Technische Universität München
Fakultät für Informatik
Lehrstuhl VI: Echtzeitsysteme und Robotik

A Study in Direct Policy Search

Daniël Pieter Wierstra

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. B. Brügge, Ph.D
Prüfer der Dissertation:	1. Univ.-Prof. Dr. H. J. Schmidhuber 2. Univ.-Prof. Dr. H.-J. Bungartz

Die Dissertation wurde am 22.12.2009 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 30.03.2010 angenommen.

Abstract

Reinforcement learning in partially observable environments constitutes an important and challenging problem. Since many value function-based methods have been shown to perform poorly and even to diverge in non-Markovian settings, direct policy search methods may hold more promise. The aim of this thesis is to advance the state-of-the-art in direct policy search and black box optimization. Its contributions include a taxonomy of reinforcement learning algorithms and four new algorithms: (1) a novel algorithm which backpropagates recurrent policy gradients through time, as such learning both memory and a policy at the same time with the use of recurrent neural networks, in particular Long Short-Term Memory (LSTM); (2) an instantiation of the well-known Expectation-Maximization algorithm adapted to learning policies in partially observable environments; (3) Fitness Expectation-Maximization, a new black box search method derived from first principles; (4) Natural Evolution Strategies, an alternative to conventional evolutionary methods that uses a Monte Carlo-estimated natural gradient to incrementally update its search distribution. Experimental results with these four methods demonstrate highly competitive performance on a variety of test problems ranging from standard benchmarks to deep memory tasks to fine motor control in a car driving simulation.

Acknowledgements

I would like to thank my supervisor Jürgen Schmidhuber for his guidance and support. I would also like to thank my co-authors Tom Schaul, Sun Yi, Jan Peters and Alexander Förster for their assistance and help. Most of all, I would like to thank my wife Melanie for constant encouragement, love and support.

This research was supported in part by the Swiss National Foundation, under grant 200021-111968/1.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The problem of reinforcement learning (RL) is characterized by an *agent* which acts in an *environment*, while adapting its behavior in order to optimize the scalar *reinforcement* or *reward/punishment* signals that the environment emits in response to the agent's actions. In control theory, the agent is called the *controller*, while the environment is called the *plant*, but apart from naming conventions, the basic framework is essentially the same. The task of a reinforcement learning agent is to maximize some long-term measure of these rewards by learning an appropriate (preferably optimal) policy by trial and error. Learning generally[1] takes place in *episodes*, during which, at every time step, the agent perceives an observation produced by the environment, performs an action, and subsequently receives a reinforcement signal. After an episode is ended, the environment and the agent are both reset, and a new episode commences.

Actions performed in the environment are drawn from the agent's *policy*, conditioned upon its history of current and previous actions and observations (e.g. a robot's position, the angles of a robot arm or even some representation of its visual input). The goal of reinforcement learning is to learn this policy so as to optimize its rewards, but rewards may be delayed, that is, rewards may have to be accredited to actions many time steps past, which complicates learning considerably.

It is easy to see the general potential applicability of this general framework to various real world problems, ranging widely from robot control and investment decision making to medical treatment schemes and rocket guidance. Reinforcement learning, in theory, could replace manual programming labor, or could be used to fine-tune already-preprogrammed behaviors. Both constitute profitable usages of this technology. Consequently, the past three decades have seen considerable research effort being directed to the general

[1] Some reinforcement learning problems are cast as *life-long, infinite-horizon* problems where an agent needs to optimize average reward. However, in this thesis I only consider the more usual, episodic case.

problem of reinforcement learning.

Reinforcement learning methods have indeed met with some small measure of success. Notable examples include game play (e.g. Tesauro's TD-Backgammon algorithm (Tesauro, 1994)), fine motor control for robotic arms (Peters and Schaal, 2008b) and helicopter flight control (Ng et al., 2003). Disappointingly, however, RL methods have *not* actually found widespread acceptance in the application to real world domains. This may be due to various reasons. First, RL methods generally do not scale well to large and complex environments. Second, they show difficulty in dealing with continuity in both observation representation and multidimensional action representation. Third, they do not cope well with partial observability, which is a common characteristic of numerous realistic applications.

Having emphasized the potential benefits of powerful reinforcement learning algorithms, the apparent lack of more real world applications of RL inspires this thesis' aim to advance the state-of-the-art in reinforcement learning by developing new algorithms that seek to alleviate some of the difficulties most reinforcement learning algorithms have in dealing with realistic situations, such as partial observability and continuous action/observation representations.

Interestingly, there exist two very distinct approaches to solving reinforcement learning problems. One is the more conventional, *ontogenetic* approach which trains an agent to act in an environment by updating its policy-defining parameters at every time step – lifetime learning. The *phylogenetic* approach, on the other hand, learns *across* lifetimes, and uses a general 'fitness' measure, acquired after each trial episode, to update its policies. Both phylogenetic and ontogenetic algorithms have their own specific advantages and disadvantages. Ontogenetic algorithms typically do not cope well with both partial observability and continuity, while most phylogenetic approaches (most notably neuroevolutionary methods) work well only in low-stochasticity environments and are rather ad-hoc in nature.

Another important common distinction between reinforcement learning algorithms pertains to their classification as being either *value-based* or, alternatively, an instantiation of *direct policy search*. Value-based methods aim to estimate or predict the value of world states and indirectly infer (near-)optimal policies from those values, while direct policy search amounts to changing policies directly without using value estimates. Since value-based methods are known to suffer from several severe limitations (hard to tune, divergence using function approximators such as neural networks, slow learning in low-complexity environments), the focus of this thesis is on studying the potential of direct policy search methods, extending them so as to be able to properly deal with partial observability and continuity.

In this thesis, I will present four new algorithms, two phylogenetic and two ontogenetic, to solve reinforcement problems in partially observable environments (note that all algorithms in this thesis, including benchmarks,

are available as part of the PyBrain library: `www.pybrain.org`). Instead of deriving the algorithms from either value function-based methods or from evolutionary algorithms, I derive all four algorithms from first principles. Two well-established parameter-update methods are used – gradient ascent and Expectation Maximization – in order to determine policies. In summary, I aim to demonstrate that, in contrast to what seems to be the case for conventional value-based methods, one can derive powerful reinforcement learning algorithms that are in fact simple, efficient and convergent.

1.1 Thesis Overview

The chapters in this thesis can largely be read independently of one another. Chapter 2 provides a perspective on and taxonomy of both reinforcement learning algorithms and reinforcement learning problems, specifying the varying problem characteristics that require different types of solution techniques. In addition to classifying the various types of reinforcement learning algorithms, Chapter 2 will make a case for the class of algorithms studied in this thesis, *direct policy search* methods.

In Chapter 3 I present a novel algorithm, Recurrent Policy Gradients (RPGs), which constitutes the first ontogenetic approach to solve the hard non-Markovian double pole balancing task. It uses a Monte Carlo-estimated gradient in policy space to update the parameters of its memory-capable policy representation, the Long Short-Term Memory recurrent neural network architecture (Hochreiter and Schmidhuber, 1997). RPGs also exhibit highly competitive performance on various other benchmarks that emphasize different aspects of reinforcement learning problem characteristics.

Black box ('phylogenetic') search methods applied to RL, especially neuroevolution, have met with a considerable amount of success solving rather impressive tasks with partial observability (Gomez and Miikkulainen, 1997; Stanley and Miikkulainen, 2002). This is especially striking when compared to conventional, temporal difference approaches such as those based on algorithm families such as Q-learning and actor-critic. Chapter 4 presents Natural Evolution Strategies (NES), a novel phylogenetic search method resembling evolutionary search. In contrast to evolutionary methods, however, this method uses a clean derivation to update its search distribution using an *evolution gradient*, and applies the concept of natural gradients to aid the efficiency of its policy search. NES is shown to exhibit competitive results on the standard benchmark functions. Moreover, its performance on the hard non-Markovian double pole balancing task constitutes one of the best results reported in the literature, as far as the author is aware.

Chapter 5 introduces Fitness Expectation Maximization (FEM), a phylogenetic method derived from Expectation Maximization and even simpler than NES.

Chapter 6 presents the episodic Logistic Reward-Weighted Regression (LRWR) algorithm, which, like FEM, is derived from Expectation Maximization. This is an ontogenetic method, however, which uses full episodic information to *self-model* its better behaviors. Reasonable results are shown on a batch of small discrete POMDP benchmark tasks.

Chapter 7 presents comparative results on the presented algorithms, and the concluding Chapter 8 discusses the advantages, disadvantages and future prospects of the new direct policy search algorithms that are introduced in this work. The thesis concludes with a discussion on the relative merits of value search methods versus direct search methods, and argues that hybrid solutions are likely to yield the most promising results.

Chapter 2

A Perspective on Reinforcement Learning

This chapter first presents a discussion and formal introduction to the reinforcement learning framework as used in this thesis, then proceeds to provide a taxonomy of reinforcement learning problems according to their respective properties and appropriate solution algorithms, and then discusses the possible advantages of the particular family of RL solutions investigated in this thesis, *direct policy search*.

2.1 The Reinforcement Learning Framework

As defined by Sutton and Barto, any algorithm that can solve a reinforcement learning *problem* is a reinforcement learning *algorithm* (Sutton and Barto, 1998). This section introduces the reinforcement learning framework used in this thesis and its corresponding notation in more formal terms. As usual, an agent is considered that interacts with its environment in discrete time. The environment has *state* g_t at every time step t. Transitions from state to state are governed by a probability distribution unknown to the agent but dependent upon, on the one hand, the current action a_t and all previous actions $a_{0:t-1} = \langle a_0, a_1, \ldots, a_{t-1} \rangle$ executed by the agent, and, on the other hand, the current and previous states $g_{0:t} = \langle g_0, g_1, \ldots, g_t \rangle$ of the underlying system. Let r_t be the reward assigned to the agent at time t, and let o_t be the corresponding observation produced by the environment. Both quantities are governed by fixed distributions, solely dependent on environment state g.

In the most general reinforcement learning setting, the environment is partially observable, which means that the state of the environment is not directly observable. Rather, an observation is produced at every time step which is associated with the state, but may be ambiguous and/or stochastic. The concept of partial observability typically requires the agent to memorize

its experiences from the previous time steps in order to produce optimal actions.

Often, the issue of partial observability versus full observability is framed in terms of MDPs (Markov Decision Processes) versus POMDPs (Partially Observable Markov Decision Processes). A Markov Decision Process is a formalism that describes the interactions of an agent with the world. It consists of a number of world states the agent can be in, a number of actions the agent can perform in combination with the state-transition probabilities for every action, and a reward-function. An MDP can be formally described as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, r \rangle$ where

- $\mathcal{S}$ is a set $\{g_1, g_2, \ldots, g_K\}$ of world states
- $\mathcal{A}$ is a set $\{a_1, a_2, \ldots, a_A\}$ of actions
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a probabilistic state-transition function. For every state, action, and possible successor state, $T(g, a, g')$ denotes the probability of ending in the successor state g' given the start state g and action a.
- $r : \mathcal{S} \rightarrow \mathbb{R}$ is a (possibly stochastic) local reward function. For every state, $r(g)$ produces a local reward for being in state g, according to a probability distribution over real values.

The MDP generates, at every time step $t = 0, 1, 2, \ldots$, a state g_t after execution of action a_{t-1} in state g_{t-1}. Additionally, the reward r_t is given at every time step, in accordance with the reward function $r(g_t)$. An important notion in MDP theory is the *Markov* property. It is assumed that, in the MDP, states are *Markovian*, i.e., consistent and stationary. When a state is Markov, the next state is only dependent on the current state of the world and the action taken from there. It does not depend on other historical information. In other words, the state gives a complete description of the agent's situation in the world. More formally,

$$p(g_{t+1} = g | a_t, g_t, a_{t-1}, g_{t-1}, \ldots, g_0) = p(g_{t+1} = g | a_t, g_t),$$

which means that the observable state g fully describes the state of the system. This implies that, in order to produce optimal actions, one need only look at the current state observed by the agent, and one can ignore the history of states and actions before that. Now that it is clear that the agent's actions in an MDP are dependent solely upon the current state, one can supplement the MDP with the definition of the agent's policy π:

- $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is the agent's policy, which is a (possibly stochastic) assignment of actions to states.

In an MDP – the fully observable case – the agent's observation simply *equals* the environmental state, that is, $o_t = g_t$. In contrast, a POMDP utilizes a complex observation function which may not be wholly informative. In a POMDP, an agent does not have direct access to the state. In order to predict the next-step observation and reward, one may have to take into account its entire preceding history of actions, observations and rewards. This means that a simple memoryless policy mapping observations to actions may no longer be sufficient, and POMDP solution algorithms will generally have to employ some form of agent *memory*.

One can define the *observed history*[1] h_t as the string or vector of observations and actions up to moment t since the beginning of the episode: $h_t = \langle o_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1}, o_t \rangle$. The complete history H of length T includes the unobserved states and is given by $H = \langle h_{T-1}, g_{0:T-1} \rangle$. For any time step t, one can define the *return* as

$$R_t = \sum_{k=t}^{T-1} r_k \gamma^{t-k}$$

where T denotes the length of the episode, and $0 < \gamma \leq 1$ represents the *temporal discount factor* used, that is, the extent to which future rewards are exponentially discounted in comparison to the current reward. A discount factor γ close to 0 would place emphasis on immediate rewards only, while a γ closer to 1 would attribute more significance to later rewards in the reinforcement learner's perspective. Using the extreme of setting $\gamma = 1$ would indicate valuing all rewards equally, irrespective of the time step they were received.

The expectation of this return R_t at time $t = 0$ is also a measure of the quality of the policy π and, thus, the objective of reinforcement learning is to determine a policy which is optimal with respect to the expected future discounted rewards or expected return

$$J = \mathbf{E}\left[R_0 | \pi\right] = \mathbf{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t \middle| \pi\right].$$

An optimal or near-optimal policy in a partially observable environment requires that action a_t is taken depending on the *entire* preceding observed history. However, in most cases, it is not *necessary* to store the whole sequence of events but only sufficient statistics $M(h_t)$ of the events which can be called the *memory* of the agent's past. Thus, a stochastic policy π can be defined as $\pi(a|h_t) = p(a|M(h_t); \theta)$ where θ denotes the parameters of policy π. This forms a probability distribution $\pi(h_t; \theta)$ over actions, from which actions a_t are drawn $a_t \sim \pi(h_t; \theta)$ (henceforth, the explicit dependence of π

[1] Note that such histories are also called 'path' or 'trajectory' or 'episode' or 'trial' in the literature.

on θ will be omitted for the purpose of brevity). Policy π can be implemented and represented in various ways, for example as a finite state automaton, or, as is the case throughout this thesis, as a recurrent neural network with weights vector θ and stochastically interpretable output neurons.

2.2 A Taxonomy of Reinforcement Learning

Now that the RL framework is introduced, a taxonomy of reinforcement learning problems and algorithms can be considered. Reinforcement learning problems come in many sizes and flavors, as do the algorithms for solving them. It is not clear which types of reinforcement learning algorithms are best suited to solve which kinds of reinforcement learning problems. In this section I present some dimensions along the axes of which problems and algorithms can be varied to help distinguish them from each other. Based on results and arguments in the literature, I present some conjectures as to what algorithms should work best for particular types of problems.

In the last few decades, a wide variety of algorithms have been used to successfully solve RL problems in different guises. Surprisingly, these algorithms are based on very different principles. This is possibly due to the fact that they have been proposed and are actively studied within separate academic communities (e.g. machine learning, computational intelligence and control theory). There has been limited communication between these research fields, leading to insufficient analysis of the differences and similarities between these algorithms.

At the same time, a large variety of RL problems has been defined and studied, varying from real-world continuous control problems to abstract discrete toy problems. It is well known that some RL algorithms work well for some problems where other algorithms fail, but in many cases it is not clear what algorithms perform best under what conditions. The principal dimensions along which RL problems can vary are listed below.

The purpose of this section is to discuss some distinctions between types of reinforcement learning problems and reinforcement learning methods, especially between, on the one hand *ontogenetic* and *phylogenetic* methods, and, on the other hand, between *direct policy search* and *value-based* methods; discuss relevant issues prevalent in reinforcement learning research, and to make some conjectures about what types of methods would work best on what types of problems.

2.2.1 RL Problem Dimensions

- **Discrete vs. Continuous.** The environment's state, action and observation spaces can each be discrete, continuous or mixed.

- **Size and Dimensionality.** The state, action and observation spaces

can be represented in a varying number of dimensions. E.g., the state can be represented by a single integer or by a visual scene. The size of discrete dimensions can vary from small (binary) to large.

- **State Space Structure.** There can be varying degrees of *structure* in the state space: many benchmarks assume a certain locality (i.e. state transitions reach only a neighborhood of states). That structure is not necessarily ergodic, which thus allows for 'catastrophic' actions after which the agent cannot return to parts of the state space (e.g. a tabletop robot might fall off the table).

- **Stochasticity.** The state *transitions* can have varying degrees of stochasticity. E.g., a robot's wheels might slip, so it does not know how far it will move when trying to move forward.

- **Observability.** The environment can be *fully observable*, where the underlying state is directly accessible to the agent, or *partially observable*, where the agent can only make indirect, potentially stochastic *observations* of the state. This can impose a memory requirement on the agent as the only way to reliably infer the current state. In addition, observations can have varying degrees of redundancy, which in turn can make learning harder.

- **Generalization.** The observation representation might have meaningful structure, encapsulating aspects of the transition model and enabling generalization to similar states.

- **Reward Regime.** Rewards can vary from a single signal at the end of an episode (e.g. when winning a game) to many informative intermediate rewards (which can correspond to subtasks).

- **Episodic vs Life-long.** In case the task can be reset to a (distribution of) start state(s), one usually speaks of *episodic* RL, otherwise of *life-long infinite horizon* RL. Often, a temporal discount factor γ is used in RL problems, in order to give more weight to earlier rewards rather than later rewards (a $\gamma < 1$ is typically necessary in order to make J well-defined).

2.3 Issues in Reinforcement Learning

As alluded to in the above sections, most reinforcement learning algorithms suffer from severe problems that hinder their acceptance as a usable tool in the development of decision making algorithms. This section will describe some of the main issues associated with reinforcement learning that need to be addressed in order to increase the viability of the reinforcement learning framework.

2.3.1 The Temporal Credit Assignment Problem

In order to learn a policy well, the agent has to adapt its actions so as to maximize future rewards. However, because of possibly delayed rewards, it is not always easy to determine which actions are responsible for which reinforcements that are obtained. Especially when a policy is not satisfactorily learned yet, it is not easy to infer which actions in what situations contributed most to what rewards. Not all actions contribute equally to rewards, and often rewards are obtained despite having executed some bad actions. This problem of credit inference is called the *temporal credit assignment problem*.

2.3.2 Partial Observability

Most reinforcement learning research is directed towards fully observable environments, that is, environments in which the agent has access to the full environmental state as if it were simply an observation. In this case, the observation effectively *equals* the environmental state. However, this is a highly unrealistic assumption in many real world domains. For example, a robot's sensors might only provide approximate distances in a few directions, which would not constitute enough information to deduce its exact location in a room. It is therefore necessary to stress the importance and frequent occurrence of *partial observability*, that is, where the agent does *not* observe the environment directly, but rather observes an incomplete, possibly stochastic and/or ambiguous representation of the current state. The fact that different states might map to the same observations, combined with the issue of stochasticity or noisiness in observations, necessitates the use of some form of *memory* by the agent. This memory would maintain an internal state, incorporating and integrating previous observations, which the agent's policy can then utilize for its decisions.

Many POMDP solution techniques, judging the full problem too hard, assume knowledge of an environment model describing state transition probabilities and known state-observation mappings. Assumptions such as these are often unrealistic. In this thesis, therefore, I will only consider model-free POMDP approaches, where reinforcement learning algorithms have to try to find an optimal policy without having a perfect a priori given world model. When assuming a model-free approach, the agent has to either learn some approximation of a world model online while acting in the world, *or* it has to find a sufficiently powerful memory representation scheme that can be used to map memory states to actions directly without knowing the precise world dynamics.

Unfortunately, problems abound when attempting to learn a memory-based model or memory-based policy. Several trainable architectures have been proposed to deal with memory in an RL setting, most prominently

among them hidden Markov models (e.g., Chrisman, 1992), decision trees (e.g., McCallum, 1995) and recurrent neural networks (e.g., Lin, 1993; Bakker, 2002a). However, due to the problem of *vanishing gradient* (Hochreiter et al., 2001), events that are more than 4 or 5 time steps apart in history can typically not be related in practice. One method purposely designed to overcome this limitation is Long Short-Term Memory (LSTM: Hochreiter and Schmidhuber, 1997; Gers et al., 2002), and this is indeed the memory architecture I will use in the research of this thesis to represent memory-capable policies. Nevertheless, it might be fair to say that one of the main reasons why POMDP research has underachieved in the last few years, is the fact that suitable machine-learnable architectures are mostly lacking.

2.3.3 Observation Representation

Many environments are not characterized by small enumerable discrete state spaces and/or observation spaces, but rather have high-dimensional continuous representations. E.g., consider the orientation of a robot arm where the various angles, speeds and positions together form the representation of the environmental state. In order to be able to deal with continuous observations, policy π needs to be able to map continuous observation vectors or sequences of continuous vectors (histories) to actions. This has to be accomplished using some form of machine-learned *function approximator*. This function approximator will have to be able to generalize across states or (continuous) observations. Artificial neural networks constitute one of the most frequently used function approximators (or, in case of partial observability, *recurrent* neural networks), and various successes (e.g., Tesauro, 1994; Bakker, 2002a) have been reported in the literature. Function approximator training is, however, difficult and requires great tuning skill. Worse, it can even prove unstable – fail to converge – in the case of value-function based reinforcement learning methods (Baird, 1995). Great care must therefore be taken in appropriately choosing and training a function approximator.

2.3.4 Value Functions vs Direct Policy Search

Most reinforcement learning solution techniques can be classified as *value-based* methods, as opposed to *direct policy search* methods. Value function methods associate environmental states or state-action pairs directly with values, that is, expected returns when following the current policy. After having constructed a value function, all an agent has to do is select the action which has the highest estimated value. Numerous methods have been developed to learn and approximate this value function, most noteworthy among them TD(λ)-learning (Sutton, 1988), Q-learning (Watkins, 1989), and SARSA (Rummery and Niranjan, 1994).

In TD(λ)-learning, the *value* $V^\pi(g)$ of a state g in this MDP with policy π is defined to be the expected future discounted reward (also called *return*) received from starting in state g and following policy π from there afterwards:

$$\begin{aligned} V^\pi(g) &= \mathbf{E}[r_t + \gamma r_{t+1} + \gamma r_{t+2}^2 + \cdots | g_t = g, \pi] \\ &= \mathbf{E}\Big[\sum_{i=t}^{T-1} \gamma^{i-t} r_i | g_t = g, \pi\Big] \end{aligned}$$

Likewise, for Q-learning and SARSA-learning every state-action pair (g, a) has an associated quality or *Q-value* $Q^\pi(g, a)$ defined as

$$\begin{aligned} Q^\pi(g,a) &= \mathbf{E}[r_t + \gamma r_{t+1} + \gamma r_{t+2}^2 + \cdots | g_t = g, a_t = a, \pi] \\ &= \mathbf{E}\Big[\sum_{i=t}^{T-1} \gamma^{i-t} r_i | g_t = g, a_t = a, \pi\Big] \end{aligned}$$

These types of values offer a clean and insightful methodology for learning and analyzing policies in a reinforcement learning framework. However, one significant drawback of value-based methods pertains to its difficulty in constructing function approximators that do not diverge while learning the value function. The fact that even linear function approximators have been shown to diverge on especially constructed environments (Baird, 1995) does not inspire confidence in the usage of value-based methods in general, even though recent work (Sutton et al., 2009) addresses this issue somewhat satisfactorily. Sutton's work suggests that a clean solution to the issue of unstable function approximators in the fully observable case is possible. This might render value function methods viable for some real world tasks, but still does not address the divergence problem for partially observable settings. Nevertheless, in spite of these difficulties, carefully tuned neural networks used as value function approximators have been shown to perform well on a number of tasks.

For partially observable domains, using value functions is equally precarious. Even on simple partially observable environments, value methods can diverge catastrophically (Singh et al., 1994), as such significantly exacerbating the problems that value function methods already have using function approximators in the fully observable case. In fact, one might be tempted to say that whenever value function methods work well, this is *either* due to careful tuning, *or* luck, *or* prodigious hacking skills.

Direct policy search constitutes a wholly different paradigm. In direct policy search techniques, the algorithm searches directly in the space of policy representations to find a good policy, without estimating values for particular states or state-action pairs. Constructing good policies is often easier

than constructing correct value functions, because small and simple policy representations can work well in complex environments. This is especially true in partially observable environments, where the state space structure is actually unknown, which makes learning a value function particularly difficult. For POMDPs, bypassing the need to learn an approximately correct (memory-based) environmental model that can be used to represent values might be particularly helpful. A direct search in policy space appears to be more promising in numerous cases.

2.3.5 Action Representation

Some environments have the additional complication of having a continuous action structure. This might involve high-dimensional fine-tuned commands, such as muscle control in a robot arm or steering commands for car racing tasks. Unfortunately, it is as of yet very difficult to represent continuous action structure using most value function methods. This is so because value function methods typically have to be able to *choose* the action with maximum expected value, and since in continuous spaces there are in effect an infinite amount of possible actions, one has to perform a function optimization operation on the function approximator that is used as value function. In fact, recent techniques using evolutionary computation have done just that with reasonable results (personal communication, 2009), but nevertheless, dealing with continuous high-dimensional action spaces remains rather cumbersome for most reinforcement learning approaches.

2.4 A Taxonomy of RL Methods

A large number of algorithms have been devised to solve reinforcement learning problems. These can be broadly divided into *ontogenetic* algorithms and *phylogenetic* algorithms. In biology, *ontogeny* refers to the development of an organism *during* its lifetime, while *phylogeny* refers to the relatedness between groups of individuals, hence *across* multiple lifetimes. Phylogenetic algorithms (neuroevolution, for example) are those that only use the environment's fitness function f to update its policy-defining parameters θ. This fitness function is typically some measure of rewards accrued during one or more episodes, and may be completely unknown to the algorithm. Phylogenetic methods notably do *not* keep track of the particular states visited each episode, and typically (but not always) retain a 'population' of several policies. Since phylogenetic methods do *not* make use of the particular information obtained during episodes, these methods are often referred to as 'black box' methods. In contrast, ontogenetic algorithms (Q-learning, for example) can take into account the full information on which states/observations were visited and which states yielded which rewards,

```
PHYLO.INITIALIZE()
repeat
  θ ← PHYLO.GENERATE_POLICY()
  reset environment
  t ← 0
  fitness ← 0
  repeat
    observe o_t
    draw action a_t ∼ π(h_t|θ)
    execute action a_t
    obtain reward r_t
    t ← t + 1
  until episode's last time step
  fitness ← accumulate_fitness(r_0 ... r_{t−1})
  PHYLO.REPORT_FITNESS(fitness)
until stopping criterion met
```

Algorithm 2.1: The phylogenetic reinforcement learning framework. The 'PHYLO' object abstractly represents a phylogenetic reinforcement algorithm. As can be seen in this pseudocode, the phylogenetic reinforcement framework only allows for fitness values to be provided to the RL algorithm. Updates of the policy are only made after an episode has finished, based on the resulting fitness. Note that the 'accumulate_fitness' function must be defined, e.g., it could simply be calculating the future discounted reward as is usual in ontogenetic RL, or it could be any other arbitrary measure of policy quality. Evolutionary algorithms for control constitute a popular type of phylogenetic RL algorithm.

and typically update a single policy, instead of operating on a population of policies.

Phylogenetic approaches

Phylogenetic methods treat the RL problem as a black box optimization problem, optimizing a policy for maximal accumulated reward over one or several episodes. In principle, any black box optimization method could be used, including local search methods such as hill climbing and simulated annealing. More commonly, however, evolutionary algorithms like evolution strategies and genetic algorithms are used for phylogenetic RL. These algorithms work by maintaining a population of policies. Each policy is assigned a fitness based on the accumulated reward over one or more episodes. Less fit policies are then removed from the population and replaced with new policies, constructed through combining and varying more fit policies. Some algorithms from swarm intelligence, e.g. particle swarm optimization, work according to similar principles and can also be used for phylogenetic RL. Since phylogenetic approaches do not estimate value functions, all phyloge-

```
θ ← ONTO.INITIALIZE()
repeat
  reset environment
  t ← 0
  repeat
    observe o_t
    draw action a_t ∼ π(h_t|θ)
    execute action a_t
    obtain reward r_t
    θ ← ONTO.UPDATE_POLICY(o_t,a_t,r_t)
    t ← t + 1
  until episode's last time step
  θ ← ONTO.UPDATE_POLICY_LAST_STEP()
until stopping criterion met
```

Algorithm 2.2: The ontogenetic reinforcement learning framework. This pseudocode abstractly depicts the framework of ontogenetic reinforcement learning. The 'ONTO' object represents the particular ontogenetic RL algorithm instantiated. From the code, it is clear that ontogenetic algorithms can potentially change the policy-defining parameters θ after every time step. Also, it should be noted that the 'ONTO' object is given all information on which actions, observations and rewards were processed at each time step, this in stark contrast to phylogenetic methods which are provided with a fitness value only. Popular examples of ontogenetic reinforcement learning include Q-learning and policy gradient methods. For Q-learning, for example, θ would be comprised of all Q-values and the exploration rate.

netic techniques are *direct policy search* methods. Algorithm 2.1 presents the phylogenetic reinforcement learning framework in pseudocode. In this thesis, I will present two novel phylogenetic direct policy search methods, namely, Natural Evolution Strategies (Chapter 4) and Fitness Expectation Maximization (Chapter 5).

Ontogenetic approaches

A formal depiction of the ontogenetic reinforcement learning framework is provided in Algorithm 2.2. As is obvious from the pseudocode, ontogenetic algorithms, in contrast to phylogenetic methods, can update the policy at every time step during execution, and the algorithm has full access to which observations, actions and rewards have been encountered.

The value function algorithms described in the discipline-defining book *Reinforcement Learning* (Sutton and Barto, 1998), (such as Q-learning and SARSA), are all ontogenetic. At each time step, the action with the highest associated value is chosen (according to the value function). Based on rewards received from the environment, the value function is updated *each*

time step using temporal difference methods.

Not all ontogenetic methods are value function-based, though. A different family of ontogenetic RL algorithms are those based on policy gradient ascent, which directly learn a policy without estimating state-dependent value functions. In the policy gradient framework (Williams, 1992; Baxter et al., 2001; Peters and Schaal, 2008a), policies are stochastic and their parameters are updated directly using a gradient in the direction of better expected return. These methods, when applied to function approximators like neural networks (Wierstra et al., 2007), constitute an alternative to value-based methods that is more similar to phylogenetic methods in that they typically do not need to use value functions and are able to represent policies directly, and also since they implicitly represent a distribution of policies (instead of a single greedy policy) because of their inherently stochastic actions. In this thesis, I will present two novel ontogenetic direct policy search methods that can deal with function approximators in partially observable environments. These algorithms, Recurrent Policy Gradients (Wierstra et al., 2009, 2007) and the episodic Logistic Reward Weighted Regression algorithm (Wierstra et al., 2008b), will be discussed in Chapters 3 and 6, respectively.

2.4.1 Which algorithms work best? For which problems?

There does not seem to be any particular RL algorithm that performs better than other algorithms on all problems, at least in practice: certain universal RL methods can be proven to be optimal, however, these are either incomputable (Hutter, 2005) or currently suffer from insurmountable computation overhead that prevent their practical use (Schmidhuber, 2006). Different algorithms have different strengths and weaknesses, many of them currently unknown. However, the literature contains a number of theoretical arguments and empirical results suggesting the superiority of some families of algorithms over others. Table 2.1 provides a schematic taxonomy of representative reinforcement learning algorithms in their corresponding subcategories of value-based ontogenetic methods, direct policy search ontogenetic methods, and phylogenetic methods.

Phylogenetic methods do not have satisfying convergence guarantees even in fully observable domains, whereas ontogenetic methods do (e.g., Bertsekas and Tsitsiklis, 1996). Phylogenetic methods also suffer more severely from the *credit assignment problem*, especially in stochastic or large domains, since only one single fitness measure is attributed to an entire episode roll-out, whereas ontogenetic methods can utilize all information encountered during a trajectory, including observations and rewards received at specific time steps. For phylogenetic approaches, many reruns of the same

	Value-based	**Direct policy search**
Ontogenetic	actor-critic Q-learning SARSA	recurrent policy gradients (RPG) logistic reward-weighted regression (LRWR) finite policy graphs
Phylogenetic		evolutionary methods: – evolution strategies (ES) – covariance matrix adaptation (CMA-ES) – enforced sub-populations (ESP) – cooperative synapse evolution (CoSyNE) hillclimbing particle swarm optimization (PSO) fitness expectation maximization (FEM)

Table 2.1: **A taxonomy of some conceptually important reinforcement learning algorithms.** The two broad categories are phylogenetic and ontogenetic. Whereas all (currently known) phylogenetic methods are said to be *direct policy search* methods, ontogenetic methods can be divided into value-based and direct policy search methods. The methods presented here include covariance matrix adaptation (CMA-ES) (Hansen and Ostermeier, 2001), particle swarm optimization (Kennedy and Eberhart, 2001), ESP (Gomez and Miikkulainen, 1997), CoSyNE (Gomez et al., 2006), and finite policy graphs (Meuleau et al., 1999). The four novel algorithms presented in this thesis, Natural Evolution Strategies (NES, Chapter 4), Recurrent Policy Gradients (RPG, Chapter 3), Fitness Expectation Maximization (FEM, Chapter 5) and Logistic Reward-Weighted Regression (LRWR, Chapter 6) span both the phylogenetic and the ontogenetic framework, but all four methods fall within the category of direct policy search.

genome may be required to reliably estimate the true fitness of the genome in case there is any stochasticity or noise inherent in the environment. Ontogenetic methods do not suffer from this limitation as they can more reliably (at least in the fully observable case) attribute rewards and value to exactly those states that have actually been visited during an episode. One example would be the case where a robot arm must be trained to pick up an object from a tabletop, irrespective of where it is located. A phylogenetic method's fitness measure can only be approximated by repeating many episodes using the same policy with the object in different locations, which may result in an unreasonable amount of real world experience required for just one piece of information – one single fitness value. On the other hand, an ontogenetic method is by definition allowed to use all history information contained in one episode (observations, actions, rewards), and to use that information to update its policy after every single trial and even after every time step. It can attribute failure or success to only those situations that have actually been visited instead of having to approximate one broad, high-variance fitness measure. This, in principle, should be significantly more efficient.

Another limitation of phylogenetic approaches is that the difficulty grows with the size of the policy parameter space. For example, typical phylogenetic algorithms (such as evolution strategies) typically cannot evolve more than a few hundred parameters, whereas ontogenetic methods scale well with the number of states (Bertsekas and Tsitsiklis, 1996). In summary, there are good arguments that in fully observable discrete state spaces that are either large or that have inherent state transition stochasticity, ontogenetic approaches would outperform phylogenetic ones.

As sketched above, many value-based ontogenetic approaches diverge and/or are hard to train using function approximators, especially when using neural networks (Baird, 1995). Recent advances in function approximation such as neural fitted Q iteration (Riedmiller, 2005) and Sutton's gradient TD (Sutton et al., 2009) have somewhat lessened this problem. Nevertheless, reasonable performance for most ontogenetic algorithms that operate in partially observable environments remains elusive in practice (Lucas and Togelius, 2007). Phylogenetic approaches on the other hand, among which neuroevolution features most prominently, have been fairly successful and robust in practice on various domains (Gomez et al., 2008; Stanley, 2004; Igel, 2003; Gomez et al., 2008; De Nardi et al., 2006). They generally do not suffer as much from function approximator fine-tuning problems as value-based ontogenetic approaches do (Whiteson et al., 2007), and do not have the same divergence properties, especially in POMDP settings. One reason, one might presume, is that it is easier to find a good policy than an approximately correct value function. Note that even approximated value functions that have low error may still yield very inappropriate policies, and many partially observable environments have simple optimal policies but complex value functions.

Solving POMDPs using value-based ontogenetic methods is theoretically unsound, exceedingly hard and requires prodigious fine-tuning skills, especially when using memory-capable function approximators such as recurrent neural networks. There is one class of ontogenetic methods, however, which I will demonstrate in this thesis – Recurrent Policy Gradients – that can easily and naturally deal with both partial observability and continuous environments without divergence. This method constitutes an ontogenetic *direct policy search* approach.

2.4.2 Some Conjectures on Reinforcement Learning

Considering the above arguments, I conjecture that phylogenetic direct search methods such as neuroevolution generally outperform ontogenetic methods on problems with continuous state spaces and/or partial observability, especially when the number of trainable policy parameters can be kept comparatively small. On the other hand, some ontogenetic methods with value functions, such as Q-learning and SARSA, are unbeatable on problems

with discrete state spaces and full observability. For environments with significant stochasticity, ontogenetic techniques will almost certainly prove to have a competitive edge over phylogenetic methods, since the stochasticity requires phylogenetic algorithms to rerun one policy many times to gain a reliable fitness estimate, while ontogenetic methods can do proper temporal credit assignment onto the states and actions actually visited. In larger parameter spaces, the performance of phylogenetic methods may start do degrade faster than that of ontogenetic methods. Since both phylogenetic methods and value-based ontogenetic methods are likely to meet with rather serious problems as the complexity and size of the domains scale up, I conjecture that direct policy search ontogenetic methods may constitute one of the more promising research directions in the long run.

2.4.3 Reinforcement Learning Benchmarks

Reinforcement learning benchmarks have obvious relevance to a taxonomy of reinforcement learning algorithms. Many attempts have been made to compare different RL methods on benchmark problems. These include open competitions (e.g., Loiacono et al., 2008), and efforts to standardize simple benchmarks through source code sharing. However, each of these efforts typically only compare a few algorithms on a single problem, leading to contradictory results regarding the merits of different RL methods.

An approach to more exhaustive reliable characterization of methods would be to create benchmarks that could be varied along as many as possible of the problem dimensions listed above. Through tuning benchmark parameters, one could then corroborate, falsify or qualify hypotheses about relative method performance. In this thesis I will use a set of various benchmarks that embody several problem dimensions that have been discussed so far – discrete vs continuous observations, discrete vs continuous actions, state transition stochasticity, problem dimensionality, state space structure, partial observability – in order to demonstrate the strengths and weaknesses of new algorithms on different types of environments.

2.5 Direct Policy Search

The aim of this research is to advance the state-of-the-art in reinforcement learning algorithms for partially observable, continuous environments, as that constitutes the most generally applicable, and arguably most important case. Having discussed the various classes of reinforcement learning algorithms in existence today, one can guess with some confidence that the most promising research direction would lead one to investigate direct policy search techniques as applied to POMDPs, whether they be phylogenetic or ontogenetic.

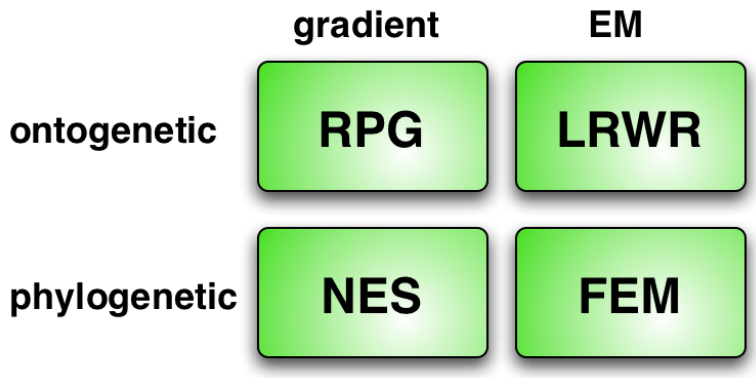

Figure 2.1: **The categorical properties of NES, FEM, RPG and LRWR.** This figure depicts a schematic algorithm categorization of Natural Evolution Strategies (NES, Chapter 4), Recurrent Policy Gradients (RPG, Chapter 3), Fitness Expectation Maximization (FEM, Chapter 5) and Logistic Reward-Weighted Regression (LRWR, Chapter 6). Of the four new RL algorithms, two are phylogenetic, two ontogenetic. Two methods are gradient ascent based, two are based on Expectation Maximization (EM).

In this thesis, I present four new RL algorithms, two phylogenetic, two ontogenetic. Two methods are gradient ascent based, two are based on Expectation Maximization. Figure 2.1 presents a schematic depiction of the categorical properties of these four algorithms.

The first algorithm, Recurrent Policy Gradients, is discussed in Chapter 3 and represents a policy gradient method applied to LSTM recurrent neural networks. This ontogenetic search method backpropagates history-dependent eligibilities through time, as such obtaining gradient estimates for updating the policy.

The second algorithm, Natural Evolution Strategies, is presented in Chapter 4 and takes on reinforcement learning from a phylogenetic perspective, and in effect constitutes an alternative to evolutionary methods derived from first principles (i.e. significantly less heuristic). As a gradient-based method, it uses the *natural gradient* (Amari, 1998) rather than the plain 'vanilla' gradient, and so avoids some of the adverse problems that regular gradient methods usually experience.

Chapter 5 discusses the third algorithm, Fitness Expectation Maximization, which is an instantiation of the well-known Expectation Maximization framework for parameter optimization. As it is derived from Expectation Maximization, this phylogenetic method represents, as is the case for Natural Evolution Strategies, a principled alternative to regular evolutionary methods.

The fourth, the episodic Logistic Reward-Weighted Regression algorithm,

is an ontogenetic technique that instantiates a form of Expectation Maximization, using self-modeling to improve upon its own experiences. Chapter 6 discusses the algorithm and the experimental results.

As can be gathered from the comparative results presented in Chapter 7, these four new techniques have different strengths and weaknesses in different types of environments. Nevertheless, I hope to demonstrate with the work and good results in this thesis that the direct policy search framework constitutes a viable alternative to value-based methods in partially observable domains.

Chapter 3

Recurrent Policy Gradients

Reinforcement learning in partially observable environments is a challenge as it requires policies with an internal state. Traditional approaches suffer significantly from this shortcoming and usually make strong assumptions on the problem domain such as perfect system models, state-estimators and a Markovian hidden system. Recurrent neural networks (RNNs) offer a natural framework for dealing with policy learning using hidden state and require only few limiting assumptions. As they can be trained well using gradient descent, they are suited for policy gradient approaches.

In this chapter, I follow (Wierstra et al., 2009, 2007) and present a new policy gradient method, the *Recurrent Policy Gradients* (RPGs) algorithm which constitutes a model-free ontogenetic reinforcement learning method. It is aimed at training memory-based stochastic policies on problems which require long-term memories of past observations. The approach involves approximating a policy gradient for a recurrent neural network by back-propagating return-weighted characteristic eligibilities through time. Using a "Long Short-Term Memory" recurrent neural network architecture, RPGs are able to outperform previous ontogenetic RL methods on three important benchmark tasks with different properties regarding the extent of stochasticity, continuity and partial observability. Furthermore, I introduce history-dependent baselines and show that they help reducing estimation variance significantly, thus enabling the approach to tackle more challenging, highly stochastic environments.

3.1 Introduction

For partially observable and non-Markovian problems, an optimal policy will require a policy representation with an internal memory. Among all function approximators with internal state, recurrent neural networks appear to be the method of choice and can make a big difference in reinforcement learning problems. However, only few reinforcement learning methods are theoreti-

cally sound when applied in conjunction with such function approximation (see Chapter 2), and catastrophic divergence of traditional methods can be shown in this context (Singh et al., 1994).

Policy gradient (PG) algorithms constitute an exception, as these direct policy search methods allow for learning policies even with noisy state information (Baxter et al., 2001), work in combination with function approximation (Sutton et al., 2001; Bhatnagar et al., 2007), are compatible with policies that have internal memory (Aberdeen, 2003), can naturally deal with continuous actions (Peters and Schaal, 2008a; Kohl and Stone, 2004), and are guaranteed to converge at least to a local minimum. Furthermore, most successful algorithms for solving real world reinforcement learning tasks are in fact applications of PG methods. See, for example, (Benbrahim and Franklin, 1997; Moody and Saffell, 2001; Prokhorov, 2007; Baxter et al., 2001; Peters and Schaal, 2006, 2008b) for an overview. Provided the choice of policy representation is powerful enough, PGs can tackle quite complex RL problems.

At this point, policy gradient-based reinforcement learning exhibits two major drawbacks from the perspective of recurrent neural networks, i.e., (i) the lack of scalability of policy gradient methods in the number of parameters, and (ii) the small number of algorithms that were developed specifically for recurrent neural network policies with large-scale memory. Most PG approaches have only been used to train policy representations with maximally a dozen parameters, while RNNs can have hundreds or thousands. Surprisingly, the obvious combination with standard backpropagation techniques has not been extensively investigated (a notable exception being the SRV algorithm (Gullapalli, 1990, 1992), which was, however, solely applied to feedforward networks). In this chapter, I address this shortcoming, and show how PGs can be naturally combined with backpropagation, and *backpropagation through time* (BPTT: Werbos, 1990) in particular, to form a powerful RL algorithm capable of training complex neural networks with large numbers of parameters.

Work on policy gradient methods with memory has been scarce so far, largely limited to finite state controllers. Strikingly, memory models based on finite state controllers perform less than satisfactorily, even on quite simple benchmarks. Illustratively, single pole balancing without velocity information (see section 3.4.3) cannot be learned beyond 1000 time steps (Aberdeen, 2003; Meuleau et al., 1999), whereas evolutionary methods and the algorithm presented in this chapter manage to balance the pole $100,000+$ steps. One could conjecture that, for finite state controllers, the reason is that a *stochastic* memory state model must be learned in conjunction with a policy, which is prohibitively expensive. In this chapter, I extend policy gradient methods to more sophisticated policy representations capable of representing memory using an RNN architecture called Long Short-Term Memory (LSTM) for representing the policy (Hochreiter and Schmidhuber,

1997). A new reinforcement learning algorithm is developed that is aimed specifically at RNNs that can effectively learn memory-based policies for deep memory POMDPs. This algorithm, the *Recurrent* Policy Gradient (RPG) algorithm, backpropagates the estimated return-weighted *eligibilities* backwards through time using recurrent connections in the RNN. As a result, policy updates can become a function of any event in the history. The presented method is shown to outperform other RL methods on three important RL benchmark tasks with different properties: first, continuous control in a non-Markovian double pole balancing environment; second, discrete control on the deep memory T-maze task (Bakker, 2002a), which was designed to test an RL algorithm's ability to deal with extremely long term dependencies; and third, the still-unsolved (up to human-level performance) stochastic 89-state maze task. Moreover, promising results in a complex car driving simulation are shown, which is challenging for humans. Here, real-time improvement of the policy is demonstrated, something which has been largely unachieved in reinforcement learning for such complex tasks.

The chapter is organized as follows. The next section briefly reviews LSTM's architecture. The subsequent sections introduce the derivation of Recurrent Policy Gradient algorithm, and present experimental results using RPGs with memory. The chapter concludes with a discussion on possible future extensions of the method.

3.2 LSTM as Policy Representation

Recurrent neural networks are designed to deal with issues of time, such as approximating time series. A crucial feature of this class of architectures is that they are capable of relating events in a sequence, in principle even if placed arbitrarily far apart. A typical RNN π maintains an *internal state* $M(h_t)$ (or *memory*) which it uses to pass on (compressed) history information to the next moment by using recurrent connections. At every time step, the RNN takes an input vector o_t and produces an output vector $\pi(M(h_t))$ from its internal state $M(h_t)$. Since the internal state $M(h_t)$ of any step is a function m of the previous state and the current input signal $M(h_t) = m(o_t, M(h_{t-1}); \theta)$ where θ represents the RNN's parameters/weights, it can take into account the entire history of past observations by using its recurrent connections for recalling events. Not only can RNNs represent memory, they can, in theory, be used to model any dynamic system (Siegelmann and Sontag, 1991). Like conventional neural networks, they can be trained using a special variant of backpropagation, backpropagation through time (BPTT: Werbos, 1990; Williams and Zipser, 1989).

Usually, BPTT is employed to find the gradient $\nabla_\theta E$ in parameters θ (that define m and π) for minimizing some error measure E, e.g. summed squared error. This is done by first executing a forward pass through the

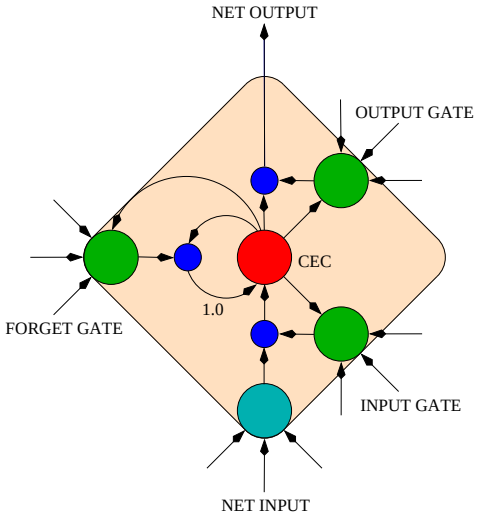

Figure 3.1: **The Long Short-Term Memory cell**. The figure shows an LSTM cell with a net input, a Constant Error Carousel (CEC), an input gate, a forget gate and an output gate. The cell has an internal state CEC and a forget gate that determines how much the CEC is attenuated at each time step. The input gate controls access to the state by the external inputs and the outputs of other cells, and the output gate determines how much and when the cell fires. Figure designed by J. Schmidhuber ©.

RNN from the beginning of the sequence all the way to the end of the sequence, at every time step unfolding the RNN, reusing parameters θ for the recurrent connections, producing outputs, and computing the errors δ_t for all time steps. Then a (reverse) backwards pass is performed, computing the gradient backwards through time by backpropagating the errors. Usually, this is done in a *supervised* fashion, but here this technique is applied to a reinforcement learning setting.

RNNs have attracted some attention in the past decade because of their simplicity and potential power. However, though powerful in theory, they turn out to be quite limited in practice due to their inability to capture long-term time dependencies – they suffer from the problem of *vanishing gradient* (Hochreiter et al., 2001), the fact that the gradient signal vanishes as the error signal is propagated back through time. Because of this, events more than 10 time steps apart can typically not be related.

One method purposely designed to avoid this problem is Long Short-Term Memory (LSTM: Hochreiter and Schmidhuber, 1997; Gers et al., 2002), which constitutes a special RNN architecture capable of capturing long term time dependencies. The defining feature of this architecture is that it consists of a number of differentiable *memory cells*, which can be

used to store activations arbitrarily long. Access to the internal state of the memory cell (the Constant Error Carousel or CEC) is *gated* by gating units that learn to open or close depending on the context. Three types of (sigmoidal) gates are present: *input gates* that determine the input to the memory cell, *forget gates* that control how much of the CEC's value is transferred to the next time step, and *output gates* which regulate the output of the memory cell by gating the cell's output. See Figure 3.1 for a depiction of LSTM's structure.

The gates also receive inputs from other neurons, and a function over their inputs determines whether they open or close. The amount each gate g_i of memory cell i is open or closed at time t is calculated by

$$g_i^{in}(t) = \varsigma\left[\sum_j w_{ij}^{in} y_j(t-1) + \sum_k w_{ik}^{in} x_k(t)\right],$$

$$g_i^{forget}(t) = \varsigma\left[\sum_j w_{ij}^{forget} y_j(t-1) + \sum_k w_{ik}^{forget} x_k(t)\right],$$

$$g_i^{out}(t) = \varsigma\left[\sum_j w_{ij}^{out} y_j(t) + \sum_k w_{ik}^{out} x_k(t)\right],$$

where $w_{ij}^{\{in,forget,out\}}$ is the weight from the output y_j of cell j to gate i, $w_{ik}^{\{in,forget,out\}}$ is the weight from external input x_k to the gate i, and ς is the familiar logistic sigmoid function $\varsigma(x) = 1/(1+\exp(-x))$. The external inputs to the cell are added up in $net_i(t)$:

$$net_i(t) = \sum_j w_{ij}^{cell} y_j(t-1) + \sum_k w_{ik}^{cell} x_k(t).$$

The internal state of cell i is calculated by copying the state from the previous time step modulated by the forget gate's output, and allowing additional inputs to enter the state by multiplying the input gate to the cell's net input:

$$s_i(t) = net_i(t) g_i^{in}(t) + g_i^{forget}(t) s_i(t-1).$$

The output gates modulate the cell outputs y_i:

$$y_i(t) = g_i^{out}(t) s_i(t).$$

The above formulas describe the forward pass of the LSTM architecture in the BPTT algorithm. The entire architecture as is used for BPTT is more fully described and discussed in detail in (Graves and Schmidhuber, 2005).

LSTM networks have been shown to outperform, in supervised learning settings, other RNNs on numerous time series requiring the use of deep

memory (Schmidhuber, 2004). Therefore, they seem well-suited for usage in policy gradient algorithms for complex tasks that require deep memory. Whereas LSTM networks have usually been used to *predict*, here it is used to *control* an agent directly, to represent a controller's policy receiving observations and producing action probabilities at every time step. Note that in this thesis, for comparison purposes, all four presented algorithms use the LSTM network architecture as its controller, and all approaches operate on LSTM's weights to find better policies.

3.3 Recurrent Policy Gradients

In this section, I first formally derive the Recurrent Policy Gradient framework. Subsequently, history-dependent baselines are introduced, and the section is concluded with a description of the Recurrent Policy Gradient algorithm.

3.3.1 Derivation of Recurrent Policy Gradients

The type of RL algorithm employed in this chapter falls in the class of policy gradient algorithms, which, unlike many other (notably temporal difference) methods, update the agent's policy-defining parameters θ directly by estimating a gradient in the direction of higher (average or discounted) reward. This makes this algorithm a direct policy search algorithm.

Now, let $R(H)$ be some measure of the total reward accrued during a history. $R(H)$ could be the average of the rewards for the average reward case, or the discounted sum for the discounted case. Let $p(H|\theta)$ denote the probability of a history given policy-defining weights θ. The quantity the algorithm should be optimizing is

$$J = \int_H p(H|\theta)R(H)dH.$$

This, in essence, indicates the expected reward over all possible histories, weighted by their probabilities under policy π. In order to be able to apply gradient ascent to find a better policy, one has to find the gradient $\nabla_\theta J$, which can then be used to incrementally update parameters θ of policy π in small steps. Since it is clear that rewards $R(H)$ for a given history H do *not* depend on the policy parameters θ (that is, $\nabla_\theta R(H) = 0$), one can write

$$\begin{aligned} \nabla_\theta J &= \nabla_\theta \int_H p(H|\theta)R(H)dH \\ &= \int_H \nabla_\theta p(H|\theta)R(H)dH. \end{aligned}$$

Now, using the "likelihood-ratio trick" one finds

$$\begin{aligned}\nabla_\theta J &= \int \nabla_\theta p(H) R(H) dH \\ &= \int \frac{p(H)}{p(H)} \nabla_\theta p(H) R(H) dH \\ &= \int p(H) \nabla_\theta \log p(H) R(H) dH.\end{aligned}$$

Taking the sample average as Monte Carlo (MC) approximation of this expectation by taking N trial histories one gets

$$\begin{aligned}\nabla_\theta J &= \mathbf{E}_H\Big[\nabla_\theta \log p(H) R(H)\Big] \\ &\approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \log p(H^n) R(H^n).\end{aligned}$$

which is a fast approximation of the policy gradient for the current policy.

Probabilities of histories $p(H)$ are dependent on an unknown initial state distribution, on unknown observation probabilities per state, on unknown state transition function $p(g_{t+1}|a_{0:t}, g_{0:t})$, and on *known* action probabilities given the agent's policy. The environment's properties are considered to be unknown, but at least the agent knows its own action probabilities, so the log derivative for agent parameters θ in $\nabla_\theta \log p(h)$ can be acquired by first realizing that the probability of a particular history is the product of all actions and observations given subhistories:

$$p(H_T) = p(\langle o_0, g_0 \rangle) \prod_{t=1}^{T-1} p(\langle o_t, g_t \rangle | h_{t-1}, a_{t-1}, g_{0:t}) \pi(a_{t-1}|h_{t-1})$$

Taking the log-derivative results into transforming this large product into a sum

$$\log p(H_T) = (\text{const}) + \sum_{t=0}^{T-1} \log \pi(a_t|h_t) :$$

where most parts are not affected by θ, i.e., are constant. Thus, when taking the derivative of this term, the following is obtained:

$$\nabla_\theta \log p(H_T) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|h_t).$$

Substituting this term into the MC approximation results in a gradient estimator which only requires observed variables. However, making use of the fact that future actions do not depend on past rewards, one can omit these

terms from the gradient estimate. Thus, an unbiased gradient estimator is given by

$$\nabla_\theta J \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T(H^n)-1}\nabla_\theta \log \pi(a_t|h_t^n)R_t^n$$

which yields the desired gradient estimator which only has observable variables.

3.3.2 History-dependent Baselines

Nevertheless, an important problem with this Monte Carlo approach is the often high variance in the gradient estimate. For example, if $R(h) = 1$ for all h, the variance can be given by $\sigma_T^2 = \mathbf{E}[\sum(\nabla_\theta \log \pi(a_t|h_t^n))^2]$ which grows linearly with T. One way to tackle such problems and reduce this variance is to include a constant *baseline* b – first introduced in Williams (1992) – into the gradient estimate

$$\nabla_\theta J \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T(H^n)-1}\nabla_\theta \log \pi(a_t|h_t^n)(R_t^n - b).$$

Baseline b is typically taken to be the expected average return and subtracted from the actual return, such that the resulting quantity $(R_t - b)$ intuitively yields information on whether the return was *better or worse than expected*. Due to the likelihood-ratio trick

$$\begin{aligned}\int p(H)\nabla_\theta \log p(H) b dH &= \nabla_\theta \int p(H) b dH \\ &= \nabla_\theta 1 \\ &= 0,\end{aligned}$$

it can be guaranteed that

$$\mathbf{E}\left[\sum_{n=1}^{N}\nabla_\theta \log p(H_t^n) b\right] = 0,$$

and, thus, the baseline can only reduce the variance but not bias the gradient in any way.

Whereas previously a *constant* baseline was used, one can in fact extend the baseline concept to include subhistory-dependent function approximators $B(h_t)$ parameterized by w. The correctness of this approach can be realized by applying the same trick $\int_a \nabla_\theta \pi(a|h_t)B(h_t)da = 0$ for every possible subhistory h_t. Now the baseline $B(h_t)$ can be represented as an LSTM RNN receiving observations and actions as inputs, trained to predict future return given the current policy π. This construct closely resembles the concept of value functions in temporal difference methods. However, note

that one cannot use temporal difference methods for training the history-dependent baseline network, since such updates can be arbitrarily bad in partially observable environments (Singh et al., 1994). Instead, supervised training is applied using simply the actually experienced returns as targets for every time step. Using non-constant, history-dependent baselines, the algorithm now uses *two* separate RNNs: one policy π parameterized by θ, and one baseline network B parameterized by w. Using the extended baseline network, the gradient update for the policy now becomes

$$\nabla_\theta J \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|h_t^n;\theta)(R_t^n - B(h_t^n|w)).$$

3.3.3 The Recurrent Policy Gradients Algorithm

Typically, PG algorithms learn to map observations to action probabilities, i.e. they learn stochastic reactive policies. As noted before, this is clearly suboptimal for all but the simplest partial observability problems. One would like to equip the algorithm with adaptable memory, using LSTM to map histories or *memory states* to action probabilities. Unlike earlier methods, our method makes full use of the backpropagation technique while doing this: whereas most if not all published and experimentally tested PG methods (as far as the author is aware) estimate parameters θ individually, RPGs use *eligibility-backpropagation through time* (as opposed to standard error-backpropagation or BPTT (Werbos, 1990)) to update all parameters conjunctively, yielding solutions that better generalize over complex histories. Using this method, one can map *subhistories* (all observations and actions experienced in the episode so far) to actions instead of *observations* to actions.

In order to estimate the gradient for a history-based approach, one maps histories h_t to action probabilities by using LSTM's internal state representation. Backpropagating return-weighted eligibilities (Williams, 1992) affects the policy such that it makes histories that were better than other histories (in terms of reward) more likely by reinforcing the probabilities of taking similar actions for similar subhistories.

Recurrent Policy Gradients are architecturally nearly equal to supervised RNNs, however, the output neurons are interpreted as a probability distribution. It takes, at every time step during the forward pass of BPTT, as input observation o_t. Together with the recurrent connections, these produce outputs $\pi(h_t)$, representing the probability distribution on actions.

Only the output part of the neural network is interpreted stochastically. This allows us, during the backward pass, to only estimate the eligibilities of the output units at every time step. The gradient on the other parameters θ can be derived efficiently via eligibility backpropagation through time, treating output eligibilities as normal errors in an RNN trained with gradient

descent. Also, by having only stochastic output units, computing complicated gradients on stochastic internal (belief) states such as done in (Aberdeen, 2003; Meuleau et al., 1999) is unnecessary – eligibility backpropagation through time disambiguates relevant hidden state automatically, when possible.

3.4 Experiments

Here I present empirical results on four fundamentally different problem domains. The first task, double pole balancing with incomplete state information, is a *continuous* control task that has been a benchmark in the phylogenetic RL community for many years. In fact, RPGs constitute the very first ontogenetic RL task to solve this problem, as far as the author is aware. The second task, the T-maze, is a difficult discrete control task that requires remembering its initial observation until the end of the episode. On this task, RPGs outperformed the second-best method by more than an order of magnitude for longer corridors. The third task, the 89-state maze, is a highly stochastic POMDP maze task which has yet to be solved up to human level performance. On this task, RPGs outperform all other (model-free) algorithms.

Last, I show promising results on a complex car driving simulation (TORCS) which is challenging for humans. Here, real-time improvement of the policy is shown, something which has been largely unachieved in reinforcement learning for such complex tasks.

All experiments were carried out with 10-cell LSTM recurrent neural networks, with initial weight range between -0.01 and 0.01. The constant baseline estimator used was simply a moving average of the returns from the last 100 episodes, except for the 89-state maze task, where an additional LSTM network was used to estimate a history-dependent baseline.

3.4.1 Non-Markovian Double Pole Balancing

Pole balancing is a task which involves trying to balance a pole hinged on a cart that moves on a finite track. The single control consists of the force F applied to the cart (in Newtons), and observations include the cart's position x, the pole's angle β and velocities $\dot{x}$ and $\dot{\beta}$. It provides a perfect testbed for algorithms focussing on learning fine control in continuous state and action spaces. The challenging version of this task has no velocity information $\dot{x}$ and $\dot{\beta}$ such that the problem becomes non-Markovian. Additionally, a second pole can be included on the same cart, of length 1/10th of the original one. Both poles must then be balanced simultaneously. This yields non-Markovian double pole balancing (Wieland, 1991), which can be considered a difficult benchmark task for control optimization. I use the implementation as found in (Gomez and Miikkulainen, 1997).

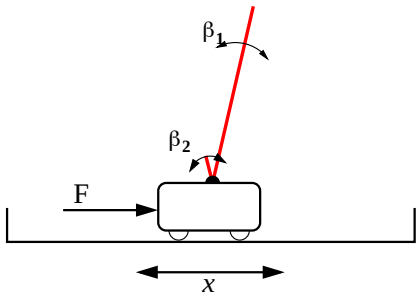

Figure 3.2: **The non-Markovian double pole balancing task.** This task consists of a moving cart on a track, with two poles of different lengths ($1m$ and $0.1m$) hinged on top. The controller applies a (continuous) force F to the cart at every time step, after observing pole angles β_1 and β_2. The objective is to keep the poles from falling for at least $100,000$ time steps (or one half hour in simulated time).

	Markov	non-Markov
1 pole	756 ± 227	2087 ± 607
2 poles	5218 ± 1236	6139 ± 1425

Table 3.1: **RPG results on the pole balancing tasks.** The table shows the results for RPGs on the pole balancing task, for the four possible cases investigated in this chapter: 1 pole Markov, 2 poles Markov, 1 pole non-Markov, and 2 poles non-Markov. The results show the mean and standard deviation of the number of evaluations until the success criterion was reached, that is, when a run lasts more than $100,000$ time steps. Results are computed over 200 runs. All runs achieved the objective within 100,000 trials.

I applied RPGs to the pole balancing task, using a Gaussian output structure for the LSTM recurrent neural network, consisting of two output neurons: a mean μ (which was interpreted linearly) and a standard deviation σ (which was scaled with the sigmoidal logistic function between 0 and 1 in order to prevent variances from being negative) where eligibilities were calculated using the fact that, for the Gaussian policy, $\nabla_\mu \log \pi = \frac{a-\mu}{\sigma^2}$ and $\nabla_\sigma \log \pi = \frac{(a-\mu)^2-\sigma^2}{\sigma^3}$. I utilize a learning rate proportional to the variance $\alpha\sigma^2$ – as suggested in (Williams, 1992) – to prevent numerical instabilities when variances tend to 0, and use learning rate $\alpha = 0.001$, $momentum = 0.9$ and discount factor $\gamma = 0.99$. Initial parameters θ were initialized randomly between -0.01 and 0.01. Reward was always 0.0, except for the last time step when one of the poles falls over, where it is -1.0.

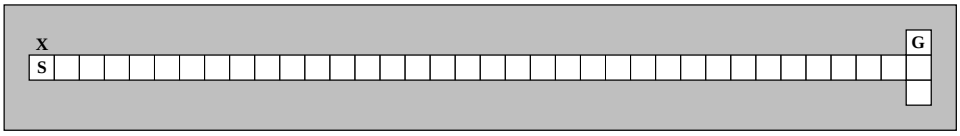

Figure 3.3: **The T-maze task.** The agent observes its immediate surroundings and is capable of the actions goNorth, goEast, goSouth, and goWest. It starts in the position labeled 'S', there and only there observing either the signal 'up' or 'down', indicating whether it should go up or down at the end of the alley. It receives a reward if it goes in the correct direction, and a punishment if not. In this example, the direction is 'up' and N, the length of the alley, is 35.

A run was considered a *success* when the pole(s) did not fall over for $100,000$ time steps. Table 3.1 shows results averaged over 200 runs. RPGs clearly outperform earlier PG methods (for a comparison, see (Meuleau et al., 1999)'s finite state controller, which cannot balance a single pole in a partially observable setting for more than 1000 time steps, even after 500,000 trials). RPGs constitute the only published ontogenetic approach published in the literature that can satisfactorily solve this problem. See Chapter 7 for comparisons to other (phylogenetic) algorithms, including two other novel algorithms published in this thesis, NES (Chapter 4) and FEM (Chapter 5).

3.4.2 Long Term Dependency T-maze

The second experiment was carried out on the T-maze (Bakker, 2002a) (see Figure 3.3). Designed to test an RL algorithm's ability to correlate events far apart in history, it involves having to *learn* to remember the observation from the first time step until the episode ends. At the first time step, it starts at position **S** and perceives the **X** either north or south – meaning that the goal state **G** is in the north or south part of the T-junction, respectively. Additionally, the agent perceives its immediate surroundings. The agent has four possible actions: North, East, South and West. These discrete actions are represented in the network as a softmax layer, which extends Williams' sigmoid units as $\nabla_{net_i} \log \pi = a_i - p_i$ where unit i's firing probability is calculated as $p_i = \frac{\exp(net_i)}{\sum_j \exp(net_j)}$, where net_i represents the net input to output unit i, and a_i indicates whether the i-th action was chosen ($a_i = 1.0$) or not ($a_i = 0.0$). When the agent makes the correct decision at the T-junction, i.e. go south if the **X** was south and north otherwise, it receives a reward of 4.0, otherwise a reward of -0.1. In both cases, this ends the episode. Note that the corridor length N can be increased to make the problem more difficult, since the agent has to learn to remember the initial 'road sign' for $N + 1$ time steps. In Figure 3.3, an example T-maze with corridor length $N = 35$

is shown.

Corridor length N was systematically varied from 10 to 100, and for each length 100 runs were performed. Discount factor $\gamma = 0.98$ was used. For best results, training was performed in batches of 20 normalizing the gradient to length 0.3 (alternatively, using the more 'standard' settings used elsewhere of learning rate $\alpha = 0.001$ and $momentum = 0.9$ without gradient normalization produced results that learned roughly twice as slow, reliably up to corridor $N = 70$). In Figure 3.4 the results are displayed, in addition to other algorithms' results (RL-Elman and RL-LSTM) taken from (Bakker, 2002a), of which the results on RL-LSTM were the best results reported so far. One can see that RPGs clearly outperform these value-based methods, even by more than an order of magnitude in terms of iterations for corridor lengths longer than 40. Additionally, RPGs are able to reliably solve this task up to N=90 (at N=100, it failed to find the optimal policy 22 of of 100 runs), while the second best algorithm, RL-LSTM, solves it up to N=50. The large performance gain on this task for Recurrent Policy Gradients might be due to the difference in complexity of learning a simple (memory-based) policy versus learning unnecessarily complex value functions. The impressive performance advantage of RPGs over value-based methods on this domain could indicate a possibly significant potential for the application of Recurrent Policy Gradients to other deep-memory domains.

3.4.3 The 89-state Maze

In this highly stochastic benchmark task (see Figure 3.5; see (Littman et al., 1995) for a complete description) the aim for the agent is to get to the goal as fast as possible – at the goal the reward is 1, other locations have reward 0 – from a random starting position and orientation, but within 251 time steps. For reward attribution, discount factor $\gamma = 0.98$ is used. The agent has not only a position, but also an orientation, and its actions consist of moving forward, turning left, turning right, turning about, and doing nothing. State transactions are extremely noisy. Observations, which consist of 4 bits representing adjacent wall information – wall (represented 1.0) or no wall (represented 0.0) – are noisy and are inverted with probability 10%, which sets the chance of getting the correct observation somewhere between 0.65 and 0.81, depending on the agent's location. It is interesting to note that, to the author's knowledge, this domain has as of yet not been satisfactorily solved, that is, solved up to human-comparable performance. Humans still greatly outperform all algorithms that the author is aware of. Therefore, the performance on this challenging benchmark task might be considered, to a large extent, indicative of the subjective 'strength' of a given POMDP algorithm.

Because of the random starting position, this task is extremely difficult without the use of any history-dependent baseline, since the agent might

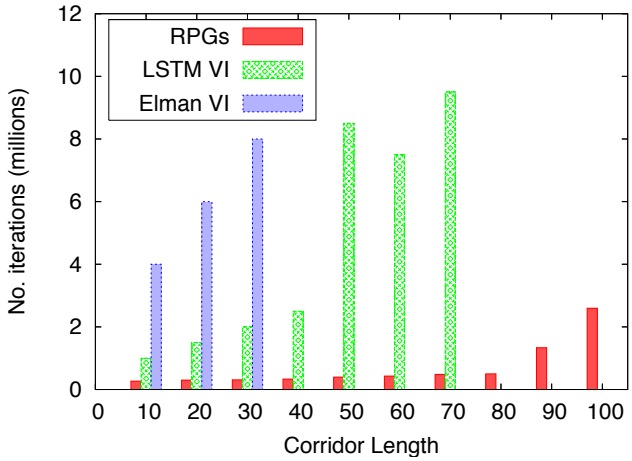

Figure 3.4: **T-maze results for RPGs.** Elman-based Value Iteration (Elman VI) starts to degrade after corridor length $N = 10$, LSTM Value Iteration (LSTM VI) falters after $N = 50$, while Recurrent Policy Gradients' performance starts to degrade at length $N = 100$. The plot shows the number of average iterations required to solve the task, averaged over the successful runs. RPGs clearly outperform other RL methods on this task, to the best of the author's knowledge. The results for the Value Iteration based algorithms are taken from (Bakker, 2002a).

start close to the target or not, which influences the expected rewards accordingly. That is why I apply a history-dependent baseline for this task, trained one step with $\alpha = 0.001$ and $momentum = 0.9$ after every episode. 20 runs were performed to test the performance of the algorithm, using a history-dependent baseline which was trained on actually received returns using a separate LSTM network with 10 memory cells with the same inputs as the policy network including a bias. Each run was executed for $30,000,000$ iterations. After that, the resulting policy was evaluated. The median number of steps to achieve the goal (in case the goal is achieved) was 58, and the goal was reached in 95% of the trials (calculated over 1000 roll-outs). This compares favorably with the second best other (model-free) method the author is aware of, Bakker's RL-LSTM algorithm (Bakker, 2004) with 61 steps and 94%, respectively. See Table 7.3 in Chapter 7 for a comparison to other algorithms, including Fitness Expectation Maximization (Chapter 5) and Natural Evolution Strategies (Chapter 4) that are introduced in this thesis. In (Littman et al., 1995) the median human performance of 29 steps with a 100% success rate is highlighted, which again underlines the difficulty and importance of the task as a benchmark. However, the fact that RPGs outperform all other algorithms on this task might indicate that the appli-

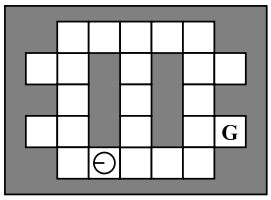

Figure 3.5: **The 89-state maze.** In this highly stochastic maze, the agent has a position, an orientation, and can execute five different actions: forward, turnleft, turnright, turnabout, and doNothing. The agent starts every trial in a random position. Its goal is to move to the square labeled 'G'. Observations comprise the local walls but are noisy (there is a high probability of observing walls where there are none and vice versa). Action outcomes are noisy and cannot be relied on. See (Littman et al., 1995) for a complete description of this problem domain.

cation of Recurrent Policy Gradients to RNNs, especially in combination with history-dependent baselines[1], might indeed be fruitful.

3.4.4 Car Racing with Recurrent Policy Gradients

In order to demonstrate what this algorithm could do in a complicated setting which is also difficult for humans, I have carried out experiments on the TORCS car racing simulator (Torcs, 2007). TORCS is an advanced open source racing game with a graphical user interface and simulated simplified physics which provide a challenging experience for game play. Additionally to being open source, the game was specifically designed for programming competitions between steering agents, and the code framework allows for easy plug-ins of code snippets for competitions between preprogrammed drivers. As such, it provides a perfect testbed for reinforcement learning algorithms that aims to go beyond the current benchmark standards.

I trained the RPG agent on one single track (see Figure 3.6), on which it has to learn to drive a Porsche GT1 and stay on the road while achieving high speed. Whenever the car gets stuck off the road, a learning episode ends, the car is put back on track and a new episode begins. The steering outputs of the RNN, which were executed at a rate of 30 frames per second, are interpreted as a Gaussian with one output neuron interpreted linearly (μ, the mean) and one output neuron interpreted logistically between 0 and 1 (σ, the standard deviation) to ensure it is nonnegative. The four observations

[1] Note that history-dependent baselines are necessary for good performance on this task. Using simply a constant baseline, computed as an average over the last 1000 episodes, but otherwise using the same learning settings, results in the policy reaching the goal 85% of the time, with a median of 70 steps, which is significantly worse than the performance of RPGs using the history-dependent baselines.

Figure 3.6: **The TORCS racing car simulator.**

provided by the TORCS environment were (experimentally, using human driving experience) normalized around 0 with standard deviation 1, and include the speed, the steering angle, position on the road and look-ahead-distance (which was linearly varied with speed), and a bias of 1.0. Its rewards consist of speed measurements (in km/h) at every time step, and the agent receives negative rewards for spending time off track (penalty -100). A large penalty is inflicted upon the car getting stuck off track (penalty -1000), which ends an episode. The car's speed starts off at 10 km/h, which is linearly increased over time to reach 70 km/h after 30 minutes of simulated driving.

I performed 10 runs on this lap using the same learning settings and 10-cell network as applied to the non-Markovian double pole balancing task. Training was executed and the (average) baseline was updated after every 1000 time steps. It was found that the agent learns, for all runs, to consistently steer and stay on the road after just under 2 minutes of real-time behavior. In all runs, the car first drives off the track immediately four or five times, then learns to stay on track until it hits the first curve, where it slides off again. Within two minutes, however, it drives nearly perfectly in the middle of the road, and learns to 'cut curves' slightly when the speed is increased gradually to 70 km/h after 30 minutes. The agent can learn to drive safely – not getting off track – up to 70 km/h, after which its behavior destabilized in all runs. Future work will investigate how to make the behavior more robust and how to cope with higher speeds. This will have to include speed control and braking by the network as well, which could be actualized using additional (softmax) output neurons for gears, brakes and gas. The fastest lap time achieved after 30 minutes of training was just under 3 minutes, which is, unfortunately, still twice as slow as a trained human player or the preprogrammed agent.

To conclude, the car driving agent learns fast, in real-time (2 minutes), to steer correctly and keep the vehicle on the road. This is about as fast

as a novice human player learns to stay on the road. Moreover, it reaches high speeds of up to 70 km/h within 30 minutes of online training time. Learning to drive relatively well relatively quickly can be done by RPGs because it is an ontogenetic algorithm, updating its policy online step by step. Although rigid preprogrammed speed control destabilizes the agent with higher speeds, the fast learning suggests this approach might be worth investigating when dealing with real-time learning problems in continuous robot control.

3.5 Conclusion and Future Work

In this chapter, I have introduced Recurrent Policy Gradients, an elegant and powerful method for dealing with reinforcement learning in partially observable environments. The algorithm, an RNN-based policy gradient method equipped with memory capable of memorizing events from arbitrarily far in the past, involves computing and backpropagating action eligibilities through time with 'Long Short-Term Memory' memory cells, thus updating a policy which maps event histories to action probabilities. RPGs constitute the first ontogenetic reinforcing learning method to solve the non-Markovian double pole balancing task, and the approach outperformed other ontogenetic RL methods on three important benchmarks with different characteristics. It is not unlikely that Recurrent Policy Gradients would constitute both one of the simplest, and one of the most efficient RL algorithms to date for difficult non-Markovian tasks.

Future investigations may involve the use of natural gradients (Peters and Schaal, 2006, 2008b) and other gradient optimization methods to optimize performance (e.g., Schraudolph et al., 2006). Extending the estimation of history-dependent baselines to include temporal difference methods could constitute another profitable line of research.

Chapter 4

Natural Evolution Strategies

This chapter presents Natural Evolution Strategies to optimize unknown 'fitness' functions in order to perform phylogenetic reinforcement learning. Natural Evolution Strategies form a novel algorithm that constitutes a principled alternative to standard black box optimization methods such as evolutionary search. It maintains a multinormal search distribution on the set of solution candidates. The natural gradient is used to update the distribution's parameters in the direction of higher expected fitness. Using fitness baselines and importance mixing (a procedure adjusting batches with minimal numbers of fitness evaluations), the algorithm yields competitive results on a number of black box benchmarks as well as on the hard non-Markovian double pole balancing RL task.

4.1 Introduction

Evolutionary algorithms and other black box optimization algorithms aim to optimize a 'fitness' function that is either unknown or too complex to model directly. They allow domain experts to search for good or near-optimal solutions to numerous difficult real-world problems in areas ranging from medicine and finance to control and robotics. In particular, these algorithms can be applied in phylogenetic reinforcement learning to learn a controller's parameters directly, for example, by using the evolved genome to a recurrent neural network which takes observations as inputs and produces actions as outputs, and by using a fitness measure which corresponds to the experienced returns of episode roll-outs. By estimating a controller's parameters directly, one can circumvent the need to estimate a value function (see Chapter 2).

A variety of algorithms has been developed within this framework, including methods such as the cross-entropy method (Rubinstein and Kroese, 2004), simultaneous perturbation stochastic optimization (Spall, 1999), hill climbing, particle swarm optimization (Kennedy and Eberhart, 2001) and

the class of evolutionary algorithms, of which evolution strategies (ES, Rechenberg, 1971; Beyer and Schwefel, 2002; Beyer, 1996), and in particular its covariance matrix adaption instantiation (CMA-ES: Hansen and Ostermeier, 2001), are most relevant to the work presented in this chapter.

Evolution strategies, so named because of their inspiration from natural evolution, generally produce consecutive batches ('generations') of samples ('genomes'). During each generation, a batch of samples is generated by perturbing the parents' parameters – *mutating* their genes, if you will [1]. A number of samples is *selected*, based on their fitness values, while the less fit individuals are discarded. The winners are then used as parents for the next generation. This process typically leads to increasing fitness over the generations. The basic ES framework, though simple and heuristic in nature, has proven to be very powerful and robust, spawning a wide variety of algorithms.

Evolution strategies generally generate mutations using a normal distribution with specific mutation sizes associated with every objective parameter. One of the major research topics in evolution strategies concerns the automated adjustment of the mutation sizes for the production of new samples, a procedure generally referred to as *self-adaptation*. Obviously, choosing mutation sizes that are too large will produce debilitating mutations and ensure that convergence to a sufficiently fit region of parameter space will be prevented. Excessively small mutation sizes tend to lead to overly slow and premature convergence. Generally the mutation size must be chosen from a small range of values specific to both the problem domain and to the distribution of current individuals on the fitness landscape. Evolution strategies must therefore adapt mutation during evolution, based on the progress made on the recent evolution path. Often this is done by simultaneously evolving both objective parameters and the corresponding mutation sizes. This has been shown to produce excellent results in a number of cases (e.g., Beyer, 1996).

The covariance matrix adaptation algorithm CMA-ES constitutes a more sophisticated approach. CMA-ES adapts a variance-covariance mutation *matrix* $\mathbf{\Sigma}$ from which mutations are drawn. This enables the algorithm to generate *correlated* mutations, speeding up evolution significantly for many real-world fitness landscapes. Self-adaptation of this mutation matrix is then achieved by integrating information on successful mutations on its recent evolution path, by making similar mutations more likely. CMA-ES performs excellently on most standard benchmark tasks. One of the problems with the CMA-ES algorithm, however, is its ad-hoc nature and relatively complex or even contrived set of mathematical justifications and 'rules of thumb'.

[1] note that ES generally uses asexual reproduction: new individuals typically are produced without using crossover or similar techniques that are prevalent in the field of genetic algorithms

Another problem pertains to its sensitivity to local optima.

Typically, three objectives have to be kept in mind when developing evolutionary algorithms—one wants (1) robust performance; (2) few (potentially costly) fitness evaluations; (3) easy tuning of hyperparameters.

In this chapter, Natural Evolution Strategies (NES: Wierstra et al., 2008a; Yi et al., 2009b,a) are presented, a new class of evolutionary algorithms less ad-hoc than traditional evolutionary methods.

The NES algorithm, like CMA-ES, maintains and iteratively updates a multinormal mutation distribution. Parameters are updated by estimating a *natural evolution gradient*, i.e. the natural gradient on the parameters of the mutation distribution, and following it towards better expected fitness. Well-known advantages of natural gradient methods include isotropic convergence on ill-shaped fitness landscapes (Amari and Douglas, 1998). This avoids drawbacks of 'vanilla' (regular) gradients which are prone to slow or premature convergence (Peters and Schaal, 2008b). In conjunction with the techniques of *fitness baselines* and *fitness shaping* this yields robust performance (objective 1).

To reduce the number of potentially costly evaluations (objective 2), we utilize the recently introduced method of *importance mixing*, a type of steady-state enforcer which keeps the distribution of the new population conformed to the current mutation distribution, while allowing the algorithm to recycle old samples, as such significantly reducing the number of required new samples in order to make an update.

An additional advantage of importance mixing is that it renders the algorithm relatively insensitive to the batch size. This, combined with the fact that a learning rate of 1 is nearly optimal for natural gradients, makes the algorithm easy to work with in practice (objective 3).

The resulting algorithm, Natural Evolution Strategies, is elegant, requires no additional heuristics and has few parameters that need tuning. It performs consistently well on both standard unimodal and multimodal benchmarks, and shows highly competitive performance on the difficult non-Markovian double pole balancing task and other phylogenetic RL settings (see Chapter 7).

The chapter is organized as follows. The next section provides a quick overview of the general problem framework of black box function optimization. The ensuing sections describe the derivation of the 'vanilla' gradient approach, the concept of 'fitness shaping', importance mixing and the natural gradient instantiation of the algorithm. The section on experimental results describes NES' performance on the standard comparison set of unimodal and multimodal benchmark problems used in the literature, as well as its results on the pole balancing benchmark. The chapter concludes with a discussion, and points out some possible directions for future work.

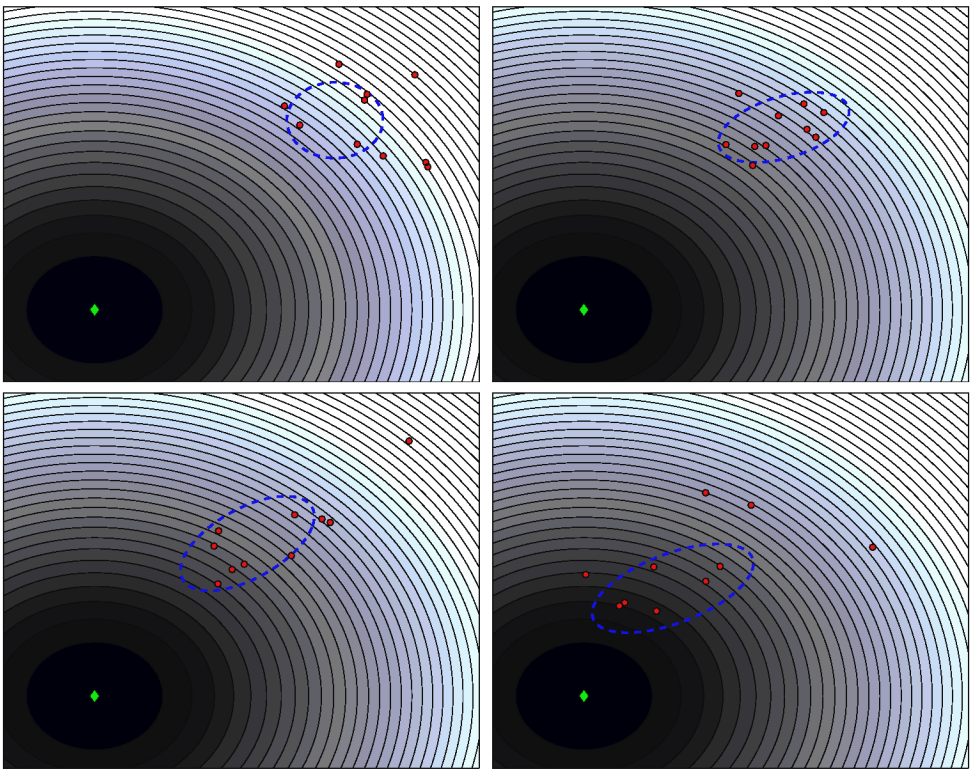

Figure 4.1: **The evolution, over four generations, of Gaussian-generated sample points and the ellipsoids representing their corresponding mutation covariance matrices on a 2-dimensional fitness landscape.**

4.2 Algorithm Framework

In this section, first the algorithm framework and the corresponding notation are introduced. The objective is to optimize the d-dimensional continuous vector of objective parameters $\mathbf{x}$ for an unknown fitness function $f : \mathbb{R}^d \rightarrow \mathbb{R}$. The function is unknown or 'black box', in that the only information accessible to the algorithm consists of function measurements selected by the algorithm. The goal is to optimize $f(\mathbf{x})$, while keeping the number of function evaluations – which are considered costly – as low as possible. This is done by evaluating a number $1 \dots N$ of separate individuals $\mathbf{z}_1 \dots \mathbf{z}_N$ each successive generation g, using the information from fitness evaluations $f(\mathbf{z}_1) \dots f(\mathbf{z}_N)$ to adjust both the current candidate objective parameters $\mathbf{x}$ and the mutation sizes. Figure 4.1 shows what a hypothetical search might look like.

In conventional evolution strategies, optimization is achieved by mimicking natural evolution: at every generation, parent solution $\mathbf{x}$ produces

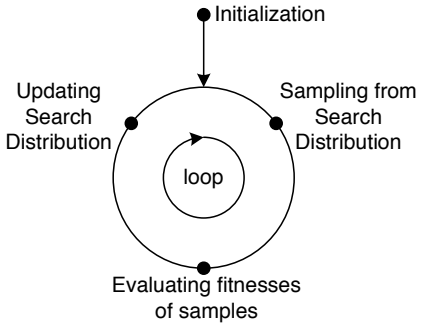

Figure 4.2: **A schematic depiction of the black box framework as used in phylogenetic reinforcement learning.** First, samples are produced (e.g., the weights of an RNN representing a policy). Then, fitness evaluations are performed for the newly produced samples. Depending on the fitness values found, the search distribution ('population') is adjusted towards producing likely more fit individuals.

offspring $\mathbf{z}_1 \dots \mathbf{z}_N$ by *mutating* string $\mathbf{x}$ using a multivariate normal distribution with zero mean and some variance σ. After evaluating all individuals, the best M individuals are kept (*selected*), stored as candidate solutions and subsequently used as 'parents' for the next generation. This simple process is known to produce excellent results for a number of challenging problems. A schematic depiction of the black box framework is given in Figure 4.2.

4.3 'Vanilla' Gradients for Evolution Strategies

NES is different from conventional evolution strategies in one important respect. Instead of 'wasting' information by discarding low-fitness samples, it aims to use all available fitness values, even the bad ones, to generate a *gradient* for updating the population.

The core idea is that one wants to optimize expected 'fitness' $J = \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})]$ of the next generation. At every generation g, a population $\pi^{(g)}$ parameterized by $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$ is assumed, representing the current candidate solution ('parent') $\mathbf{x}$ and mutation matrix $\mathbf{\Sigma}$ used for producing the next generation of search points.

In order to adjust parameters $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$ towards solutions that are likely more fit, we estimate a gradient on θ for the expected fitness. Now let $f(\mathbf{z})$ be the fitness at a particular search point $\mathbf{z}$, and, utilizing the familiar multivariate normal distribution, let

$g \leftarrow 1$
initialize population parameters $\theta^{(g)} = \langle \mathbf{x}, \mathbf{\Sigma} = \mathbf{A^T A} \rangle$
repeat
  for $k = 1 \dots N$ **do**
    draw $\mathbf{z}_k \sim \pi(\mathbf{x}, \mathbf{\Sigma})$ using importance mixing
    evaluate cost of $f'(\mathbf{z}_k)$
    $\nabla_{\mathbf{x}} \log \pi(\mathbf{z}_k) = \mathbf{\Sigma}^{-1}(\mathbf{z}_k - \mathbf{x})$
    $\nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}_k) =$
      $\frac{1}{2}\mathbf{\Sigma}^{-1}(\mathbf{z}_k - \mathbf{x})(\mathbf{z}_k - \mathbf{x})^{\mathbf{T}}\mathbf{\Sigma}^{-1} - \frac{1}{2}\mathbf{\Sigma}^{-1}$
    $\nabla_{\mathbf{A}} \log \pi(\mathbf{z}_k) = \mathbf{A}\left[\nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}_k) + \nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}_k)^{\mathbf{T}}\right]$
  $\mathbf{\Phi} = \begin{bmatrix} \nabla_{\mathbf{x}} \log \pi(\mathbf{z}_1) & \nabla_{\mathbf{A}} \log \pi(\mathbf{z}_1) & 1 \\ \vdots & \vdots & \vdots \\ \nabla_{\mathbf{x}} \log \pi(\mathbf{z}_N) & \nabla_{\mathbf{A}} \log \pi(\mathbf{z}_N) & 1 \end{bmatrix}$
  $\mathbf{R} = [f'(\mathbf{z}_1), \dots, f'(\mathbf{z}_N)]^{\mathbf{T}}$
  $\delta\theta = (\mathbf{\Phi^T \Phi})^{-1}\mathbf{\Phi^T R}$
  $\theta^{(g+1)} \leftarrow \theta^{(g)} - \alpha \cdot \delta\theta$
  $g \leftarrow g + 1$
until stopping criterion is met

Algorithm 4.1: Pseudocode for the Natural Evolution Strategies algorithm.

$$\begin{aligned} \pi(\mathbf{z}|\theta) &= \mathcal{N}(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}) \\ &= \frac{1}{(2\pi)^{n/2}|\mathbf{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{z} - \mathbf{x})^{\mathbf{T}}\mathbf{\Sigma}^{-1}(\mathbf{z} - \mathbf{x})\right] \end{aligned}$$

denote the probability density of search point $\mathbf{z}$ given the current population $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$. Expected fitness can then be expressed as

$$\begin{aligned} J &= \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})] \\ &= \int \pi(\mathbf{z}|\theta) f(\mathbf{z}) d\mathbf{z}. \end{aligned}$$

Taking the derivative of J with respect to θ of population π, one can write

$$\begin{aligned} \nabla_\theta J &= \nabla_\theta \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})] \\ &= \int \nabla_\theta \pi(\mathbf{z}|\theta) f(\mathbf{z}) d\mathbf{z} \\ &= \int \frac{\pi(\mathbf{z}|\theta)}{\pi(\mathbf{z}|\theta)} \nabla_\theta \pi(\mathbf{z}|\theta) f(\mathbf{z}) d\mathbf{z} \\ &= \int \pi(\mathbf{z}|\theta) \nabla_\theta \log \pi(\mathbf{z}|\theta) f(\mathbf{z}) d\mathbf{z} \end{aligned}$$

using the 'likelihood-ratio' trick. Taking a Monte Carlo approximation of this expectation by choosing N search points yields

$$\begin{aligned}\nabla_\theta J &= \mathbf{E}_{\mathbf{z}}\left[\nabla_\theta \log \pi(\mathbf{z}|\theta) f(\mathbf{z})\right] \\ &\approx \frac{1}{N}\sum_{k=1}^{N} \nabla_\theta \log \pi(\mathbf{z}_k|\theta) f(\mathbf{z}_k).\end{aligned}$$

The population parameter vector $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$ is comprised of both the current candidate solution center and its mutation matrix, concatenated in one single vector. In order to calculate the derivatives of the log-likelihood with respect to individual elements of θ for this mixture of multivariate normal distributions, first note that

$$\begin{aligned}\log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right) &= \frac{n}{2}\log(2\pi) - \frac{1}{2}\log\det \mathbf{\Sigma} \\ &- \frac{1}{2}\left(\mathbf{z}-\mathbf{x}\right)^{\mathbf{T}} \mathbf{\Sigma}^{-1}\left(\mathbf{z}-\mathbf{x}\right).\end{aligned}$$

Obtaining its derivatives will be necessary, that is, $\nabla_{\mathbf{x}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right)$ and $\nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right)$. The first is simply

$$\nabla_{\mathbf{x}} \log \pi\left(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}\right) = \mathbf{\Sigma}^{-1}\left(\mathbf{z}-\mathbf{x}\right),$$

while the latter is

$$\nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}|\theta\right) = \frac{1}{2}\mathbf{\Sigma}^{-1}\left(\mathbf{z}-\mathbf{x}\right)\left(\mathbf{z}-\mathbf{x}\right)^{\mathbf{T}}\mathbf{\Sigma}^{-1} - \frac{1}{2}\mathbf{\Sigma}^{-1}.$$

Mutation matrix $\mathbf{\Sigma}$ needs to be constrained, though, in order to preserve symmetry, ensure positive variances and to keep it positive semi-definite. One can accomplish that by representing $\mathbf{\Sigma}$ as a product $\mathbf{\Sigma} = \mathbf{A}^{\mathbf{T}}\mathbf{A}$. Instead of using the log-derivatives on $\nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}\right)$ directly, the derivatives with respect to $\mathbf{A}$ can be computed as

$$\nabla_{\mathbf{A}} \log \pi\left(\mathbf{z}_k\right) = \mathbf{A}\left[\nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}_k\right) + \nabla_{\mathbf{\Sigma}} \log \pi\left(\mathbf{z}_k\right)^{\mathbf{T}}\right].$$

Using these derivatives to estimate $\nabla_\theta J$, one can then update parameters $\theta = \langle \mathbf{x}, \mathbf{\Sigma} = \mathbf{A}^{\mathbf{T}}\mathbf{A} \rangle$ as

$$\theta \leftarrow \theta + \alpha \nabla_\theta J$$

using learning rate α. This produces a new candidate solution $\mathbf{x}^{(g+1)}$ each generation of N sample points, and simultaneously self-adapts the associated mutation matrix to $\mathbf{\Sigma}^{(g+1)}$. This simple update rule, which covers both candidate parameters and mutation parameters in one single framework, is in marked contrast to the complicated and ad-hoc and overly heuristic nature of most ES algorithms such as CMA-ES.

4.4 Fitness Shaping

For problems with wildly fluctuating fitnesses, the gradient is disproportionately distorted by extreme fitness values, which can lead to premature convergence or numerical instability. To overcome this problem, we use *fitness shaping*, an order-preserving nonlinear fitness transformation function (Wierstra et al., 2008a). The choice of (monotonically increasing) fitness shaping function is arbitrary, and should therefore be considered to be one of the tuning parameters of the algorithm. It was empirically found that ranking-based shaping functions work best for all problems that were tackled. The shaping function used for all experiments in this thesis was fixed to $f'(\mathbf{z}) = 2i - 1$ for $i > 0.5$ and $f'(\mathbf{z}) = 0$ for $i < 0.5$, where i denotes the relative rank of $f(\mathbf{z})$ in the population, scaled between $0 \ldots 1$.

4.5 Importance Mixing

At each generation, one evaluates N new individuals generated from population $\pi(\mathbf{z}|\theta)$. However, since small updates ensure that the Kullback-Leibler divergence between consecutive mutation distributions is generally small, most new individuals will fall in the high density area of the previous mutation distribution $\pi(\mathbf{z}|\theta')$. This leads to redundant fitness evaluations in that same area.

One solution to this problem is a procedure called *importance mixing* introduced in (Yi et al., 2009b,a), which aims to *reuse* fitness evaluations from the previous generation, while ensuring the updated population conforms to the new mutation distribution.

Importance mixing works in two steps: In the first step, rejection sampling is performed on the previous population, such that individual $\mathbf{z}$ is accepted with probability

$$\min\left\{1, (1-\eta)\,\frac{\pi(\mathbf{z}|\theta)}{\pi(\mathbf{z}|\theta')}\right\}.$$

Here $\eta \in [0, 1]$ is the *minimal refresh rate*. Let N_a be the number of individuals accepted in the first step. In the second step, reverse rejection sampling is performed as follows: Generate individuals from $\pi(\mathbf{z}|\theta)$ and accept $\mathbf{z}$ with probability

$$\max\left\{\eta, 1 - \frac{\pi(\mathbf{z}|\theta')}{\pi(\mathbf{z}|\theta)}\right\}$$

until $N - N_a$ new individuals are accepted. The N_a individuals from the old generation and $N - N_a$ newly accepted individuals together constitute the new population. Note that only the fitnesses of the newly accepted individuals need to be evaluated. The advantage of using importance mixing is twofold: On the one hand, it reduces the number of fitness evaluations

required in each generation, on the other hand, if ones fixes the number of newly evaluated fitnesses, then many more fitness evaluations can potentially be used to yield more reliable and accurate gradients.

The minimal refresh rate η lower bounds the expected proportion of newly evaluated individuals $\rho = \mathbf{E}\left[\frac{N-N_a}{N}\right]$, namely $\rho \geq \eta$, with the equality holding iff $\theta = \theta'$. In particular, if $\eta = 1$, all individuals from the previous generation will be discarded, and if $\eta = 0$, ρ depends only on the distance between $\pi(\mathbf{z}|\theta)$ and $\pi(\mathbf{z}|\theta')$. Normally one should set η to be a small positive number, e.g. 0.01 (as utilized throughout this work), to avoid too low an acceptance probability at the second step when $\pi(\mathbf{z}|\theta')/\pi(\mathbf{z}|\theta) \simeq 1$.

The updated population conforms to the mutation distribution $\pi(\mathbf{z}|\theta)$. In the region where $(1-\eta)\pi(\mathbf{z}|\theta)/\pi(\mathbf{z}|\theta') \leq 1$, the probability that an individual from previous generations appears in the new population is

$$\pi\left(\mathbf{z}|\theta'\right) \cdot (1-\eta)\pi(\mathbf{z}|\theta)/\pi\left(\mathbf{z}|\theta'\right) = (1-\eta)\pi(\mathbf{z}|\theta).$$

The probability that an individual generated from the second step entering the population is $\eta\pi(\mathbf{z}|\theta)$, since

$$\max\left\{\eta, 1-\pi\left(\mathbf{z}|\theta'\right)/\pi(\mathbf{z}|\theta)\right\} = \eta.$$

So the probability of an individual entering the batch is just $\pi(\mathbf{z}|\theta)$ in that region. The same result is valid for the region where it holds that $(1-\eta)\pi(\mathbf{z}|\theta)/\pi(\mathbf{z}|\theta') > 1$.

Measuring the usefulness of importance mixing (Yi et al., 2009b,a), it was empirically found that it reduces the number of required fitness evaluations for all experiments by at least a factor 3. Additionally, it reduced the algorithm's sensitivity to both the learning rate and the population size. Using larger populations has the effect that more samples tend to get reused simply because more sample points in the new region will be available. Using smaller learning rates has the same effect, since more samples will then fall in the overlapping high density area of consecutive parameter sets. While importance mixing does not render NES parameter-free, the near-desensitization to both batch size hyperparameter N and learning rate α makes the application of NES rather convenient to the user: in effect, only the initial search region defined by θ and the fitness shaping function need to be determined.

4.6 Natural Evolution Strategies

Standard gradient methods have been shown to converge slowly on fitness landscapes with ridges and plateaus. An ad-hoc and often-used method for overcoming this would be the use of momentum. Natural gradients constitute a more principled approach, however. First introduced by Amari

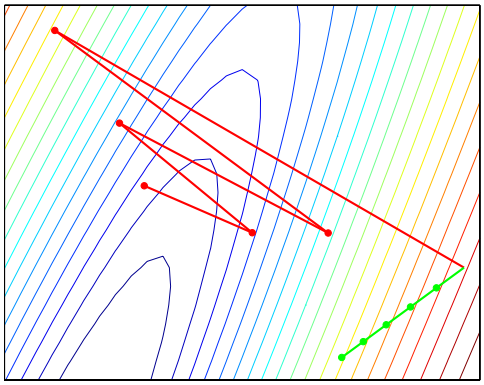

Figure 4.3: **Vanilla gradients and natural gradients on a valley-shaped landscape.** The vanilla gradients zigzag around the valley, while the natural gradient points towards the steepest descent.

in (Amari, 1998), natural gradients have numerous advantages over 'vanilla' gradients (see Figure 4.3 for a schematic depiction of a vanilla and a natural gradient on the same landscape).

The traditional gradient ∇J simply follows the steepest descent in the space of the actual parameters. While this might be a good idea in many problems, the main drawback comes if one needs to maintain uncertainty as it generates the necessary exploration for the solutions. In this case, one needs to stay close to the presented type of solutions while maximizing the fitness. As the solutions are produces as random samples, it is necessary to use a measure of distance $D(\theta'||\theta)$ between probability distributions $\pi_\theta(\mathbf{z})$ and $\pi_{\theta'}(\mathbf{z})$. The natural measure distance between two probability distributions is the Kullback-Leibler divergence. In this case, a special case of the natural gradient is considered with natural gradient update $\delta\theta$ calculated as

$$\begin{aligned} & \max_{\delta\theta} J(\theta + \delta\theta) = \delta\theta^{\mathbf{T}} \nabla J, \\ \textit{such that} \quad & D(\theta + \delta\theta||\theta) = \varepsilon, \end{aligned}$$

where $J(\theta)$ is the expected fitness of population π parameterized by θ, $\delta\theta$ is the direction of constrained steepest descent, ∇J is the steepest descent or gradient, $D(\theta + \delta\theta||\theta)$ a measure of closeness on probability distributions (the Kullback-Leibler divergence) and ε a small increment size.

The constraints impose a geometry on θ which differs from the Euclidean one. If one uses this gradient, the Kullback-Leibler divergences between updates on the search distribution tend toward constancy as the learning rate goes down. For a simple Gaussian, this would mean that a large step on the mean would correspond to a smaller step on the variance, and vice versa. Intuitively, this constitutes a very favorable exploration-exploitation trade-off in black box optimization: if the mean is close to the optimum,

exploration is automatically reduced for finetuning, while if it is not, the algorithm keeps exploring by maintaining larger variances.

If $D(\theta + \delta\theta||\theta)$ is the Kullback-Leibler divergence, it follows that

$$D(\theta + \delta\theta||\theta) = \delta\theta^{\mathbf{T}}\mathbf{F}(\theta)\,\delta\theta + (\text{const}),$$

for small $\delta\theta \to 0$, where

$$\begin{aligned}\mathbf{F}(\theta) &= \int \pi(\mathbf{z})\,\nabla \log \pi(\mathbf{z})\,\nabla \log \pi(\mathbf{z})^{\mathbf{T}}\,d\mathbf{z}, \\ &= \mathbf{E}\left[\nabla \log \pi(\mathbf{z})\,\nabla \log \pi(\mathbf{z})^{\mathbf{T}}\right]\end{aligned}$$

is the Fisher information matrix which yields the natural gradient $\delta\theta$ defined by the necessary condition

$$\mathbf{F}(\theta)\,\delta\theta = \alpha\nabla J,$$

with α being the learning rate. Additionally, one can introduce a *fitness baseline* b (analogously to the one introduced in Chapter 3) as

$$\begin{aligned}\nabla J &= \int \nabla\pi(\mathbf{z})\,f(\mathbf{z})\,d\mathbf{z} + 0 \\ &= \int \nabla\pi(\mathbf{z})\,f(\mathbf{z})\,d\mathbf{z} + b\nabla\underbrace{\int \pi(\mathbf{z})\,d\mathbf{z}}_{=1} \\ &= \int \nabla\pi(\mathbf{z})\,f(\mathbf{z})\,d\mathbf{z} + b\int \nabla\pi(\mathbf{z})\,d\mathbf{z} \\ &= \int \nabla\pi(\mathbf{z})\,[f(\mathbf{z}) - b]\,d\mathbf{z} \\ &= \int \pi(\mathbf{z})\,\nabla \log \pi(\mathbf{z})\,[f(\mathbf{z}) - b]\,d\mathbf{z} \\ &= \mathbf{E}\left[\nabla \log \pi(\mathbf{z})\,[f(\mathbf{z}) - b]\right].\end{aligned}$$

Thus, the fitness baseline parameter b is obtained, which can be used to reduce the estimation variance $\text{Var}\left[\nabla \log \pi(\mathbf{z})\,[f(\mathbf{z}) - b]\right]$. Note that

$$\text{Var}\left[\nabla \log \pi(\mathbf{z})\,Cf(\mathbf{z})\right] = \text{Var}\left[\nabla \log \pi(\mathbf{z})\,f(\mathbf{z})\right]C^2,$$

that is, the variance grows quadratically with the average magnitude of the fitnesses. It can be significantly reduced if a proper fitness baseline is used, reducing the number of samples required to correctly estimate the gradient. This changes the equation to

$$\mathbf{E}\left[\phi(\mathbf{z})\,\phi(\mathbf{z})^{\mathbf{T}}\right]\delta\theta = \mathbf{E}\left[\phi(\mathbf{z})\,f(\mathbf{z})\right] - \mathbf{E}\left[\phi(\mathbf{z})\,b\right]$$

with $\phi(\mathbf{z}) = \nabla \log \pi(\mathbf{z})$ with one open parameter b. Fitness baseline b can be obtained by realizing the lower bound

$$\begin{aligned}
&\mathrm{Var}\left[\phi(\mathbf{z})\left[f(\mathbf{z}) - b\right]\right] \\
&\quad = \bar{f}^2 \,\mathrm{Var}\left[\phi(\mathbf{z})\frac{\left(f(\mathbf{z}) - \bar{f}\right)}{\bar{f}}\right] + \mathrm{Var}\left[\phi(\mathbf{z})\left[\bar{f} - b\right]\right], \\
&\quad \geq \bar{f}^2 \mathbf{E}\left[\phi(\mathbf{z})\,\phi(\mathbf{z})^{\mathbf{T}}\right] + \mathrm{Var}\left[\phi(\mathbf{z})\left[\bar{f} - b\right]\right],
\end{aligned}$$

where $\bar{f} = \mathbf{E}\left[f(\mathbf{z})\right]$. Thus, the minimum is at $b = \bar{f}$, that is, the average fitness. As $\mathbf{E}\left[\phi(\mathbf{z})\right] = 0$, it follows that

$$\begin{aligned}
0 + b &= \mathbf{E}\left[f(\mathbf{z})\right], \\
\mathbf{E}\left[\phi(\mathbf{z})\right]^{\mathbf{T}} \delta\theta + b &= \mathbf{E}\left[f(\mathbf{z})\right].
\end{aligned}$$

Now, this yields the equation system

$$\begin{aligned}
\mathbf{E}\left[\phi(\mathbf{z})\,\phi(\mathbf{z})^{\mathbf{T}}\right]\delta\theta + \mathbf{E}\left[\phi(\mathbf{z})\,b\right] &= \mathbf{E}\left[\phi(\mathbf{z})\,f(\mathbf{z})\right] \\
\mathbf{E}\left[\phi(\mathbf{z})\right]^{\mathbf{T}}\delta\theta + b &= \mathbf{E}\left[f(\mathbf{z})\right].
\end{aligned}$$

This system can be solved straightforwardly as a linear regression problem using the pseudoinverse, and when replacing the $\mathbf{E}\left[\cdot\right]$ by sample averages, this yields the general natural gradient estimator

$$\delta\theta = (\mathbf{\Phi}^{\mathbf{T}}\mathbf{\Phi})^{-\mathbf{1}}\mathbf{\Phi}^{\mathbf{T}}\mathbf{R}$$

where

$$\begin{aligned}
\mathbf{\Phi} &= \begin{bmatrix} \nabla_\theta \log \pi(\mathbf{z}_1) & 1 \\ & \vdots \\ \nabla_\theta \log \pi(\mathbf{z}_N) & 1 \end{bmatrix} \\
\mathbf{R} &= \left[f(\mathbf{z}_1), \ldots, f(\mathbf{z}_N)\right]^{\mathbf{T}}
\end{aligned}$$

The resulting Natural Evolution Strategies algorithm is outlined as pseudocode in Algorithm 4.1

4.7 Experiments

The tunable parameters of Natural Evolution Strategies are comprised of the population size N, the learning rate α, the refresh rate η (always set to 0.01) and the fitness shaping function. In addition, the initial search region must be specified by setting $\mathbf{\Sigma}$ and $\mathbf{x}$ to appropriate values.

Empirically it was found that a good and robust choice for the learning rate α is 1.0 for all problems. Therefore, the only parameter that needs tuning in practice is the batch size N, which is dependent on both the expected

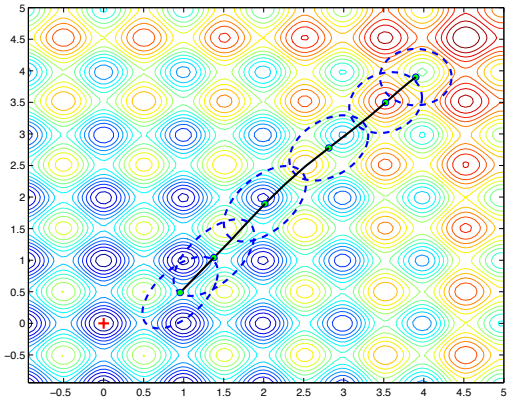

Figure 4.4: **The evolution of the mutation matrices over the generations for the Rastrigin benchmark.** Shown are the $\mathbf{\Sigma}$-defined ellipsoids that correspond to 0.5 standard deviations of the consecutive mutation distributions imposed on the fitness landscape of the multimodal Rastrigin benchmark (note that the optimum is at $(0, 0)$).

ruggedness of the fitness landscape and the problem dimensionality. N was taken to be 15 times the problem dimensionality d, and it was empirically found that this setting, in combination with importance mixing, led to near-optimal results for all experiments in this thesis as compared to other batch sizes and/or learning rates.

4.7.1 Standard Benchmark Functions

The algorithm was empirically tested on the 8 unimodal and 4 multimodal functions out of the set of standard benchmark functions that are typically used in the black box optimization literature (see, e.g., Suganthan et al., 2005; Hansen and Ostermeier, 2001), both for comparison purposes and for competitions. The initial guess was chosen randomly at average distance 1 from the optimum. In order to prevent potentially biased results, following (Suganthan et al., 2005), the functions' inputs were consistently transformed (by a combined rotation and translation), making the variables non-separable and avoiding trivial optima (e.g. at the origin). This procedure immediately renders many other methods virtually useless, since they cannot cope with correlated mutation directions. NES, however, is invariant under translation and rotation. In addition, the rank-based fitness shaping makes it invariant under order-preserving transformations of the fitness function.

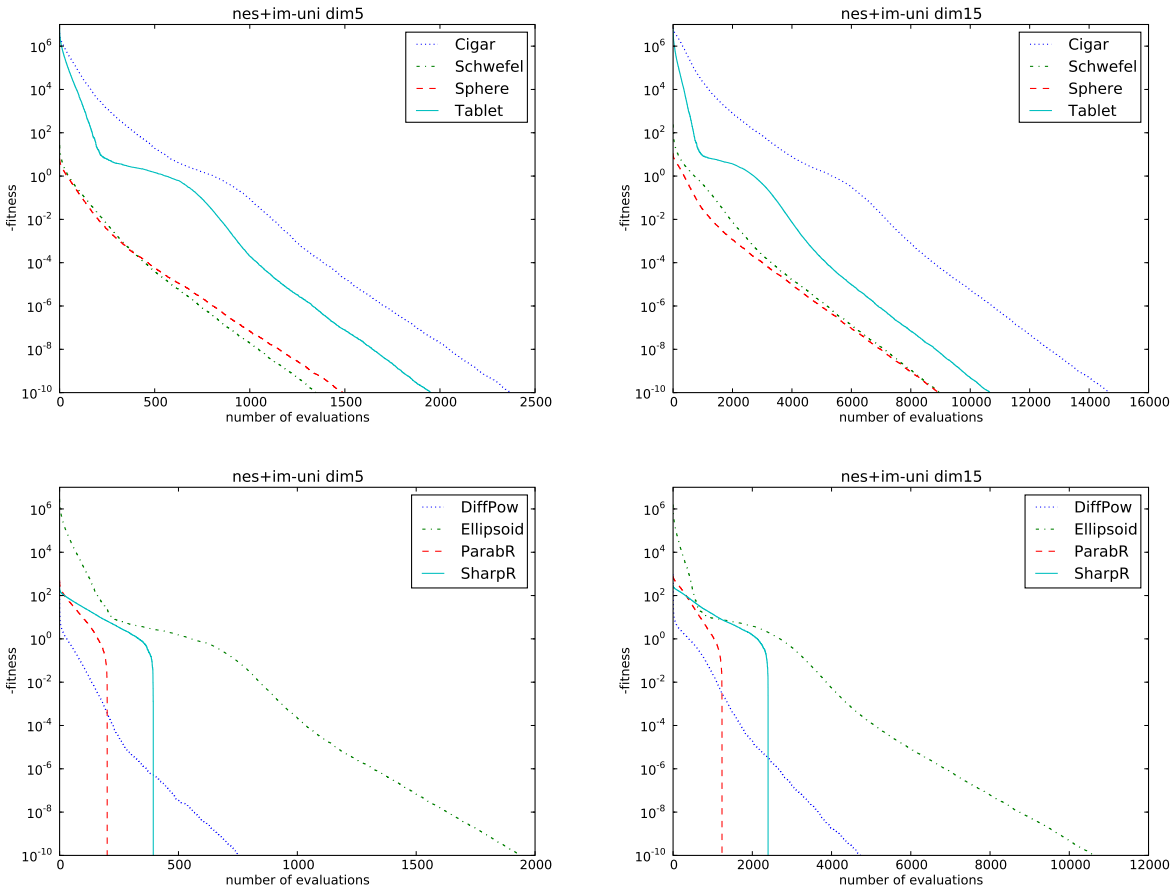

Figure 4.5: **Results for NES on the unimodal benchmark functions.** Left: results for NES experiments on the unimodal benchmark functions with dimensionality 5. Right: results for the unimodal benchmark functions with dimensionality 15. Shown are averages over 200 runs.

Name	Type	Function
Sphere	unimodal	$\sum_{j=1}^{d} z_j^2$
Schwefel	unimodal	$\sum_{j=1}^{d} \left[\sum_{k=1}^{j} z_j\right]^2$
Tablet	unimodal	$(1000 z_1)^2 + \sum_{j=2}^{d} z_j^2$
DiffPow	unimodal	$\sum_{j=1}^{d} \lvert z_j \rvert^{2+10\frac{i-1}{d-1}}$
Ellipsoid	unimodal	$\sum_{j=1}^{d} \left(z_j 1000^{\frac{i-1}{d-1}}\right)^2$
SharpR	unimodal	$-z_1 + 100\sqrt{\sum_{j=2}^{d} z_j^2}$
ParabR	unimodal	$-z_1 + 100 \sum_{j=2}^{d} z_j^2$
Cigar	unimodal	$z_1^2 + \sum_{j=2}^{d} (1000 z_j)^2$
Rastrigin	multimodal	$10n + \sum_{j=1}^{d} \left[z_j^2 - 10\cos(2\pi z_j)\right]$
Ackley	multimodal	$-20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{j=1}^{d} z_j^2}\right)$ $-\exp\left(\frac{1}{d}\sum_{j=1}^{d}\cos(2\pi z_j)\right) + 20 + e$
Weierstrass	multimodal	$\sum_{j=1}^{d}\sum_{k=0}^{20} 0.5^k \cos(2\pi 3^k (z_j + 0.5))$
Griewank	multimodal	$\sum_{j=1}^{d} \frac{z_j^2}{4000} - \prod_{j=1}^{d} \cos(\frac{z_j}{\sqrt{j}}) + 1$

Table 4.1: Standard benchmarks for black box optimization. Shown are the the 8 unimodal and 4 multimodal standard benchmark functions (for d dimensions) that are frequently used in competitions and for comparison purposes.

4.7.2 Performance on Benchmark Functions

NES was tested on the set of unimodal benchmark functions (see Table 4.1) with dimensions 5 and 15, using learning rate $\alpha = 1.0$, initial mutation matrix $\mathbf{\Sigma} = \mathbf{I}$ and $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and using a target fitness precision of 10^{-10}. Figure 4.5 shows the average performance over 200 runs for each benchmark function. Note that SharpR and ParabR are unbounded functions, which explains the abrupt drop-off.

For the experiments on the multimodal benchmark functions, the distance of the initial guess to the optimum was varied between 0.1 and 1000. Some runs were performed on dimension 2 with a target precision of 0.01, since here the focus was on avoiding local optima. Figure 4.6 shows, for all tested multimodal functions, the proportion of 200 runs where NES found the global optimum (as opposed to it getting stuck in a local extremum) conditioned on the distance from the initial guess to the optimum. Additionally, it shows a similar experiment where the dimensionality $2 \ldots 5$ was varied while keeping initial distance fixed to 1.

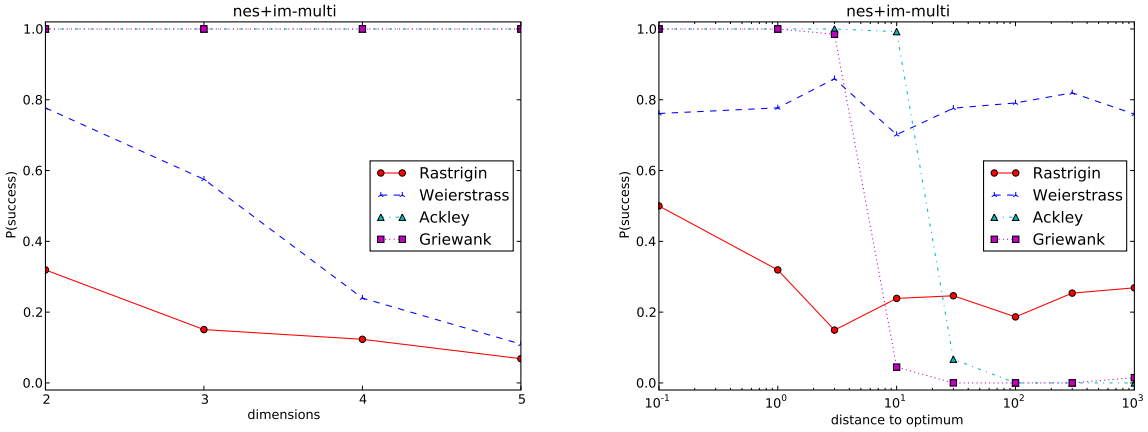

Figure 4.6: **Results for NES on the multimodal benchmark functions.** Left: results varied with the dimensionality with fixed initial distance of 1. Right: results varied with distance, with dimensionality 2. The data were recorded for 200 runs. Note that in the left graph, the performance curves of the Ackley and Griewank functions overlay each other.

4.7.3 Non-Markovian Double Pole Balancing

Used as the optimizer in phylogenetic reinforcement learning, NES was applied to the same four pole balancing benchmarks as described in Chapter 3. Fitness values were the lengths of the trials until the pole(s) fell over, and as policy representation the same recurrent LSTM network was chosen as used for RPGs, however, using only two memory cells in order to keep the number of weights low. Results on the pole balancing task are given in Table 4.2. As can be seen from Table 7.4 in Chapter 7, NES' results for the double-pole, non-Markovian case constitutes nearly the best performance

	Markov	non-Markov
1 pole	232 ± 65	538 ± 134
2 poles	1150 ± 426	1443 ± 521

Table 4.2: **NES results on the pole balancing tasks.** The table shows the results for NES on the pole balancing task, for the four possible cases investigated in this chapter: 1 pole Markov, 2 poles Markov, 1 pole non-Markov, and 2 poles non-Markov. The results show the mean and standard deviation of the number of evaluations until the success criterion was reached, that is, when a run lasts more than $100,000$ time steps (the equivalent of one half hour of simulated time). Results are computed over 200 runs. All runs achieved the objective within 100,000 trials.

on this difficult benchmark – significantly better than CMA-ES, and only slightly worse than the CoSyNE (Gomez et al., 2006) algorithm.

4.8 Conclusion and Future Work

NES is a novel alternative to conventional evolutionary algorithms, using a natural evolution gradient to adapt the population distribution. NES constitutes a competitive, theoretically well-founded and relatively simple method for phylogenetic reinforcement learning. Good results on standard benchmarks and the non-Markovian double pole balancing task affirm the promise of this research direction.

Unlike most evolutionary algorithms, NES boasts a relatively clean derivation from first principles. Using a full multinormal mutation distribution and fitness shaping, the NES algorithm is invariant under translation and rotation and under order-preserving transformations of the fitness function.

Comparing the empirical results to CMA-ES (see Chapter 7), considered by many to be the 'industry standard' of evolutionary computation (Hansen and Ostermeier, 2001), it is apparent that NES is competitive on both unimodal and multimodal benchmarks. These results, together with the results on the non-Markovian double pole balancing task, collectively show that NES can compete with state-of-the-art evolutionary algorithms.

Its theoretical relationship to the field of policy gradients (Williams, 1992; Peters and Schaal, 2008a), and in particular natural actor-critic (Peters and Schaal, 2008b), should be clear to any reader familiar with both fields. In recent work carried out independently from the work in this thesis, the similarities between policy gradient methods and evolution strategies have also been pointed out (Heidrich-Meisner and Igel, 2008), which suggests there might be fruitful future interaction between the two fields.

Future work will have to address problem of automatically determining good population sizes and dynamically adapting the learning rate, perhaps differentiated for mean and mutation matrix. Moreover, one could consider the possibility of combining this algorithm with other methods (e.g. estimation of distribution algorithms) to accelerate the adaptation of covariance matrices, improving performance on fitness landscapes where directions of ridges and valleys change abruptly. It may be worthwhile to investigate whether the use of heavy tail distributions (e.g. Cauchy) instead of the regular multinormal distribution is useful. Moreover, investigating the use of multimodal distributions could lessen the sensitivity of NES to local optima on more difficult phylogenetic tasks with complex policy representations. Last, the investigation of the theoretical relationship between CMA-ES and NES might constitute a promising research direction, as both algorithms, while entirely different in both approach and derivation, utilize a covariance matrix for producing samples.

Chapter 5

Fitness Expectation Maximization

In this chapter, *Fitness Expectation Maximization* (FEM) is presented, which is, like NES (Chapter 4) a novel method for performing 'black box' function optimization and phylogenetic reinforcement learning. FEM searches the fitness landscape of an objective function using an instantiation of the well-known Expectation Maximization (EM) algorithm, producing search points to match the sample distribution weighted according to higher expected fitness. FEM updates both candidate solution parameters and the search policy, which is represented as a multinormal distribution. Inheriting EM's stability and strong guarantees, the algorithm avoids overly greedy updates and early convergence. The method is both elegant and competitive, and performs especially well as a phylogenetic reinforcement learner on the challenging non-Markovian double pole balancing task.

5.1 Introduction

As explained in Chapter 4, real-valued 'black box' function optimization is one of the major topics in modern applied machine learning research (e.g., Spall et al., 2006). It concerns itself with optimizing the continuous parameters of an unknown (black box) objective fitness function, the exact analytical structure of which is assumed to be unknown or unspecified. Specific function measurements can be performed, however. The goal is to find a reasonably high-fitness candidate solution while keeping the number of function measurements limited. The black box optimization framework is crucial for many real-world domains, since often the precise structure of a problem is either not available to the engineer, or too expensive to model or simulate. Numerous realistic problems can be treated as real-valued black box function optimization problems. In order to illustrate the importance and prevalence of this general setup, one could point to a diverse set of tasks

such as the classic nozzle shape design problem (Klockgether and Schwefel, 1970), developing an Aibo robot gait controller (Kohl and Stone, 2004) and non-Markovian control (Gomez and Miikkulainen, 1999).

Now, since exhaustively searching the entire space of solution parameters is considered to be infeasible, and since one cannot assume to have access to a precise model of the fitness function, one is is forced to settle for trying to find a reasonably good solution that satisfies certain pre-specified constraints. This, inevitably, involves using a sufficiently intelligent heuristic approach, since in practice it is important to find the right domain-specific trade-off on issues such as convergence speed, expected quality of the solutions found and the algorithm's sensitivity to local suboptima on the fitness landscape.

One can postulate the similarity and actual equivalence of black box function optimization and one-step reinforcement learning. In the attempt to create a viable optimization technique based on reinforcement learning, one can fall back onto a classical goal of reinforcement learning, i.e., the search for a way to reduce the reinforcement learning problem to a supervised learning problem. In order to do so, it is profitable to re-evaluate the recent result in machine learning, that reinforcement learning can be reduced onto *reward-weighted regression* (Peters and Schaal, 2007) which is a novel algorithm derived from Dayan & Hinton's Expectation Maximization (EM) perspective on RL (Dayan and Hinton, 1997). It can be shown that this approach generalizes from reinforcement learning to fitness maximization to form Fitness Expectation Maximization (FEM).

This algorithm is tested on the standard set of unimodal and multimodal benchmark functions (see Chapter 7 for comparative results). Similar to NES, a defining feature of FEM is its adaptive *search policy*, which takes the form of a multinormal distribution that produces *correlated* search points in search space. Its covariance matrix makes the algorithm invariant across rotations in the search space, and enables the algorithm to fine-tune its search appropriately, resulting in arbitrarily high-precision solutions. Furthermore, using the stability properties of the EM algorithm, the algorithm seeks to avoid catastrophically greedy updates on the search policy, thus preventing premature convergence in many cases.

The organization of this chapter is as follows. The next section describes the derivation of the EM-based algorithm and the online instantiation of the algorithm. The ensuing experiments section shows initial results with a number of unimodal and multimodal benchmark problems. Furthermore, phylogenetic reinforcement learning results on the non-Markovian double pole balancing problem are presented. The last section discusses the algorithm's properties and offers some possible directions of future research.

5.2 Expectation Maximization for Black Box Function Optimization

At every point in time while running the algorithm, one wants to optimize the expected fitness $J = \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})]$ of the next batch, given the current batch of search samples. It is assumed that every batch g is generated by search policy $\pi^{(g)}$ parameterized by $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$, representing the current candidate solution $\mathbf{x}$ and covariance matrix $\mathbf{\Sigma}$.

In order to adjust parameters $\theta = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$ towards solutions with higher associated fitness, we *match* the search distribution to the actual sample points, but weighted by their utilities. Now let $f(\mathbf{z})$ be the fitness at a particular search point $\mathbf{z}$, and, utilizing the familiar multivariate normal distribution, let $\pi(\mathbf{z}|\theta) = \mathcal{N}(\mathbf{z}|\mathbf{x}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{n/2}|\mathbf{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{z}-\mathbf{x})^{\mathbf{T}}\mathbf{\Sigma}^{-1}(\mathbf{z}-\mathbf{x})\right]$ denote the probability density of search point $\mathbf{z}$ given the current search policy π. Expected fitness

$$J = \mathbf{E}_{\mathbf{z}}[f(\mathbf{z})] = \int \pi(\mathbf{z}|\theta) f(\mathbf{z}) d\mathbf{z}.$$

indicates the expected fitness over all possible sample points, weighted by their probabilities under policy π.

5.2.1 Optimizing Utility-transformed Fitness

While an objective function such as the above is sufficient in theory, algorithms which plainly optimize it have major disadvantages. They might be too aggressive when little experience – few sample points – is available, and converge prematurely to the best solution they have seen so far. On the opposite extreme, they might prove to be too passive and be biased by less fortunate experiences. Trading off such problems has been a long-standing challenge in reinforcement learning. However, in decision theory, such problems are surprisingly well-understood (Chernoff and Moses, 1987). In that framework it is common to introduce a so-called utility transformation $u(f(z))$ which has to fulfill the requirement that it scales monotonically with f, is semi-positive and integrates to a constant (note the similarity/near-equivalence to fitness shaping introduced in Chapter 4). Once a utility transformation is inserted, an expected utility function is obtained given by

$$J_u(\theta) = \int p(\mathbf{z}|\theta) u(f(\mathbf{z})) \, d\mathbf{z}.$$

The utility function $u(f)$ is an adjustment for the aggressiveness of the decision making algorithms, e.g., if it is concave, its attitude is risk-averse while if it is convex, it will be more likely to consider a fitness more than a coincidence. Obviously, it is of essential importance that this risk function

is properly set in accordance with the expected fitness landscape, and it should be regarded as a metaparameter of the algorithm.

Empirically it was found that ranking-based shaping functions work best for various problems, also because they circumvent the problem of extreme fitness values disproportionately distorting the estimation of the search distribution, making careful adaptation of the algorithm during search unnecessary even for problems with wildly fluctuating fitness. In this chapter, the simple rank-based utility transformation function also used as NES' shaping function is considered (see Chapter 4), the piecewise linear $u_k = u(f(\mathbf{z}_k)|f(\mathbf{z}_{k-1}), \ldots, f(\mathbf{z}_{k-N}))$ which first ranks all samples $k-N, \ldots, k$ based on fitness value, then assigns zero to the $N-m$ worst ones and assigns values linearly from $0 \ldots 1$ to the m best samples.

5.2.2 Fitness Expectation Maximization

Analogously as in (Peters and Schaal, 2007; Dayan and Hinton, 1997), one can establish the lower bound

$$\begin{aligned}
\log J_u(\theta) &= \log \int q(\mathbf{z}) \frac{p(\mathbf{z}|\theta) u(f(\mathbf{z}))}{q(\mathbf{z})} d\mathbf{z} \\
&\geq \int q(\mathbf{z}) \log \frac{p(\mathbf{z}|\theta) u(f(\mathbf{z}))}{q(\mathbf{z})} d\mathbf{z} \\
&= \int q(\mathbf{z}) \left[\log p(\mathbf{z}|\theta) + \log u(f(\mathbf{z})) - \log q(\mathbf{z})\right] d\mathbf{z} \\
&= \mathcal{F}(q, \theta),
\end{aligned}$$

due to Jensen's inequality with the additional constraint $0 = \int q(\mathbf{z}) d\mathbf{z} - 1$. This points us to the following EM algorithm:

Proposition 1. *An Expectation Maximization algorithm for optimizing both the expected utility as well as maximizing raw expected fitness is given by*

$$\textit{E-Step:} \quad q_{g+1}(\mathbf{z}) = \frac{p(\mathbf{z}|\theta) u(f(\mathbf{z}))}{\int p(\tilde{\mathbf{z}}|\theta) u(f(\tilde{\mathbf{z}})) d\tilde{\mathbf{z}}}, \tag{5.1}$$

$$\textit{M-Step Policy:} \quad \theta_{g+1} = \arg\max_{\theta} \int q_{g+1}(\mathbf{z}) \log p(\mathbf{z}|\theta) d\mathbf{z}. \tag{5.2}$$

Proof. The E-Step is given by

$$q = \text{argmax}_q \mathcal{F}(q, \theta)$$

while fulfilling the constraint $0 = \int q(\mathbf{z}) d\mathbf{z} - 1$. Thus, one has a Lagrangian

$$L(\lambda, q) = \mathcal{F}(q, \theta) - \lambda.$$

When differentiating $L(\lambda, q)$ with respect to q and setting the derivative to zero, one obtains

$$q^*(\mathbf{z}) = p(\mathbf{z}|\theta) u(f(\mathbf{z})) \exp(\lambda - 1).$$

Inserting this back into the Lagrangian obtaining the dual function

$$L(\lambda, q^*) = \int q^*(\mathbf{z}) d\mathbf{z} - \lambda.$$

Thus, setting $dL(\lambda, q^*)/d\lambda = 0$ yields

$$\lambda = 1 - \log \int p(\mathbf{z}|\theta) u(f(\mathbf{z})) \, d\mathbf{z},$$

and solving for q^* implies Equation (5.1). The M-steps compute

$$\theta_{g+1} = \text{argmax}_\theta \mathcal{F}(q_{g+1}, \theta).$$

□

In practice, when using a multinormal search distribution parameterized by $\theta^{(g)} = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$, the EM process comes down to simply fitting the samples in every batch to the Gaussian, weighted by the utilities.

5.3 Online Fitness Expectation Maximization

In order to speed up convergence, the algorithm can be executed *online*, that is, sample by sample, instead of batch by batch. The online version of the algorithm can yield superior performance since updates to the policy can be made at every sample instead of just once per batch. Crucial is that a *forget factor* α is now introduced to modulate the speed at which the search policy adapts to the current sample. Batch size N is now only used for utility ranking function u which ranks the current sample among the N last seen samples. The resulting FEM algorithm pseudocode can be found in Algorithm 5.1.

use shaping function u, batch size N, forget factor α
$k \leftarrow 1$
initialize search parameters $\theta^{(k)} = \langle \mathbf{x}, \mathbf{\Sigma} \rangle$
repeat
  draw one sample $\mathbf{z}_k \sim \pi(\mathbf{x}, \mathbf{\Sigma})$
  evaluate fitness $f(\mathbf{z}_k)$
  compute rank-based fitness shaping $u_k = u(f(\mathbf{z}_k)|f(\mathbf{z}_{k-1}), \ldots, f(\mathbf{z}_{k-N}))$
  $\mathbf{x} \leftarrow (1 - \alpha u_k)\mathbf{x} + \alpha u_k \mathbf{z}_k$
  $\mathbf{\Sigma} \leftarrow (1 - \alpha u_k)\mathbf{\Sigma} + \alpha u_k (\mathbf{x} - \mathbf{z}_k)(\mathbf{x} - \mathbf{z}_k)^\mathbf{T}$
  $k \leftarrow k + 1$
until stopping criterion is met

Algorithm 5.1: Fitness Expectation Maximization.

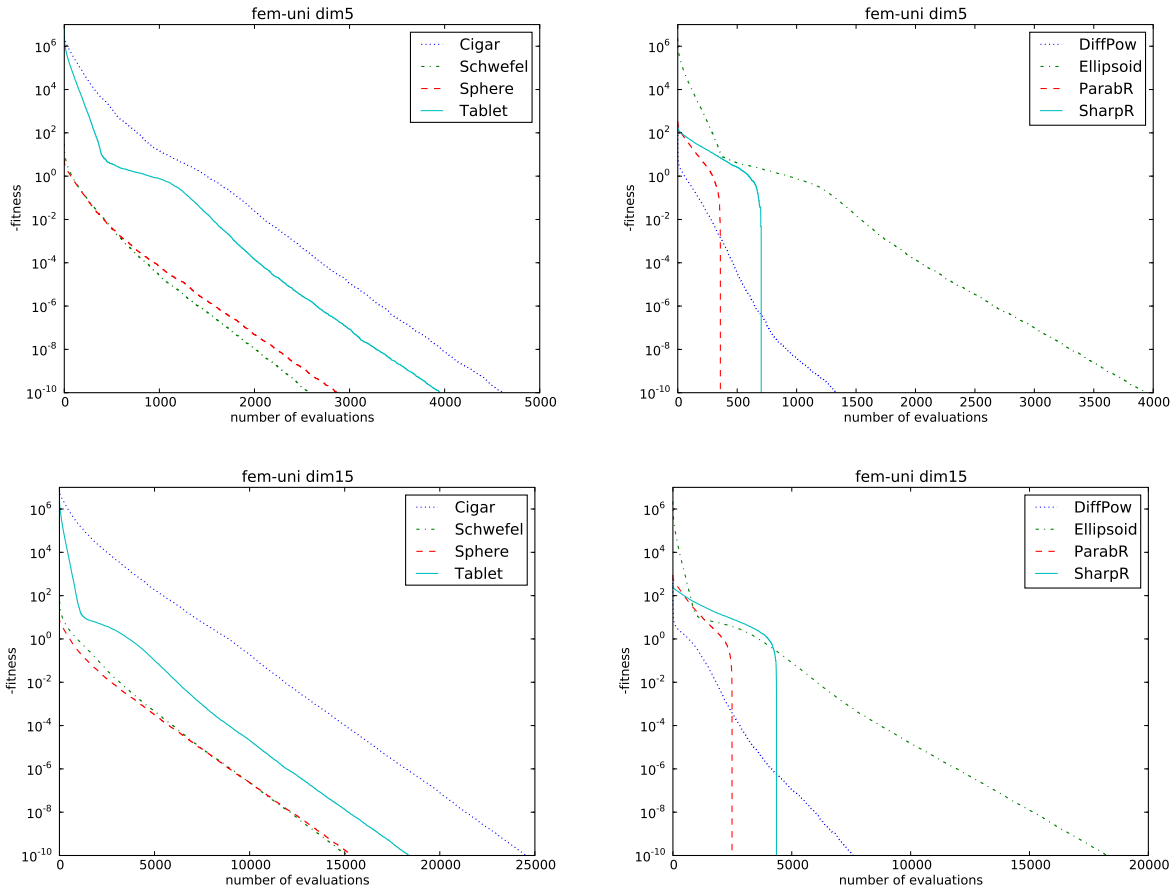

Figure 5.1: **Results for experiments on the unimodal benchmark functions.** Top: dimensionality 5, bottom: dimensionality 15.

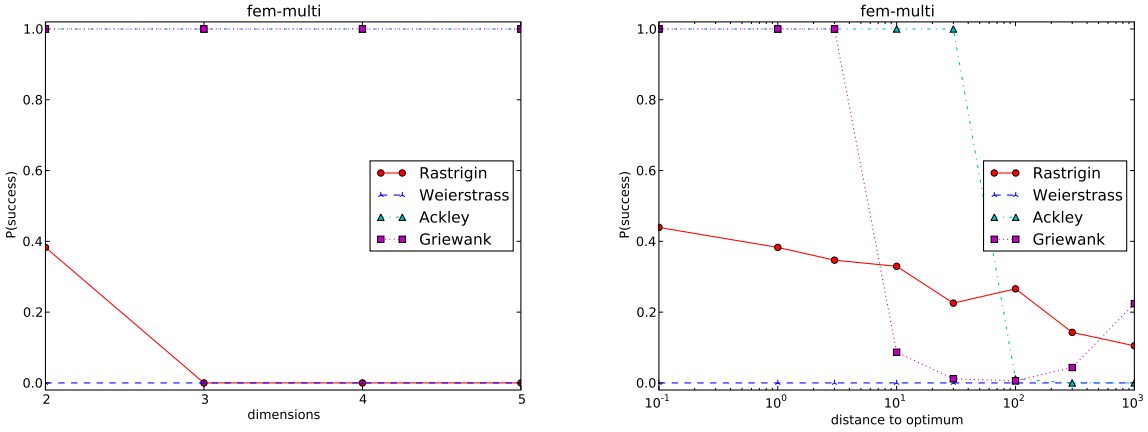

Figure 5.2: **Results for experiments on the multimodal benchmark functions.** Left: results varied with the dimensionality, right: results varied with distance. The data are recorded for 200 runs. Note that in the left graph, the performance curves of the Ackley and Griewank functions overlay each other. It can be seen that FEM fares especially badly on the Weierstrass and Rastrigin functions.

5.4 Experiments

5.4.1 Standard benchmark functions

In order to prevent potentially biased results, and to avoid trivial optima (e.g. at the origin), I again follow (Suganthan et al., 2005) as in Chapter 4 and consistently transform (by a combined rotation and translation) the functions' inputs in order to make the variables non-separable. This immediately renders many evolutionary methods virtually useless, since they cannot cope with correlated search directions, unlike FEM and NES.

The tunable parameters of the FEM algorithm are comprised of batch size N, the fitness shaping function u applied on the fitness function f and forget factor α. The parameters should be chosen by the expert to fit the expected ruggedness of the fitness landscape. The forget factor must be low enough such that it does not too quickly forget earlier successful search points. The shaping function must be chosen such that enough randomness is preserved in the search policy after every update, which entails including the less fit samples in utility attribution.

FEM was run on the set of unimodal benchmark functions with dimensions 5 and 15 using a target precision of 10^{-10}. Figure 5.1 shows the average performance over 200 runs on the unimodal functions. The parameter settings for dimensionality 5 were identical in all runs: $\alpha = 0.1$ and $N = 50$, m for selecting the shaping function's top m samples, was set at $m = 5$. The

	Markov		non-Markov	
1 pole	103 ± 37	[200/200]	235 ± 48	[200/200]
2 poles	1203 ± 632	[164/200]	1304 ± 801	[152/200]

Table 5.1: **FEM results on the pole balancing tasks.** The table shows the results for FEM on the pole balancing task, for the four possible cases investigated in this chapter: 1 pole Markov, 2 poles Markov, 1 pole non-Markov, and 2 poles non-Markov. The results show the mean and standard deviation of the number of evaluations until the success criterion was reached, that is, when a run lasts more than $100,000$ time steps. Results are computed over 200 runs, only counting the successful ones. The table shows the number of successful runs (out of 200) in brackets.

parameter settings for all runs in dimensionality 15 were: $\alpha = 0.02$, $N = 25$ and $m = 10$. All runs converged to the optimum.

On the multimodal benchmark functions experiments were performed while varying the distance of the initial guess to the optimum between 1 and 100. Those runs were performed on dimension 2 with a target precision of 0.01, since here the focus was on avoiding local maxima. Figure 5.2 shows, for all multimodal functions, the proportion of runs where FEM found the global optimum (as opposed to it getting stuck in a local suboptimum) conditioned on the distance from the initial guess to the optimum. The proportions are computed over 200 runs.

To summarize, the experiments on these standard black box optimization benchmarks indicate that FEM is competitive with other high-performance algorithms in black box optimization on the selected high-precision unimodal test functions. See Chapter 7 for a comparison to both NES and CMA-ES.

5.4.2 Non-Markovian Double Pole Balancing

As in Chapter 4, the FEM algorithm can be used to phylogenetically optimize the parameters of the controller of the cart in the pole balancing tasks. The controller is again implemented as a 2-cell LSTM recurrent neural network.

The algorithm's parameters were set as follows: piecewise linear shaping function with $m = 5$ top 5 selection, forget factor $\alpha = 0.05$ and batch size $N = 50$. A run was considered a *success* when the poles did not fall over for $100,000$ time steps. For non-Markovian double pole balancing, the results on a total of 200 runs are, on average, 1,304 evaluations until success (standard deviation: 801). Not included in these statistics are 48 out of 200 runs that did not reach success within the limit of 100,000 evaluations. Table 7.4 in Chapter 7 shows results of other premier algorithms applied to this task, including CMA-ES, RPGs and FEM. FEM outperforms all

other methods except CoSyNE – with the disadvantage that it prematurely converges (fails) in roughly 25% of runs. Since the algorithm both performs well and is derived from first principles (that is, it is less heuristic in nature than the other methods), the algorithm can be expected to do well on future real-world experiments with many deceptive local suboptima.

5.5 Conclusion and Future Work

In this chapter Fitness Expectation Maximization was introduced, which, like Natural Evolution Strategies, constitutes a novel algorithm to tackle the important class of real-valued black box function optimization problems with an application to phylogenetic reinforcement learning. Reframing black box optimization as a one-step reinforcement learning problem led to the development of a method similar in spirit to Expectation Maximization. Using a search policy which matches samples weighted by their utilities, FEM performs competitively on a standard benchmark set of unimodal and multimodal functions and non-Markovian double pole balancing control.

Fitness Expectation Maximization constitutes a simple, principled approach with a rather clean derivation from first principles. Its theoretical relationship to the field of reinforcement learning and in particular reward-weighted regression should be clear to any reader familiar with both fields. It can be anticipated that rephrasing the black box optimization problem as a reinforcement learning problem solvable by RL methods will spawn a whole series of additional new algorithms exploiting this connection.

Taking into account the good results on the pole balancing task, the experimental evidence indicates that FEM may become a serious competitor in the field of black box function optimization. It may be argued, though, that a method such as Fitness Expectation Maximization should be used only for problems with relatively small dimensionality, since the number of parameters in θ grows with the square of the parameters to be optimized. One needs a sufficient number of samples per batch before executing an update step. Alternatively, if the parameter space becomes infeasibly large, one could use only the variances instead of the entire covariance matrix.

Future work on FEM will include a systematic study that must determine whether it can be made to outperform other search methods consistently on other typical benchmarks and real-world tasks. One might suggest extending the algorithm from a single multinormal distribution as search policy representation to a mixture of Gaussians (which is a common procedure for 'vanilla' EM), thus reducing its sensitivity to local suboptima. Another pressing matter involves the theoretical analysis of the utility function, which should ideally be made to adapt automatically based on the data instead of tuned manually. Problematic to FEM's application is its need for parameter fine-tuning, in contrast to Natural Evolution Strategies which are trivial to

tune. Future research should investigate the possibility to automate this and derive a reasonable way of adjusting hyperparameters.

Chapter 6

Episodic Reinforcement Learning by Logistic Reward-Weighted Regression

It has been a long-standing goal in the adaptive control community to reduce the generically difficult, general reinforcement learning problem to simpler problems solvable by supervised learning. While this approach is today's standard for value function-based methods, fewer approaches are known that apply similar reductions to direct policy search methods. Recently, it has been shown that immediate RL problems can be solved by reward-weighted regression, and that the resulting algorithm is an Expectation Maximization algorithm with strong guarantees. In this chapter, this algorithm is extended from the one-step case to the episodic case and it is shown that it can be used in the context of LSTM recurrent neural networks. The resulting RNN training algorithm is equivalent to a weighted self-modeling supervised learning technique. In this chapter it is shown that this new reward-weighted logistic regression technique, used in conjunction with an RNN architecture, can solve simple standard benchmark POMDPs.

6.1 Introduction

In order to apply reinforcement learning to real-life scenarios it is often essential to deal with hidden and incomplete state information. While such problems have been discussed in the framework of partially observable Markov decision problems for a long time, this class of problems still lacks a satisfactory solution (Kaelbling et al., 1998). Most known methods to solve small POMDPs rely heavily on knowledge of the complete system, typically

in the form of a belief-estimator or filter. Without such important information, the problem is considered intractable even for linear systems, and is not distinguishable from non-Markovian problems (Aoki, 1967). As a result, both POMDPs and non-Markovian problems largely defy traditional value function based approaches.

While policy search based approaches can be applied even with incomplete state information (Baxter and Bartlett, 1999), they cannot yield an optimal solution unless the policy has an internal state (Wierstra et al., 2007). As the internal state only needs to represent the features of the belief state and not all of its components, a function approximator with an internal state would be the ideal representation of a policy, and a recurrent neural network constitutes one of the few choices. It offers an internal state estimator as a natural component and is well-suited for unstructured domains.

However, the training of recurrent neural networks in the context of reinforcement learning is non-trivial as traditional methods often do not easily transfer to function approximators, and even if they do transfer, the resulting methods such as policy gradient algorithms do no longer employ the advantages of the strong results obtained for supervised learning. As a way out of this dilemma, one falls back onto a classical goal of reinforcement learning, i.e., one searches for a way to reduce the reinforcement learning problem to a supervised learning problem where a multitude of methods exists for training recurrent neural networks. In order to do so, one can again use the recent result in machine learning (see Chapter 5), that reinforcement learning can be reduced onto *reward-weighted regression* (Peters and Schaal, 2007) which is a novel algorithm derived from Dayan & Hinton's Expectation Maximization perspective on RL (Dayan and Hinton, 1997). We show that this approach generalizes from immediate rewards to episodic reinforcement learning to form episodic Logistic Reward-Weighted Regression (LRWR).

As a result, a novel, general learning method is obtained for training memory-based policies in model-free partially observable environments, that is, a method that does not require prior knowledge of any of the dynamics of the problem setup. Using similar assumptions as in (Peters and Schaal, 2007), it can be shown that episodic reinforcement learning can be solved as a utility-weighted nonlinear logistic regression problem in this context, which greatly accelerates the speed of learning. A reinforcement learning setup is obtained which is well-suited for training LSTM recurrent neural networks, using the E-step of the algorithm to generate weightings for training the memory-capable LSTM network in the M-step. Intuitively, the network is trained to imitate or *self-model* its own actions, but with more successful episodes weighted more heavily than the unsuccessful ones, resulting in a convergence to an ever better policy. LRWR is evaluated on a number of standard POMDP benchmarks, and it is shown that this method provides a viable alternative to more traditional RL approaches.

6.2 Logistic Reward-Weighted Regression for Recurrent Neural Networks

Intuitively, it is clear that the general reinforcement learning problem is related to supervised learning problems as the policy should *match* previously taken motor commands such that episodes are more likely to be reproduced if they had a higher return. The network is trained to imitate or *self-model* its own actions, but with more successful episodes weighted more heavily than the unsuccessful ones, resulting in a convergence to an ever better policy. In this section, this approach is solidified based on (Peters and Schaal, 2007), and extended from the previous results and a single-step, immediate reward scenario to the general episodic case.

First, the basic assumptions are discussed. Subsequently, it is shown how a utility-weighted mean-squared error emerges from the general assumptions for an Expectation Maximization algorithm. Finally, the entire resulting algorithm is presented.

6.2.1 Optimizing Utility-transformed Returns

Let the return $R(H)$ be some measure of the total reward accrued during a history (e.g., $R(H)$ could be the average of the rewards for the average reward case or the future discounted sum for the discounted case), and let $p(H|\theta)$ be the probability of a history given policy-defining weights θ, then the quantity the algorithm should be optimizing is the expected return

$$J = \int_H p(H|\theta)R(H)dH. \tag{6.1}$$

This, in essence, indicates the expected return over all possible histories, weighted by their probabilities under policy π.

While a goal function such as found in Eq. (6.1) is sufficient in theory, algorithms which plainly optimize it have major disadvantages. They might be too aggressive when little experience is available, and converge prematurely to the best solution they have seen so far. On the opposite extreme, they might prove to be too passive and be biased by less fortunate experiences. Trading off such problems has been a long-standing challenge in reinforcement learning. As in Chapter 5, it is useful to introduce a so-called utility transformation $u_\tau(R)$ which has to fulfill the requirement that it scales monotonically with R, is semi-positive and integrates to a constant. Once a utility transformation is inserted, one obtains an expected utility function given by

$$J_u(\theta) = \int p(H|\theta)u_\tau(R(H))\,dH.$$

The utility function $u_\tau(R)$ is an adjustment for the aggressiveness of the decision making algorithms, e.g., if it is concave, its attitude is risk-averse

while if it is convex, it will be more likely to consider a reward more than a coincidence. Obviously, it is of essential importance that this risk function is not manually tweaked but rather chosen such that its parameters τ can be controlled adaptively in accordance with the learning algorithm.

In this chapter, one simple utility transformation function will be condidered, the soft-transform $u_\tau(r) = \tau \exp(\tau r)$ also used in (Peters and Schaal, 2007).

6.2.2 Expectation Maximization for Reinforcement Learning

Analogously as in the case for Fitness Expectation Maximization (Chapter 5), one can establish the lower bound

$$\begin{aligned}
\log J_u(\theta) &= \log \int q(H) \frac{p(H|\theta) u_\tau(R(H))}{q(H)} dH \\
&\geq \int q(H) \log \frac{p(H|\theta) u_\tau(R(H))}{q(H)} dH \\
&= \int q(H) \left[\log p(H|\theta) + \log u_\tau(R(H)) - \log q(H)\right] dH \\
&= \mathcal{F}(q, \theta, \tau),
\end{aligned}$$

due to Jensen's inequality with the additional constraint $0 = \int q(H) dH - 1$. This points one to the following EM algorithm:

Proposition 2. *An Expectation Maximization algorithm for optimizing both the expected utility as well as the reward-shaping is given by*

$$\textit{E-Step:} \quad q_{k+1}(H) = \frac{p(H|\theta) u_\tau(R(H))}{\int p(\tilde{H}|\theta) u_\tau\left(R(\tilde{H})\right) d\tilde{H}}, \tag{6.2}$$

$$\textit{M-Step Policy:} \quad \theta_{k+1} = \arg\max_\theta \int q_{k+1}(H) \log p(H|\theta) dH, \tag{6.3}$$

$$\textit{M-Step Utility Adaptation:} \quad \tau_{k+1} = \arg\max_\tau \int q_{k+1}(H) \log u_\tau(R(H)) \, dH. \tag{6.4}$$

Proof. The E-Step is given by

$$q = \text{argmax}_q \mathcal{F}(q, \theta, \tau)$$

while fulfilling the constraint $0 = \int q(H) dH - 1$. Thus, one has a Lagrangian

$$L(\lambda, q) = \mathcal{F}(q, \theta, \tau) - \lambda.$$

When differentiating $L(\lambda, q)$ with respect to q and setting the derivative to zero, one obtains

$$q^*(H) = p(H|\theta) u_\tau(R(H)) \exp(\lambda - 1).$$

Inserting this back into the Lagrangian yields the dual function

$$L\left(\lambda, q^*\right) = \int q^*(H)dH - \lambda.$$

Thus, by setting $dL\left(\lambda, q^*\right)/d\lambda = 0$, one obtains

$$\lambda = 1 - \log \int p(H|\theta)u_\tau\left(R(H)\right)dH,$$

and solving for q^* implies Equation (6.2). The M-steps compute

$$\left[\theta_{k+1}, \tau_{k+1}\right]^T = \text{argmax}_{\theta,\tau}\mathcal{F}\left(q_{k+1}, \theta, \tau\right).$$

One can maximize

$$\mathcal{F}\left(q_{k+1}, \theta, \tau\right)$$

for θ, τ independently, which yields Equations (6.3,6.4). □

6.2.3 The Utility-Weighted Error Function for the Episodic Case

For every reinforcement learning problem, one needs to establish the cost function $\mathcal{F}\left(q_{k+1}, \theta, \tau\right)$ and maximize it in order to derive a policy. For episodic reinforcement learning, first the general settings need to be recapped. One can denote the probabilities $p(H|\theta)$ of histories H by

$$p(H|\theta) = p(\langle o_0, g_0\rangle) \prod_{t=1}^{T(H)-1} p(\langle o_t, g_t\rangle|h_{t-1}, a_{t-1}, g_{1:t})\pi(a_{t-1}|h_{t-1})$$

which are dependent on an unknown initial state and observation distribution $p(\langle o_0, g_0\rangle)$, and on unknown state transition function. However, the policy $\pi(a_t|h_t)$ with parameters θ is known, where h_t denotes the history which is collapsed into the hidden state of the network.

It is clear that the expectation step has to follow by simply replacing the expectations by sample averages. Thus, one has

$$q_{k+1}(H_i) = \frac{u_\tau\left(R(H_i)\right)}{\sum_{j=1}^N u_\tau\left(R(H_j)\right)}$$

as E-step. Here, $U_N = \sum_{j=1}^N u_\tau\left(R(H_j)\right)$ is defined as the summed utility of all N histories.

The maximization or M-step of the algorithm requires optimizing θ such that $\mathcal{F}\left(q_{k+1}, \theta, \tau\right)$ is maximized. In order to optimize θ one must realize that the probability of a particular history is simply the product of all actions and

observations given subhistories (Eq. 6.2.3). Taking the log of this expression transforms this large product into a sum

$$\log p(H|\theta) = (\text{const}) + \sum_{t=0}^{T(H)-1} \log \pi(a_t|h_t)$$

where most parts are not affected by policy-defining parameters θ, i.e., are constant, since they are solely determined by the environment. Thus, when optimizing θ, this constant can be ignored, and one can purely focus on the outputs of the policy, optimizing the expression

$$\mathcal{F}\left(q_{k+1}, \theta, \tau\right) \propto \sum_{i=1}^{N} q_{k+1}(H_i) \log p(H_i|\theta) = \sum_{i=1}^{N} \frac{u_\tau\left(R(H_i)\right)}{U_N} \sum_{t=0}^{T(i)-1} \log \pi(a_t^i|h_t^i),$$

where a_t^i denotes an action from the complete history i at time t and h_t^i denotes the collapsed history i up to time-step t.

6.2.4 Logistic Reward-Weighted Regression for LSTMs

As it seems prudent to use recurrent neural networks as policies while avoiding the vanishing gradient problem, an LSTM recurrent neural network is again used as policy $\pi(a_t|h_t)$ with parameters θ. Here, one still conditions on the history h_t of the sequence up to time step t as it is collapsed into the hidden state of the network. A standard LSTM architecture is used where the discrete actions are drawn from a softmax output layer, that is:

$$\pi(a_t|h_t) = \frac{\exp(v(a_t, h_t))}{\sum_{a=1}^{A} \exp(v(a, h_t))}$$

for all A actions where the net inputs of the neurons are $v(a_t, h_t)$. We can compute the cost function $\mathcal{F}\left(q_{k+1}, \theta, \tau\right)$ for this policy and obtain the utility-weighted conditional likelihood function

$$\mathcal{F}\left(q_{k+1}, \theta, \tau\right) = \sum_{i=1}^{N} \frac{u_\tau\left(R(H_i)\right)}{U_N} \sum_{t=0}^{T(i)-1} \left(a_t^i v(a_t^i, h_t^i) - \log \sum_{a=1}^{A} \exp(v(a, h_t)) \right).$$

This optimization problem is equivalent to a weighted version of logistic regression (Kleinbaum et al., 2002). As $v(a, h_t)$ is linear in the parameters of the output layer, these can be optimized directly. The hidden state related parameters of $v(a, h_t)$ can be optimized using supervised learning with back-propagation through time (BPTT) of the LSTM architecture. Both linear and nonlinear logistic regression problems cannot be solved in one single shot. Nevertheless, it is straightforward to show that the second-order expansion simply yields a linear regression problem which is equivalent to a

Newton-Rapheson step on the utility-weighted conditional likelihood. As a result, one has an approximate regression problem

$$\mathcal{F}\left(q_{k+1}, \theta, \tau\right) \approx \sum_{i=1}^{N} \frac{u_\tau\left(R(H_i)\right)}{U_N} \sum_{t=0}^{T(i)-1} \left(a_t^i - \pi(a_t^i|h_t^i)\right)^2,$$

which is exactly the utility-weighted squared error. Optimizing this expression by gradient descent allows one to use standard methods for determining the optimum. Nevertheless, there is a large difference in comparison to regular reward-weighted regression where the regression step can only be performed once – instead we can perform multiple BPTT training steps until convergence. In order to prevent overfitting the common technique of early stopping is used, assigning the sample histories to two separate batches for training and validation. While this supervised training scheme requires a relatively large demand in computation per sample history, it also reduces the number of episodes necessary for the policy to converge. Lastly, the update of τ optimizing Eq. 6.2 follows (Peters and Schaal, 2007) as $\tau_{k+1} = \frac{\sum_{i=1}^{N} u_\tau(R(H_i))}{\sum_{i=1}^{N} u_\tau(R(H_i))R(H_i)}$. The complete algorithm pseudocode is displayed in Algorithm 6.1.

Initialize θ, training batch size N, $\tau = 1$, $k = 1$.
repeat
  for $i = 1 \ldots N$ **do**
    Sample episode $H_i = \langle o_0, a_0, o_1, a_1, \ldots, o_{T-2}, a_{T-2}, o_{T-1} \rangle$ using policy π.
    Evaluate return for $t = 0 : R(H_i)$.
    Compute utility of H_i as $u_\tau(R(H_i))$.
  Train weights θ of policy π until convergence with BPTT to minimize
  $\mathcal{F}\left(q_{k+1}, \theta, \tau\right) \approx \sum_{i=1}^{N} \frac{u_\tau\left(R(H_i)\right)}{U_N} \sum_{t=0}^{T(i)-1} \left(a_t^i - \pi(a_t^i|h_t^i)\right)^2,$
  using validation sample histories for early stopping.
  Recompute $\tau \leftarrow \frac{\sum_{i=1}^{N} u_\tau(R(H_i))}{\sum_{i=1}^{N} u_\tau(R(H_i))R(H_i)}$.
  $k \leftarrow k + 1$
until stopping criterion is met

Algorithm 6.1: Episodic Logistic Reward-Weighted Regression.

6.3 Experiments

In order to test the LRWR algorithm, empirical experiments were performed on small discrete POMDP benchmarks commonly used in the literature. McCallum's CheeseMaze, the Tiger problem, Chrisman's Shuttle Docking benchmark and the 4x3Maze are all classic POMDP problems which range

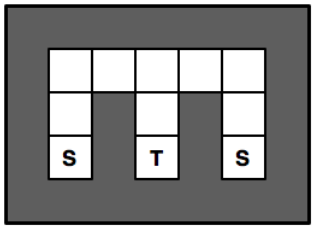

Figure 6.1: **The CheeseMaze POMDP benchmark.** The agent's observation consists of a four-dimensional vector indicating walls or the absence of walls in directions N, E, S and W. The agent starts every episode in either one starting location labeled 'S', and the goal is to reach target 'T' (the 'cheese') where a reward of 1.0 is given (thus ending the episode), using actions N, E, S and W. Bumping into a wall is punished with -1.0.

from 2 to 11 states, with 2 to 7 observations, with varying degress of stochasticity regarding action stochasticity and observation noise. Additionally, experiments were again performed on the T-maze (see Chapter 3) with limited hallway lengths 3, 5 and 7.

6.3.1 McCallum's CheeseMaze

This classical partially observable maze task was one of the first problems that model-free POMDP algorithms were applied to (McCallum, 1993).
States: The agent's unobservable state is its location in the grid (see Figure 6.1). Starting states are randomly chosen from the two states denoted with 'S', and the goal state that ends the episode is indicated with 'T'.
Actions: Actions are N, E, S, W. Bumping into a wall leaves the agent in its place.
Observations: The agent observes 'wall' or 'no-wall' in directions N, E, S and W from its locations, encoded with 1.0 and -1.0, respectively.
Rewards: The agent is rewarded 1.0 for reaching goal state 'T', and punished -1.0 if its bumps into a wall. A discount factor of $\gamma = 0.7$ is used.

6.3.2 The Tiger problem

The Tiger problem (Littman et al., 1995) was designed to study the information gain problem in POMDPs. That is, some actions do not change the state of the environment but cause the agent to gain information which it can use to act better under uncertainty. The setup is as follows. There are two doors, and behind one is a penalty (the tiger: penalty -100) and behind the other is a reward ($+10$). You can open either one of the doors, or decide to listen. If you listen and the tiger is behind the left door, then

with probability 85% you hear the tiger on the left and on the right with probability 15%, and vice versa. Listening causes a small penalty (-1). The problem is iterated – after opening the door, the setup is reset and the tiger replaced randomly behind one of the two doors.
States: There are two possible situations: the tiger is either behind the left or behind the right door.
Actions: There are three possible actions: open left, open right, and listen. Note that listening gives the correct observation only 85% percent of the time.
Observations: The tiger is observed behind either the left or the right door, which is encoded using two inputs as $(1, -1)$ or $(-1, 1)$.
Rewards: Opening the door with the tiger causes a penalty of -100. Opening the correct door yields a reward of 10. Listening costs -1. A discount of $\gamma = 0.75$ is used.

6.3.3 Chrisman's Shuttle Docking Benchmark

Chrisman's Shuttle Docking benchmark – see (Chrisman, 1992) for a more complete description – involves a spaceship that moves cargo between two space stations. The task is small but highly stochastic, with unreliable actions and noisy observations. The agent's (underlying, unobservable) state indicates whether the agent is docked to either one space station, whether it is in between the two stations or in front of a station. Additionally, it indicates which space station the ship is facing, and what the target station is (it needs to load/unload the stations alternately). Rewards are given for successfully alternately moving between stations, and the objective is to yield a high future discounted reward this way, over 1000 time steps.
States: The problem can be encoded using 8 states, taking the symmetry of two space stations into account. The initial state is docked.
Actions: There are three actions, GoForward, TurnAround and Backup. TurnAround causes the ship to turn around to face the other space ship. GoForward either deattaches the ship from the dock, launches into the free space between the stations, or causes the ship to bump into the station if it is right in front, yielding a penalty of -3. The backup action, which must be done facing the other station, is used for docking, but fails with probability 30% (in which case it moves forward instead).
Observations: The observations comprise whether it sees a space station or empty space, and whether it it docked or not. There is a probability 30% that the indicator station/empty space is toggled. If it sees a station, it also observes an 'accepting deliveries' sign whether this is the current target station. It also observes whether it received a reward (for successfully docking to the correct station) in the last time step. These five possible observations are encoded using five inputs, one of which is 1.0, the others 0.0

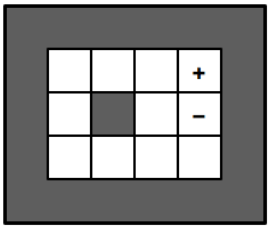

Figure 6.2: **Parr and Russell's 4x3Maze.** This deterministic task consists of a partially observable 4x3 gridworld where the agent can only observe whether there are walls left or right of it. The '+' goal state indicates a reward 1.0 and ends the episode, the '–' state gives a penalty of −1.0. The agent starts every episode in a random location.

Rewards: A penalty −3 is given for bumping into a station, and +10 is provided for successfully docking. A discounting factor of $\gamma = 0.9$ is used.

6.3.4 Parr and Russell's 4x3Maze

This stochastic 4x3Maze task (Parr and Russell, 1995) comprises a 4 by 3 grid world with a single obstacle and two reinforcement states indicated '+' and '–' in Figure 6.2. The task is stochastic since actions are unreliable.
States: There are 11 states in this gridworld. The starting state is chosen randomly each episode.
Actions: Four actions are available: N, E, S and W. Executing an action leads to the desired movement 80% of the time. 20% of the actions leads the agent to slide sideways in one of the two directions perpendicular to the intended one, with equal probability. Bumping into walls does not, however, lead to punishment.
Observations: The agent only observes whether there are walls to the east and west of it. Additionally, it can observe the '+' and '–' states, which leads to six possible observations which are encoded using six inputs, one of which is 1.0, the others 0.0.
Rewards: The agent receives a +1.0 reward at the '+' state (which ends the episode) and a punishment −1.0 at the '–' state. No discounting is used (i.e., $\gamma = 1$), but every state other than the '+' or '–' states yield −0.04, as such inducing the agent to reach the goal as fast as possible.

6.3.5 Bakker's T-maze

The last experiment that LRWR was tested on was the T-maze (see Chapter 3 for a more complete description), which was designed to test an RL algorithm's ability to correlate events far apart in history (Bakker, 2002a).

It involves having to *learn* to remember the observation from the first time step until the episode ends. Its difficulty, depending on corridor lengths, can be adjusted. LRWR's performance was investigated on the T-maze, using corridor lengths 3, 5 and 7.

6.3.6 Settings and Results

Task	Hand	Random	Normal	Greedy
Tiger	0.67	-36	-6.5 ± 4.9	-5.7 ± 7.4
Shuttle Docking	1.69	-0.31	0.709 ± 0.059	0.0 ± 0.0
4x3Maze	0.27	0.056	0.240 ± 0.085	0.246 ± 0.092
CheeseMaze	0.257	0.072	0.177 ± 0.032	0.212 ± 0.057
T-Maze3	1.0	0.166	0.917 ± 0.043	1.0 ± 0.0
T-Maze5	0.667	0.046	0.615 ± 0.032	0.662 ± 0.021
T-Maze7	0.5	0.002	0.463 ± 0.008	0.484 ± 0.090

Table 6.1: **Logistic Reward-Weighted Regression results.** This table shows results averaged over 25 runs. Displayed are the average rewards and standard deviations obtained for the trained policy (Normal) after 100 EM steps, its greedy variant (Greedy) which always takes the learned policy's most likely action, and both a randomized policy (Random) and a hand-crafted (Hand) policy as a reference. Shown results include statistics for T-mazes with corridor lengths 3, 5 and 7. Note that for the Shuttle Docking benchmark only the results for the successful runs are shown. Shown results were calculated using the average of 200 roll-outs.

The policy was, as elsewhere in this thesis, represented as an LSTM network, with input layer size dependent on the observational encoding, a hidden layer containing 2 LSTM memory cells, and a softmax output layer with size dependent on the number of actions applicable to the environment. One tunable parameter, batch size N, was always set to 30, except for the CheeseMaze and the T-maze, where it was set to 75. One third of all batch sample episodes was used for validation in an early stopping scheme to prevent overfitting.

Every batch was trained until convergence using early stopping – for Tiger, the Shuttle Docking and the 4x3Maze were trained on 20 episodes and validated on the remaining 10 episodes, while for the CheeseMaze and the T-mazes 50 episodes were used for training using 25 episodes for validation. The LSTM network was initialized with weights uniformly distributed between -0.1 and 0.1. It was trained using the standard BPTT procedure for LSTM (Graves and Schmidhuber, 2005) with learning rate 0.002 and momentum 0.95, while sample reweightings were used that are proportional to the self-adapting soft-transform $u_\tau(r) = \tau \exp(\tau r)$, but normalized such

that the maximal weighting in every batch was always 1.0. All experiments consisted of 100 consecutive EM-steps, and were repeated 25 times to gain some statistics.

The results are shown in Table 6.1. One can see that all problems converged quickly to a good solution, except for the Shuttle Docking benchmark where 13 out of 25 runs failed to converge to an acceptable solution (i.e. the shuttle does not dock at all). This might be due to the problem's inherently stochastic nature, which possibly induces the algorithm to converge prematurely. The T-maze results for LRWR are significantly less impressive than for RPGs found in Chapter 3 and the algorithm presented in (Bakker, 2002b), where corridor lengths of 90 and 70 are reached, respectively. However, it can be argued that the solving T-maze length 7 in less than 100 EM steps with batch size 75 constitutes a reasonable result.

The results were obtained without much fine tuning. This encourages one to expect that extensions of the approach will produce a rather general POMDP solver.

6.4 Conclusion and Future Work

In this chapter, a novel, surprisingly simple EM-derived episodic reinforcement learning algorithm was presented that learns from temporally delayed rewards. The method can learn to deal with partially observable environments by using LSTM, the parameters of which are updated using utility-weighted logistic regression as supervised training method. The successful application of this algorithm to a number of small POMDP benchmarks shows that reward-weighted regression might constitute a promising research direction for episodic reinforcement learning in non-Markovian settings. Nevertheless, the relatively high computational cost of supervised learning, combined with results that are significantly less impressive than those achieved by more mature approaches such as Recurrent Policy Gradients (see Chapters 3 and 7), makes one realize that there is much room for future improvements in this domain.

Future work should include the application of history-dependent baselines analogous to those used in Chapter 3 for the Recurrent Policy Gradient algorithm, in order to alleviate some of the credit assignment problem. Moreover, extensions to continuous actions can be considered, using a supervised regression method on both the means and variances of the actions. Extensions could also include the properly re-weighted reuse of information from previous batches, resetting network weights for every EM step, and various improvements to the supervised learning scheme. Future work can involve the investigation of the possibility of the use of value-functions and other time-specific reward-attributions to alleviate estimation variance problems, by shifting responsibilities from entire sequences to single actions.

Chapter 7

Comparisons

This chapter summarizes some comparative results for the algorithms presented in this thesis. For black box optimization, it compares NES and FEM with CMA-ES (Hansen and Ostermeier, 2001) on the standard unimodal and multimodal benchmark functions. Furthermore, all algorithms are tested on a suite of small discrete POMDP tasks, and RPGs, NES and FEM are discussed in the context of the hard non-Markovian double pole balancing benchmark.

7.1 Black Box Optimization: Unimodal

In this section I present a comparison between, on the one hand, FEM and NES, and on the other hand CMA-ES (Hansen and Ostermeier, 2001), the industry standard in black box optimization. Figures 7.1 and 7.2 show the results plotted for the standard unimodal benchmark functions. These standard benchmark functions, that are regularly used in competitions in evolutionary computation, are chosen to test black box optimization algorithms' behavior on various problem characteristics, such as convergence speed and sensitivity to ill-shaped fitness landscapes. Generally, it is found that algorithms that do well on this suite of problems, do well on realistic problem settings (Beyer, 1996).

On low dimensionalities (dimension 5), FEM outperforms NES (see Chapters 4 and 5). But on higher dimensions, such as the displayed 15-dimensional benchmark functions, results are less clear-cut. CMA-ES converges faster than FEM and NES on the Sphere function, but on Tablet, DiffPow and Ellipsoid, which are more ill-shaped and therefore harder, NES clearly outperforms CMA-ES and FEM. Note that follow-up work that we published on NES (Yi et al., 2009a,b) shows how a more computationally efficient implementation of NES, Efficient NES, scales up well with dimensionalities up to 100.

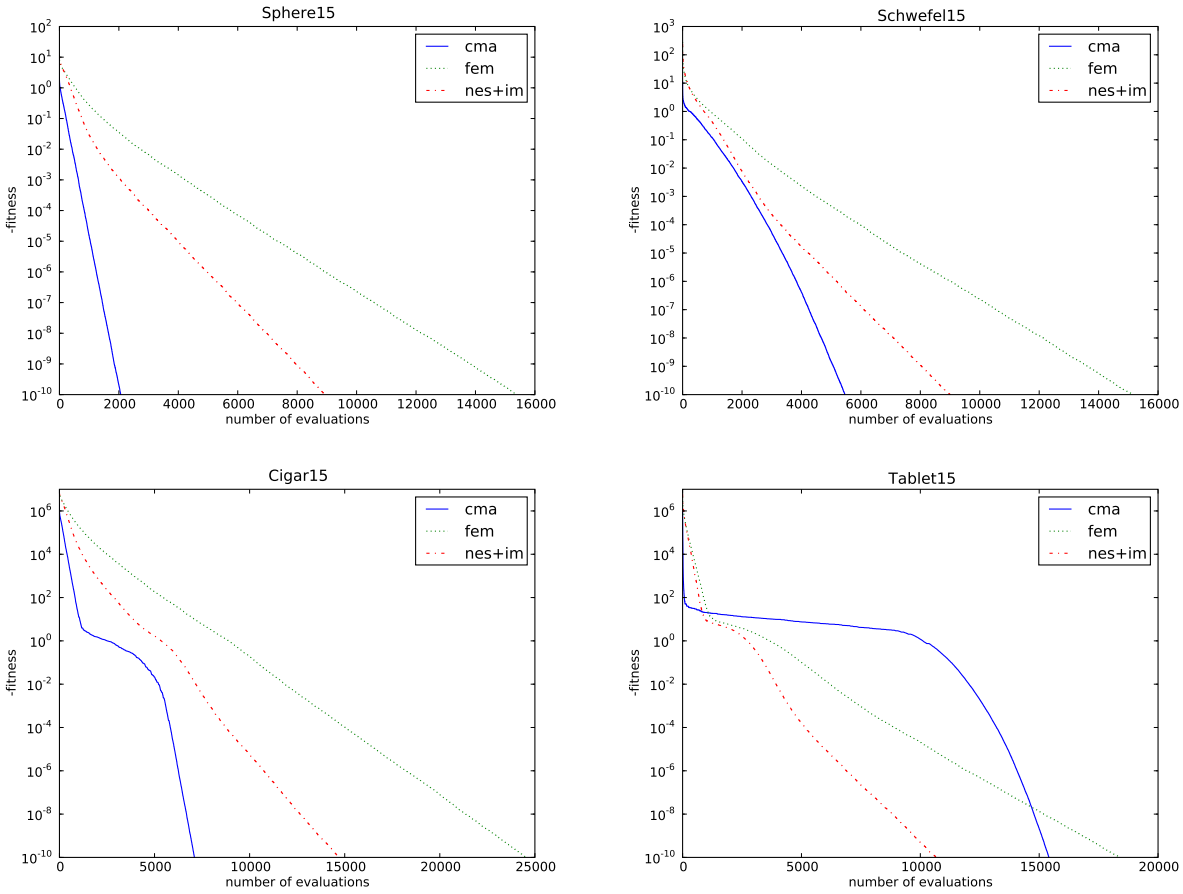

Figure 7.1: **Comparative results for the standard unimodal benchmark functions on FEM, NES and CMA-ES, dimensionality 15.** This figure shows results for the Sphere, the Schwefel function, the Cigar function and the Tablet function, calculated over 200 runs.

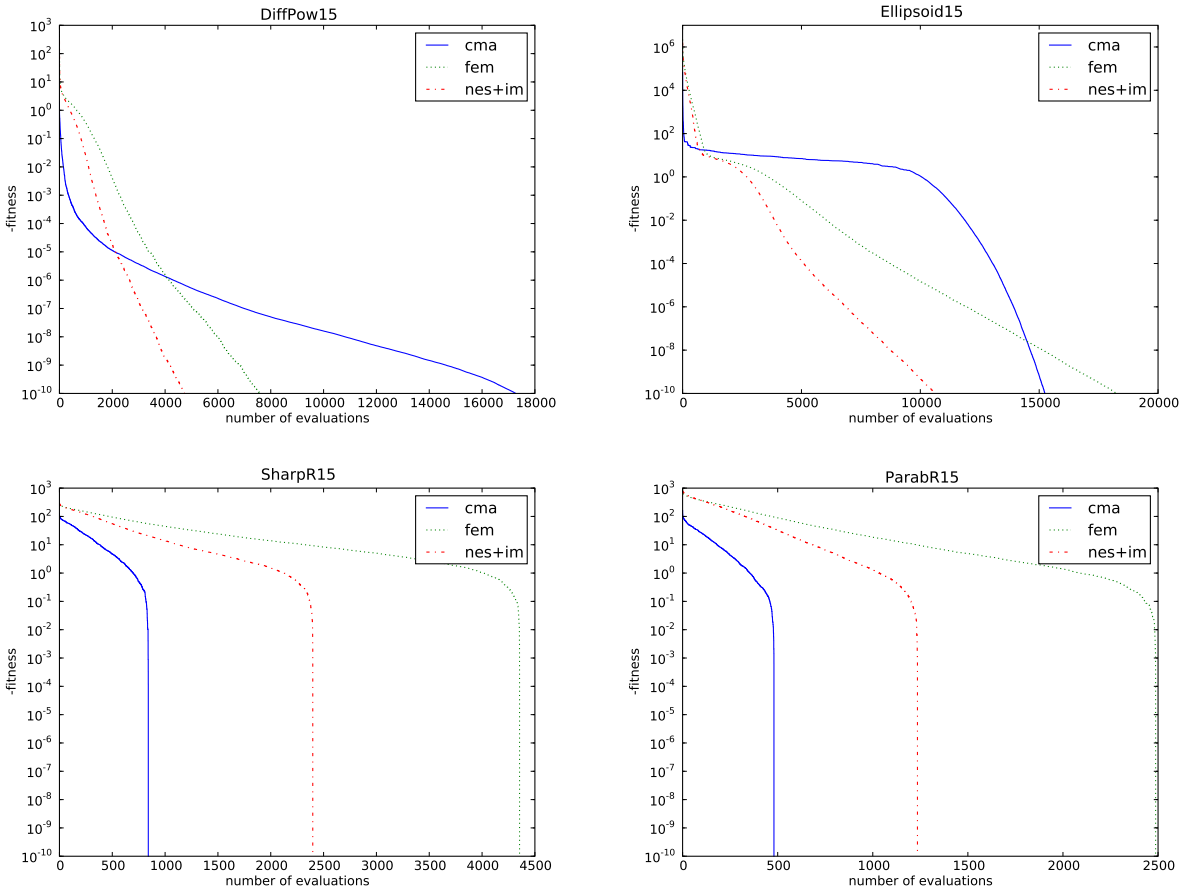

Figure 7.2: **Comparative results for standard unimodal benchmark functions on FEM, NES and CMA-ES, dimensionality 15.** Shown are the DiffPow, Ellipsoid, SharpR and ParabR benchmarks, calculated over 200 runs.

7.2 Black Box Optimization: Multimodal

Figure 7.3 shows results for NES, FEM and CMA-ES on multimodal functions, and displays how the performance of these algorithms varies with both problem dimensionality and the initial distance to the optimum. The left figures show how the proportion of successful runs scales with increased initial distances to the global optimum when the dimensionality is kept fixed at 2. The right figures show how these proportions vary with problem dimensionality with fixed initial distance of 1.0. Note that the number of deceptive local suboptima grows rapidly with dimensionality. The top row show results on NES, the middle row shows results on FEM, the bottom row shows results on CMA-ES, the main algorithm used in the black box optimization community on continuous optimization. Note that the behavior of CMA-ES and NES is quite comparable, despite their very different algorithm structures, while FEM's performance quickly deteriorates with increased dimensionality.

7.3 Reinforcement Learning Comparisons

This section provides a succinct comparative overview of the empirical results on reinforcement learning in partially observable environments for the four algorithms introduced in this thesis. Table 7.1 provides an overview of the properties of the discrete POMDPs tackled in this work, and Table 7.2 shows the comparative results for RPGs, NES, FEM and LRWR on the smaller discrete POMDPs.

To obtain results for NES, FEM and RPGs, an equal number of episodes were allowed to all algorithms as for LRWR's results in Chapter 6. FEM and NES were used with their standard settings equal those of the double pole balancing benchmark, except that the batch size (window size) was set to equal that used for LRWR. All policies used the same 2-cell LSTM architecture. Shown results were computed using 200 policy roll-outs. Fitness values for NES and FEM were taken to be simply the episodes' returns. Note that for NES and FEM I did not aggregate multiple roll-outs in order to obtain a reliable fitness estimate. In practice, aggregating returns did not improve performance given the limited number of evaluations allowed in this comparison.

McCallum's CheeseMaze: This maze was one of the first tasks that were used to demonstrate the abilities of model-free POMDP algorithms. On this small, deterministic POMDP task, Recurrent Policy Gradients achieved optimal performance. Both ontogenetic algorithms RPGs and LRWR achieved better results than phylogenetic FEM and NES, which is surprising given the conjecture stated in Chapter 2 that suggests that phylogenetic methods should tend to outperform ontogenetic methods on small, deterministic

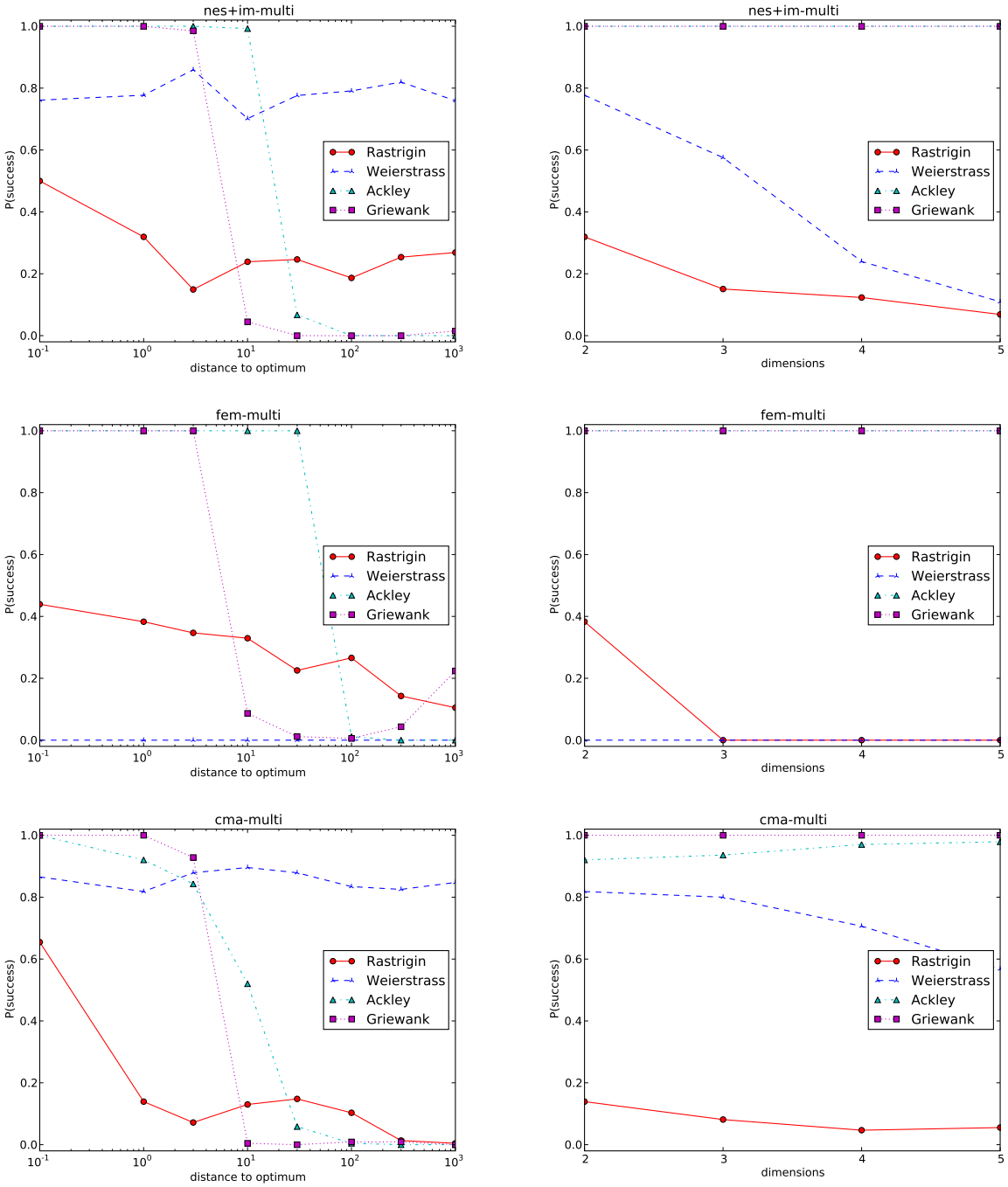

Figure 7.3: **Comparative results for standard multimodal benchmark functions on FEM, NES and CMA-ES.**, calculated over 200 runs.

Task	$\|S\|$	$\|\mathcal{A}\|$	$\|O\|$	observation noise	action noise	long-term dep.
Shuttle	8	3	5	yes	yes	no
CheeseMaze	11	4	7	no	no	no
Tiger	2	3	2	yes	no	no
4x3Maze	11	4	6	no	yes	no
T-maze7	18	4	6	no	no	yes
89-state maze	89	5	17	yes	yes	yes

Table 7.1: Discrete POMDP properties. This table provides an overview of the characteristics of the discrete POMDP mazes used in this thesis. $|S|$ indicates the number of states, $|\mathcal{A}|$ indicates the number of actions and $|O|$ denotes the number of possible observations. Also shown is whether the POMDP has noise in the observations and actions. 'Long-term dep.' is a subjective measure that is introduced here that indicates whether, in order to achieve a policy reasonably close to optimal, one needs to remember events from more than 5 time steps ago.

tasks.

The Tiger Problem: The two-state Tiger problem was introduced to check whether POMDP algorithms have the ability to choose actions for their information gain. Its actions are deterministic, but the listening action has a substantial probability of producing the wrong observation. RPGs achieve near-optimal performance, and again ontogenetic LRWR outperforms both NES and FEM substantially, which corroborates the hypothesis that stochasticity hampers the performance of phylogenetic algorithms. Most likely, in order to obtain optimal results using either NES or FEM, unreasonably large batch sizes would be required.

Chrisman's Shuttle Docking Benchmark: This task is small yet highly stochastic, both with respect to its (noisy) observations and its actions. RPGs and LRWR again outperform NES and FEM, though LRWR fails to converge to an acceptable policy about half the time, while the other three algorithms always produce reasonable (though not optimal) policies.

Parr and Russell's 4x3Maze: The 4x3Maze task has deterministic (partially observable) observations but slightly stochastic actions. The starting location is random, which makes the fitness calculation in a phylogenetic settings cumbersome – many trials must be 'wasted' to obtain one single fitness value. RPGs achieve near-optimal performance. Surprisingly given the stochasticity of the initial position, phylogenetic methods NES and FEM do not fare significantly worse than ontogenetic LRWR.

The T-maze: The T-maze is deterministic. What makes this task hard is the requirement to 'memorize' the first time step's observation all the way to the end of the episode. On this task, all four algorithms perform similarly,

Task	Hand	RPG	LRWR	FEM	NES
Tiger	0.67	0.67	−6.5	−13.4	-15.6
Shuttle Docking	1.69	1.31	0.709*	0.34	0.45
4x3Maze	0.27	0.27	0.240	0.246	0.21
CheeseMaze	0.257	0.257	0.177	0.112	0.10
T-Maze3	1.0	1.0	0.917	1.0	0.98
T-Maze5	0.667	0.667	0.615	0.667	0.64
T-Maze7	0.5	0.5	0.463	0.485	0.455

Table 7.2: **Discrete POMDP benchmarks: comparative results.** This table shows results averaged over 25 runs. Displayed are the average rewards obtained for the four algorithms LRWR, RPGs, NES and FEM. Note that for LRWR, only the Shuttle Docking results for policies that converged are shown. The same batch sizes are used as in Chapter 6 to obtain the results for NES and FEM, and all algorithms were given an equal number of episode evaluations. RPGs were trained using learning rate 0.001 and momentum 0.9, with the baseline being the average return from the previous batch. Shown results were calculated using the average of 200 roll-outs.

at least up to maze length 7. After length 10, NES, FEM and LRWR do no better than the simple memoryless policy (average reward less than 0.177) – always moving right except when it reaches the end of the hall, where it will randomly move either up or down. RPGs can learn the optimal policy of hall lengths up to nearly 100 (see Chapter 3).

The 89-state maze: This highly stochastic maze is interesting because no algorithm as of yet has achieved human level performance. A human (Michael Littman) trained on a simulator and managed to achieve success in all trials, while to date no POMDP algorithm exists that can approach this level of performance. The task has 89 states, noisy observations and unreliable actions. The symmetry of its floor plan requires the agent to capture long-term time dependencies if it is to find the goal quickly. Chapter 3 describes the result for RPGs, and results for NES and FEM were obtained using the same number of evaluations, a batch size of 100, and 250 episodes per aggregated fitness computation. NES and FEM operated on a 2-cell LSTM policy architecture (larger architectures had too many trainable parameters, making both NES and FEM unreasonably slow). Table 7.3 shows comparative results. Running phylogenetic algorithms on this task has proved to be nearly pointless, most likely because of the task's inherent stochastic nature and relatively large state space. The author did not manage to get the phylogenetic algorithms to achieve better results even than the simple memoryless SARSA(λ) policy. RPGs, however, perform quite well, though after a significant amount of training (30,000,000 iterations).

Algorithm	goal%	median steps
Random Walk	26	> 251
Human (Michael Littman)	100	29
Linear Q (seeded)	84	33
Memoryless SARSA(λ)	77	73
RL-LSTM	94	61
RPGs	95	58
NES	67	101
FEM	72	92

Table 7.3: **89-state maze: comparative results.** This table provides an overview of the performance of several algorithms on the 89-state maze. Shown are the percentage of policy roll-outs that reached the goal within 251 steps, and the median number of steps to task completion. Results for algorithms other than RPGs, NES or FEM are taken from (Littman et al., 1995) and (Bakker, 2004).

While NES and FEM do well on black box optimization tasks, their limitation pertains to the limited number of parameters that can be trained at one time (maximally a few dozen, in practical experience). RPGs do not suffer as severely from this problem since it can use BPTT to attribute changes to individual weights.

Non-Markovian double pole balancing: This is a hard task that most conventional reinforcement learning algorithms cannot solve. In fact, RPGs constitute the first ontogenetic method to solve this task. Table 7.4 shows the results on the non-Markovian double pole balancing benchmark for various algorithms, including RPGs, NES and FEM. On this task, which is continuous, RPGs actually perform considerably worse than both NES and FEM, which are among the best known algorithms for this problem except for the evolutionary algorithm CoSyNE (Gomez et al., 2006). This reaffirms the assumption that continuous control has not been satisfactorily solved in an ontogenetic setting, especially if the task is deterministic. Nevertheless, RPGs constitute the first ontogenetic algorithm to solve this task at all, and future research may further improve its performance. It must be noted that since this is a noise-free, completely deterministic task, phylogenetic approaches may have a natural advantage over ontogenetic methods (see Chapter 2).

7.4 Summary

The comparative results for NES and FEM for black box optimization indicate that NES is a highly competitive black box algorithm on par with

Method	Evaluations
SANE	262,700
ESP	7,374
RPGs	6,139
CMA-ES	3,521
CoSyNE	1,249
FEM	1,304
NES	1,443

Table 7.4: Non-Markovian double pole balancing. Shown are the reported average numbers of evaluations for SANE (Moriarty and Miikkulainen, 1996), ESP (Gomez and Miikkulainen, 1997), RPGs, CMA-ES (Hansen and Ostermeier, 2001; Igel, 2003), CoSyNE (Gomez et al., 2006), FEM (Wierstra et al., 2008c), and NES. Note that the results for FEM only include successful runs, since FEM failed to find the optimum controller in 48 out of 200 trials, while NES always found the solution.

CMA-ES, especially on the harder ill-shaped unimodal functions and multimodal benchmark functions. Interestingly, NES' performance on the double pole balancing benchmark is significantly better than CMA-ES'. The development and optimization of the CMA-ES algorithm took nearly 10 years, which leads one to suspect that similarly stepwise improvements should be possible for NES as well.

Unfortunately, FEM's performance, while good on the double pole balancing benchmark, is not as satisfactory. FEM's performance is more sensitive to its settings (learning rate, batch size) than NES and needs tuning. This disadvantage hampers its application to more interesting applications. Nevertheless, its simplicity might be an indication of room for possible complexifying future enhancements.

On the discrete reinforcement learning benchmarks, Recurrent Policy Gradients always outperform the other algorithms. RPGs even produce the best known result on the difficult and highly stochastic 89-state maze, which is an interesting benchmark since human-level performance has not yet been achieved by any algorithm. The ontogenetic nature of RPGs allows it to profitably use all information in an episode, while FEM and NES cannot – and as such, the phylogenetic algorithms underperform (compared to RPGs) on these tasks. The relatively bad performance of ontogenetic LRWR can likely be attributed to the fact that all information to update a policy must be present in every batch. In the presence of stochasticity, in order to achieve reasonable results, these batches must then be very large.

In sum, the results both on standard black box benchmarks and on an array of POMDP problems with significantly varying properties, suggest

that both RPGs and NES constitute highly competitive algorithms for both reinforcement learning and black box optimization.

Chapter 8

Conclusion

The main objective of this thesis was to develop new methodologies for tackling reinforcement learning in partially observable environments. Within one unifying framework four new algorithms were presented for solving this important yet elusive task. Two phylogenetic algorithms and two ontogenetic algorithms were developed, two derived from EM and two gradient-based. The main focus was on direct policy search methods, as this constitutes a promising and under-studied approach quite disparate from the algorithms developed in the regular reinforcement learning community. Unlike most common reinforcement learning algorithms, all four methods are in principle direct policy search methods and as such circumvent many of the problems associated with temporal difference methods.

Recurrent Policy Gradients (RPGs, Chapter 3) are a method that back-propagates a policy gradient through time using a recurrent neural network architecture, which in the case of this thesis was consistently taken to be an Long Short-Term Memory network. The ontogenetic gradient-based nature of the approach enabled the algorithm to make use of all available information, including time-specific observations. Long Short-Term Memory's ability to relate events far apart in history aided the algorithm's ability to tackle POMDPs, especially hard ones such as the 89-state maze. It must be noted that, although this approach is in principle not a value-based method, the use of a history-dependent baseline may be seen as the use of value in some sense. Although baselines do not bias the gradient, they do help performance, even significantly in the case of the 89-state maze. A hybrid approach, using direct policy search in combination with value estimation techniques from temporal difference methods, seems to be rather promising. Recurrent Policy Gradients achieve excellent performance on important benchmarks including both the deep-memory T-maze and the stochastic 89-state maze where it outperformed all other known algorithms. Moreover, RPGs turn out to be the only ontogenetic reinforcement learning algorithm that can solve the non-Markovian double pole balancing task.

The Natural Evolution Strategies algorithm (NES, Chapter 4) constitutes a simple yet competitive gradient-based alternative to the industry standard for black box optimization and phylogenetic reinforcement learning, CMA-ES. Since this early instantiation of the algorithm already has such competitive performance on both black box optimization and phylogenetic reinforcement learning, it is to be expected that future versions will outperform CMA-ES, which has been incrementally developed over the last 10 years.

Fitness Expectation Maximization (FEM, Chapter 5), the second phylogenetic approach presented in this thesis, also performs well, especially on the double pole balancing benchmark. Unfortunately, FEM does not, as of yet, achieve as robust results as NES on unimodal and multimodal black box benchmark functions, and is highly sensitive to parameter tuning. Future work will have to address these problems.

Logistic Reward-Weighted Regression (LRWR, Chapter 6) is, in its current state, the weakest of the presented algorithms. It is an ontogenetic approach derived from Expectation Maximization that uses supervised training techniques to *self-model* successful episodes of experience. Although interesting in conception as a clean derivation from Expectation Maximization, its implementation will have to be tweaked so as to include rank estimation and a more powerful utility transform function for it to be able to perform well on POMDP tasks. It is also the only of these four algorithms to only work with discrete actions. The extension to continuous action modeling is an obvious next step.

Of the four algorithms, Recurrent Policy Gradients seem to be the most versatile and generally applicable to reinforcement learning. Its elegant incorporation of history-dependent baselines promises the fruitful collaboration with value-based techniques in hybrid form. One might say that RPGs constitute one of the best POMDP solvers existent today. However, in domains that have low stochasticity and require high precision, such as the non-Markovian double pole balancing task, FEM and NES both outperform RPGs. Unfortunately, LRWR is still to immature in its development to be considered a good algorithm of choice for solving POMDP tasks. However, future investigations might accommodate this by introducing time-variant baselines analogously as done with RPGs, and by making learning online instead of batch-based.

Since both NES and FEM perform competitively with (even better than, in some cases) CMA-ES, it is logical to explore the application of NES and FEM to other black box problems, not only phylogenetic reinforcement learning. Both their elegant and clean derivations, and their surprisingly good performance could inspire future research on the possibility to incorporate various 'tricks' that have been successful at speeding up CMA-ES into NES and FEM. Hopefully this will spawn a new class of successor algorithms in the phylogenetic framework.

In summary, this thesis presented four new algorithms for black box optimization and reinforcement learning in partially observable environments, both from a phylogenetic and ontogenetic perspective. All four methods fall within the class of direct policy search methods, and the good results that were obtained can be viewed as corroborative evidence that direct policy search methods will form a viable alternative to more traditional reinforcement learning techniques such as temporal difference methods.

Bibliography

D. Aberdeen. *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Australian National University, 2003.

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

S. Amari and S. C. Douglas. Why natural gradient? In *Acoustics, Speech, and Signal Processing, 1998. ICASSP '98. Proceedings of the 1998 IEEE International Conference on*, volume 2, pages 1213–1216 vol.2, 1998.

M. Aoki. *Optimization of Stochastic Systems*. Academic Press, New York, 1967.

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.

B. Bakker. Reinforcement learning with Long Short-Term Memory. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, Cambridge, MA, 2002a.

B. Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Syst., 14*, 2002b.

B. Bakker. *The State of Mind: : Reinforcement Learning with Recurrent Neural Networks*. PhD thesis, Leiden University, the Netherlands, 2004.

J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning. Technical report, Australian National University, 1999.

J. Baxter, P. Bartlett, and L. Weaver. Experiments with infinite-horizon, policy- gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381, 2001.

H. Benbrahim and J. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems Journal*, 1997.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3).* Athena Scientific, 1996.

H.-G. Beyer. Toward a Theory of Evolution Strategies: Self-Adaptation. *Evolutionary Computation*, 3(3):311–347, 1996.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. *Natural Computing: an international journal*, 1(1):3–52, 2002. ISSN 1567-7818.

S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee. Incremental natural actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pages 105–112. MIT Press, 2007.

H. Chernoff and L. E. Moses. *Elementary Decision Theory.* Dover Publications, 1987.

L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth International Conference on Artificial Intelligence*, pages 183–188. AAAI Press, San Jose, California, 1992.

P. Dayan and G. E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

R. De Nardi, J. Togelius, O. Holland, and S. M. Lucas. Evolution of neural networks for helicopter control: Why modularity matters. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2006.

F. A. Gers, N. Schraudolph, and J. Schmidhuber. Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, 3:115–143, 2002.

F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the 16th European Conference on Machine Learning (ECML 2006)*, 2006.

F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008.

F. J. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

F. J. Gomez and R. Miikkulainen. Solving non-Markovian control tasks with neuroevolution. In *Proc. IJCAI 99*, Denver, CO, 1999. Morgan Kaufman.

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM networks. In *Proceedings of the 2005 International Joint Conference on Neural Networks*, Montreal, Canada, 2005.

V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3(6):671–692, 1990.

V. Gullapalli. *Reinforcement learning and its application to control*. PhD thesis, University of Massachusetts, Amherst, MA., 1992.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

V. Heidrich-Meisner and C. Igel. Similarities and differences between policy gradient methods and evolution strategies. In *To appear in: 16th European Symposium on Artificial Neural Networks (ESANN)*, 2008.

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

S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. springer, 2005.

C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, 2003.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, 1998.

J. Kennedy and R. C. Eberhart. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

D. G. Kleinbaum, M. Klein, and E. R. Pryor. *Logistic Regression*. Springer, 2nd edition edition, 2002.

J. Klockgether and H.-P. Schwefel. Two-phase nozzle and hollow core jet experiments. In *Proc. 11th Symp. Engineering Aspects of Magnetohydrodynamics*, pages 141–148, 1970.

N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2004.

L. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, January 1993.

M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 362–370. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, and Y. Saez. The wcci 2008 simulated car racing competition. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

S. M. Lucas and J. Togelius. Point-to-point car racing: an initial study of evolution versus temporal difference learning. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

R. A. McCallum. Overcoming incomplete perception with utile distinction memory. In *Machine Learning: Proceedings of the Tenth International Conference*. Morgan Kaufmann, Amherst, MA, 1993.

R. A. McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 387–395. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proc. Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI '99). Morgan Kaufmann*, pages 427–436, 1999.

J. Moody and M. Saffell. Learning to Trade via Direct Reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889, July 2001.

D. E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *NIPS*, 2003.

R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1088–1094. Morgan Kaufmann, 1995.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2219 – 2225, Beijing, China, 2006.

J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008a.

J. Peters and S. Schaal. Natural actor critic. *Neurocomputing*, 71(7–9): 1180–1190, 2008b.

D. Prokhorov. Toward effective combination of off-line and on-line training in adp framework. In *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, 2007.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, TU Berlin, 1971.

M. Riedmiller. Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In J. Gama, R. Camacho, P. Brazdil, A. Jorge, and L. Torgo, editors, *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pages 317–328. Springer, 2005.

R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Monte Carlo Simulation, Randomized Optimization and Machine Learning*. Springer-Verlag, 2004.

G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

J. Schmidhuber. RNN overview, 2004. URL `http://www.idsia.ch/~juergen/rnn.html`.

J. Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In B. Goertzel and C. Pennachin, editors, *Artificial General Intelligence*, pages 119–226, 2006.

N. Schraudolph, J. Yu, and D. Aberdeen. Fast online policy gradient learning with smd gain vector adaptation. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, Cambridge, MA, 2006.

H. T. Siegelmann and E. D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

S. P. Singh, T. Jaakkola, and M. I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *International Conference on Machine Learning*, pages 284–292, 1994.

J. Spall, S. Hill, and D. Stark. Theoretical framework for comparing several stochastic optimization approaches. *Probabilistic and Randomized Methods for Design under Uncertainty*, pages 99–117, 2006.

J. C. Spall. Stochastic optimization and the simultaneous perturbation method. In *WSC '99: Proceedings of the 31st conference on Winter simulation*, pages 101–109, New York, NY, USA, 1999. ACM.

K. O. Stanley. *Efficient evolution of neural networks through complexification.* PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX, 2004.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2005.

R. Sutton and A. Barto. *Reinforcement learning: An introduction.* Cambridge, MA, MIT Press, 1998.

R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (Proceedings of the 1999 conference)*, pages 1057–1063. MIT Press, 2001.

R. Sutton, H. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvri, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the Conference on Machine Learning (ICML-2009)*, 2009.

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

G. Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

Torcs. The open racing car simulator., 2007. URL `http://torcs.sourceforge.net/`.

C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Oxford, 1989.

P. Werbos. Back propagation through time: What it does and how to do it. In *Proceedings of the IEEE*, volume 78, pages 1550–1560, 1990.

S. Whiteson, M. E. Taylor, and P. Stone. Empirical studies in action selection with reinforcement learning. *Adaptive Behavior*, 15:3350, 2007.

A. Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks (Seattle, WA)*, pages 667–673. Piscataway, NJ: IEEE, 1991.

D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN07), Porto*, pages 697–706. Springer, 2007.

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural evolution strategies. In *Proceedings of the Congress on Evolutionary Computation (CEC08), Hongkong*, pages 3381–3387. IEEE Press, 2008a.

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Episodic reinforcement learning by logistic reward-weighted regression. In *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN08), Prague*, pages 407–416. Springer, 2008b.

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Fitness expectation maximization. In *Parallel Problem Solving from Nature (PPSN08), Dortmund*, pages 337 – 346. Springer, 2008c.

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL, in press*, 2009.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent networks. *Neural Computation*, 1(2):270–280, 1989.

S. Yi, D. Wierstra, T. Schaul, and J. Schmidhuber. Efficient natural evolution strategies. In *To appear in: Genetic and Evolutionary Computation Conference (GECCO09), Montreal*. ACM, 2009a.

S. Yi, D. Wierstra, T. Schaul, and J. Schmidhuber. Stochastic search using the natural gradient. In *To appear in: Proceedings of the 26th International Conference on Machine Learning (ICML09)*. ACM, 2009b.