Lai & J. Heeney. Perspectives in HIV vaccine development: What we have learned and how we proceed forward - review article. *J AIDS Clinic Res*, S8(004), 2012.

S. Lange, T. Gabel & M. Riedmiller. *Batch Reinforcement Learning*, volume 12 of *Adaptation, Learning and Optimization*, chapter 2, pages 45–73. Springer, Berlin, Heidelberg, 2012.

S. Lawrence & W. El-Sadr. New perspectives in HIV treatment interruption: The SMART study. *The PRN Notebook*, 11(2):8–9, October 2006.

R. Lihana, D. Ssemwanga, A. Abimiku & N. Ndembi. Update on HIV-1 diversity in africa: A decade in review. *AIDS*, 14:83–100, 2012.

D. Mann & M. Ward. Virology lecture synopsis, 2006. URL: `http://www.southampton.ac.uk/~ceb/teaching/2005/206-8.htm`.

S. Marsland. *Machine Learning: An Algorithmic Perspective*. Machine Learning and Pattern Recognition Series. Chapman and Hall/CRC Press, Boca Raton, FL, 2009.

G. Mayer. Microbiology and Immunology Online - University of South Carolina, 2011. URL: `http://pathmicro.med.sc.edu/ghaffar/innate.htm`.

A. McMichael, P. Borrow, G. Tomaras, N. Goonetilleke & B. Haynes. The immune response during acute HIV-1 infection: clues for vaccine development. *Nat. Rev. Immunol.*, 10(1):11–23, 2010.

T. Mogensen, J. Melchjorsen, C. Larson & S. Paludan. Review: Innate immune recog-

nition and activation during HIV infection. *Retrovirology*, 7(54), 2010.

R. Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML03)*, pages 560–567, Washington, District of Columbia, 2003.

D. Ormoneit & S. Sen. Kernel-based batch reinforcement learning. *Machine Learning*, 49:161–178, 1999.

G. Pantaleo & A. Fauci. Immunopathogenesis of HIV infection. *Annual Review of Microbiology*, 50:825–854, 1996.

R. Paranjape. Immunopathogenesis of HIV infection. *The Indian Journal of Medical Research*, 50:240–255, 2005.

J. Pazis & M. Lagoudakis. Reinforcement learning in multidimensional continuous action spaces. In *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 97–104, April 2011.

A. Pozniak. Advances in highly active antiretroviral therapy – simplified treatment regimens. Touch Briefings - European Infectious Diseases, 2007.

C. Rasmussen & C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

S. Rerks-Ngarm, P. Pitisuttithum, S Nitayaphan, J. Kaewkungwal, J. Chiu, R. Paris, N. Premsri, C. Namwat, M. de Souza & E. et al Adams. Vaccination with ALVAC and AIDSVAX to prevent HIV-1 infection in thailand. *New England Journal of Medicine*, 361(23):2209–2220, 2009.

M. Riedmiller. Rprop - description and implementation details. Technical report, Institution f. Logik, Komplexität u. Deduktionssysteme, University of Karlsruhe, Frankfurt, Germany, January 1994.

M. Riedmiller. Neural fitted Q-iteration - first experieces with a data efficient neural reinforcement learning method. *Machine Learning*, ECML 2005:317–328, 2005a.

M. Riedmiller. Neural reinforcement learning to swing-up and balance a real pole. 2005b.

M. Riedmiller. Unit 12: Optimizing learning - (neural) fitted q iteration. Lecture Slides, October 2010.

M. Riedmiller & H. Braun. A direct adaptive method for faster backprop- agation learning: The RPROP algorithm. In H. Ruspini, editor, *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 586 – 591, San Francisco, CA, 1993.

D. E. Rumelhart, G. E. Hinton & R. J. Williams. Learning representations by backpropogating errors. *Nature*, 323(6088):533–536, 1986.

M. Santiago, F. Range & B. F. Keele et al. Simian immunodeficiency virus infection in free-ranging Sooty Mangabeys from the Taï forest, Côte d'Ivoire: Implications for the origin of epidemic HIV-2. *Journal of Virology*, 79(19):12515–27, 2005.

R. P. Sekaly. The failed HIV merck vaccine study: a step back or a launching point for future vaccine development. *The Journal of experimental medicine*, 205(1):7–12, 2008.

R. Siliciano & W. C. Greene. HIV latency. *Cold Spring Harbor Perspectives in Medicine*, 1(1), 2011.

A. Sivro, D. Stein & L. McKinnon. *Purple Paper: Innate Immunity to HIV*. National Collaborating Centre for Infectious Diseases, August 2010.

R. Smith. 2013 HIV drug chart, February 2013. URL: `http://positivelyaware.com/2013/13-02/pdfs/2013DrugChart.pdf`.

A. Sousa, J. Carneiro, M. Meier-Schellersheim, Z. Grossman & R. Victorino. CD4 T-cell depletion is linked directly to immune activation in the pathogenesis of HIV-1 and HIV-2 but only indirectly to the viral load. *The Journal of Immunology*, 169(6): 3400–3406, 2002.

M. W. Spong. The swing up control problem for the acrobot. *IEEE Control Systems Magazine*, 15(2):45–55, 1995.

M. W. Spong. Underactuated mechanical systems. In *Control Problems in Robotics and Automation*, pages 135–150. Springer, Berlin Heidelberg, 1998.

P. Stayley. AIDSMEDS: Your ultimate guide to HIV care, November 2012. URL: `http://www.aidsmeds.com/archive/`.

H. Streeck & D. F. Nixon. T-cell immunity in acute HIV-1 infection. *The Journal of Infectious Diseases*, 202(S2):S302–S308, 2010.

C. Strobl, J. Malley & G. Tutz. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological methods*, 14(4):323, 2009.

R. Sutton. Learning to predict by methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

R. S. Sutton & A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

R. Swanstrom & J. Coffin. HIV-1 pathogenesis: The virus. *Cold Spring Harbor Perspectives in Medicine*, 2(12):doi: 10.1101/cshperspect.a007443, November 2012.

D. A. Tamarkin. Cell-mediated immunity, 2011. URL: `http://faculty.stcc.edu/AandP/AP/AP2pages/Units21to23/immune/cellmedimm.htm`.

B. Tanner. The mountain car task, October 2009. URL: `http://library.rl-community.org/wiki/Mountain_Car_(Java)`.

G. Taylor & R. Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, pages 1017–1024, Montreal, Canada., 2009.

D. von Laer & G. Brandenburg. Gene therapy for HIV infection by intracellular immu-

nization with antiviral genes. *AIDS Rev*, 3:169–177, 2001.

B. Walker & A. McMichael. The T-cell response to HIV. *Cold Spring Harbor Perspectives in Medicine*, DOI: 10.1101/cshperspect.a007054, 2012.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

C. J. C. H. Watkins & P. Dayan. Q learning. *Machine Learning*, 8:279–292, 1992.

J. Weber. The pathogenesis of HIV-1 infection. *British Medical Bulletin*, 58:61–72, 2001.

L. Wehenkel. On uncertainty measures used for decision tree induction. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Granada, Spain, 1996 1996.

L. Wehenkel, D. Ernst & P. Geurts. Ensembles of extremely randomized trees and some generic applications. In *Proceedings of Robust Methods for Power System State Estimation and Load Forecasting*, 2006.

Y. Zhao, M. R. Kosorok & D. Zeng. Reinforcement learning design for cancer clinical trials. *Statistics in Medicine*, 28:3294–3315, 2009.