Policy-Gradient Learning for Motor Control

by

Timothy Field

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2005

Abstract

Until recently it was widely considered that value function-based reinforcement learning methods were the only feasible way of solving general stochastic optimal control problems. Unfortunately, these approaches are inapplicable to real-world problems with continuous, high-dimensional and partially-observable properties such as motor control tasks.

While policy-gradient reinforcement learning methods suggest a suitable approach to such tasks, they suffer from typical parametric learning issues such as model selection and catastrophic forgetting. This thesis investigates the application of policy-gradient learning to a range of simulated motor learning tasks and introduces the use of local factored policies to enable incremental learning in tasks of unknown complexity.

Acknowledgments

Nicola Slade, for her unwavering support and unquestioning commitment.

My parents, for their encouragement and endurance.

Marcus Frean, for infecting me with doubt.

Richard Mansfield, Phillip Boyle, Richard Procter and the flocks of Memphisites for their gift of distraction.

Matthew Duignan, because he promised to cite me in his dissertation.

Contents

List of Figures

Chapter 1

Introduction

The problem of understanding how real-world autonomous agents might learn useful behaviour exists in many fields. In the context of artificial systems, adaptive control theory seeks to optimise restricted yet robust controllers from limited experience while the more ambitious field of mobile robotics emphasises the problems of uncertainty stemming from limited sensory and computational resources. In natural systems, biological theories of learning seek to explain high-level behaviour of organisms which adapt to their environments, while the neuroscience of motor control considers the low-level organisation of their sensorimotor systems.

While seemingly diverse domains, these fields all address the same problem of understanding or engineering an agent which influences the behaviour of some unknown physical system in a desired fashion. By casting these problems in the framework of optimal control, it is hoped that a theoretical understanding of the mechanisms underlying adaptive behaviour can be developed. One could argue that an efficient, robust and scalable solution to the optimal control problem is the goal of machine learning and artificial intelligence in general.

Tractable general solutions to the optimal control problem are elusive. Dynamic programming based methods (until recently 'widely considered the only feasible way of solving general stochastic optimal control problems' [SB98]) are inapplicable to real-world problems. However, the recent development of policy-gradient reinforcement learning methods for once admits a theoretically plausible treatment of realistic domains such as motor learning. While motor learning is too complex a process to be viewed entirely in terms of reinforcement or supervised learning alone, little is known about the extent to which simple trial-and-error learning is used by biological motor systems.

This thesis is motivated by the problems and solutions of learning optimal control that fit the constraints of being physically, theoretically and biologically plausible. Specifically, this thesis investigates the use of feedforward networks of stochastic binary force-generating units situated in physically realistic environments which individually optimise the parameters governing their behaviour via policy-gradient learning.

1.1 Outline of the Thesis

This thesis justifies the use of policy-gradient methods for motor learning, applies these methods to a range of tasks and introduces a local policy parameterisation and related constructive algorithm suitable for online reinforcement learning.

The thesis is organised in two parts. Chapters 2 and 3 serve as an introduction to the problem and approach taken in this thesis, while chapters 4 through 6 detail the experiments performed for this thesis.

Chapter 2 introduces the problem of learning optimal control, outlines the specific complexities inherent in real-world tasks and describes the various classes of approaches.

Chapter 3 outlines the derivation of a typical policy-gradient learning algorithm which is able to optimise control from a single sample trajectory.

Chapter 4 demonstrates how policy-gradient learning can solve the low-level motor learning task of motoneuron recruitment.

Chapter 5 presents initial results from applying policy-gradient learning to more complex motor control tasks including locomotion.

Chapter 6 introduces the use of local factored policies with a constructive algorithm to incrementally solve control problems.

Chapter 7 summarises the contributions of this thesis, reflects on its limitations and suggests possible future extensions.

Appendix A lists equations used for policy-gradient derivation for binary units, describes the muscle wrapping geometry and outlines the particle simulation used in Chapter 6.

Appendix B describes the software systems developed for this thesis.

Chapter 2

Learning Control

An *agent* is any decision-making system, artificial or natural, situated in and interacting with an *environment*, the behaviour of which is dependent on the agent's actions. The problem of learning control involves adapting this behaviour in order to meet some task-dependent goals.

This chapter introduces the theoretical framework of the optimal control problem, outlines approaches to the problem and shows how the specific complexities of real-world problems limit the suitability of most approaches.

2.1 Optimal Control

The framework of *optimal control* is a general formulation of the control problem in which the agent does not receive any explicit feedback in terms of optimal actions. The problem is expressed as choosing actions in a sequential interaction with a dynamical system such that an objective function is maximised. A formal model which captures this interaction is the *Markov decision process* (MDP) and its variants [Bel61].

In an MDP the environment is modeled as a stochastic process which generates a *trajectory* $X = \mathbf{x}_{0:T}$ of length T which is a sequence of *states* $\{\mathbf{x}_0, \ldots, \mathbf{x}_T\}$ in the continuous state space $\mathcal{X} = \mathbb{R}^N$ of a discrete-time dynamical system. The trajectory begins by sampling from the initial state distribution $p(\mathbf{x}_0)$ and then evolves though sampling from the *transition model* $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$, where each action in the sequence $\mathbf{a}_{1:T}$ is selected by the agent from the space of possible actions $\mathcal{A}$ which is assumed discrete, e.g. the m-dimensional binary space $\mathcal{A} = \{0, 1\}^m$. This process is *Markovian* because the transition model dictates that, for each t, the succeeding states are conditionally independent of all previous states given the current state. If the agent's observations are not Markovian then it is convenient to assume that each observation $\mathbf{y}_t \in \mathbb{R}^M$ is generated from some underlying state via an *observation model* $p(\mathbf{y}_t|\mathbf{x}_t)$. Such a process is said to be a *partially observable* MDP (or POMDP).

At each interaction, the agent's observations are augmented by a scalar *reinforcement signal* (or *reward*) $r_t \in \mathbb{R}$ which is sampled from the stochastic *reward model* $p(r_t|\mathbf{x}_t)$ conditional on the most recent state. The optimal control task

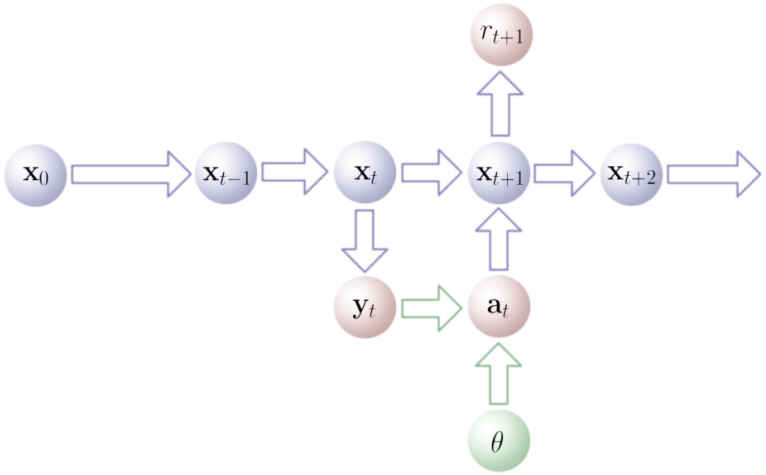

Figure 2.1: **Partially Observable Markov Decision Process.** This Bayesian network illustrates the conditional independencies between the random variables in a POMDP. The blue variables and distributions are hidden, the red are observable, and the green are adaptable by the agent. In this diagram actions are assumed to be sampled from a policy parameterised by θ. The observations, actions, rewards and policy parameters are only shown for the single time step, t.

consists of selecting at each time t an action $\mathbf{a}_t$ (given past experience in the form of $\{\mathbf{y}_{0:t}, \mathbf{a}_{0:t-1}, r_{0:t-1}\}$) such that the future expected reward is maximised over some (typically infinite) time horizon. For a completely observable MDP, the optimal controller need only take into account the most recent observation. Thus, it can be represented as a deterministic *reactive policy*, i.e. a deterministic function $\pi : \mathcal{X} \to \mathcal{A}$ mapping states to actions. However, for partially observable problems the use of stochastic reactive policies which map observations to a distribution over actions, i.e. $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$, are known to perform better [PTJ94].

In summary, the assumptions of this model are that there is some underlying state that evolves via a fixed Markov process dependent on the actions of the agent, and that the rewards and observations are generated by a fixed stochastic process dependent only on the current state. These conditional independencies are summarised in the Bayesian network shown in Figure 2.1.

The generality of this formulation is such that it is difficult to imagine a real-world problem that cannot be cast as a POMDP. A seemingly limiting assumption is that, while all of the densities are assumed fixed, the observed dynamics of real-world systems vary over time and space. However, these non-stationary dynamics can be modeled in the POMDP framework by assuming hidden state variables as needed. Also, the reward density is general enough to capture almost any control objective imaginable, including multiple weighted goals, as well as time-to-goal criteria.

2.2 Learning

The only information available to an unbiased agent which seeks to optimise a POMDP is revealed through its interaction with the environment. Thus, in order for a learning agent to adapt its behaviour, it must resort to a strategy of trial-and-error, trying out different control strategies in order to generalise from past experience and learn representations which will improve its future performance. This process can be considered as a search or optimisation problem[1].

At each interaction, the performance feedback received in optimal control problems (i.e. the reward signal) does not relate to any specific part of the action vector or necessarily even to the most recent actions. This introduces the problem of *credit assignment*, or determining the value of individual actions in the trajectory.

Because of this coarse feedback, the optimal control problem is the most general form of machine learning. For example, the supervised learning problem is over-constrained in that it assumes a teaching signal (as opposed to the *distal error* signal in optimal control) which prescribes the optimal action $\mathbf{a}_t^*$ at each interaction[2]. By contrast, unsupervised learning is an ill-posed problem where the goal of learning structure from the data is under-constrained.

The generality of the POMDP model comes at a cost, however. Even given all of the parameters of a POMDP, finding a solution is, in general, intractable [Lit96]. The difficulty of learning stems from the computational complexity (the time and space complexity of the learning algorithm) and the sample complexity (informally, the number of samples needed to optimise control)[3] of the problem. Both computational power and experience are scarce.

2.2.1 Complexity of Physical Systems

Approaches to tractably solving unknown POMDPs involve reducing the complexity of the problem through simplification of the model by abstracting away details or making assumptions. Typical assumptions include complete state observability (i.e. all observations satisfy the Markov property); access to a restart distribution or the existence of an identifiable recurrent state (allowing trajectories to be partitioned into independent *episodes*); access to a *desired trajectory* (a reward-maximising trajectory) or to the underlying transition, observation and reward models; or restricting to problems with small, finite, discrete state and action spaces with simple, noiseless dynamics.

Learning algorithms taking advantage of some or all of these assumptions have had remarkable successes in limited domains such as game-playing [Tes95].

[1] From a more general probabilistic perspective, the learning process is viewed as an inference problem in which the agent's goal is to infer a posterior performance distribution over controllers given past experience and a prior over controllers.

[2] Every supervised learning problem can be cast as an immediate reward optimal control problem by choosing a distance measure from the desired actions to the actual actions.

[3] While sample complexity is well understood in supervised learning, it is only beginning to be analysed in optimal control problems [Kak03].

However, whether motivated by a belief that successful approaches to such restricted domains would scale up to more complex problems or by theoretical and computational necessity, it seems that ignoring aspects of the real-world leads to intractable or inapplicable solutions to complex tasks.

Understanding why these algorithms fail requires an appreciation of the complexities of real-world problems. An assumption of this thesis is that such problems can be adequately described by a POMDP whose dynamics are governed by Newtonian physics (essentially a set of second-order, nonlinear differential equations). A number of features of this realistic POMDP complicate learning: the huge state spaces, the limited observability of the state, the complexity of the dynamics, the many degrees of freedom available to the agent and the cost of sampling.

Huge State Spaces

The state spaces of physical systems are continuous with very high dimensionality, e.g. the state of a single unconstrained 3D rigid body is a 12 dimensional quantity[4]. While constraints in articulated bodies reduce the effective dimensionality of the system, the state space of realistic systems is far greater than in control problems typically tackled in machine learning. For example, the human arm (including the hand) has around 30 joints with more than 50 muscles giving a state space dimensionality of well over a hundred [TL03].

The immediate impact of this is described by the *curse of dimensionality* [Bel61] which observes that the hypervolume of a space grows exponentially as a function of its dimensionality. Thus, any algorithm which uses resources proportional to the hypervolume of a space (such as a naïve partitioning method) will be intractable in high dimensions. Some form of task-dependent generalisation is mandatory. In practice, scalable methods must make no assumptions about the size of the state space — real-world state spaces are effectively infinite.

Limited Observability

The observations sensed by a real-world agent are corrupted by aliased, noisy, redundant and irrelevant features while missing important state information; in general they are not Markovian. In physical tasks the hidden state includes important predictive quantities such as forces, accelerations, velocities, masses, friction, compliancy, etc.[5]

This has a number of immediate consequences. Complete observability must be abandoned as an assumption for real-world tasks. Tasks must be considered non-episodic as, in general, it is impossible to identify a recurrent state. Also, an optimal controller must take into account past observations in selecting actions.

[4]Including just position, orientation, velocity and angular velocity and ignoring other (assumed fixed) physical parameters such as mass, friction, and elasticity.

[5]Also, partial observability is a consequence of the use of hierarchical control. By definition, each sub-controller can only see a subset of the observations.

Complex Dynamics

While simple physical systems such as unconstrained particles or rigid bodies evolve in relatively predictable continuous trajectories, interacting systems linked via interdependent constraints (such as non-interpenetration and joints) evolve in complex nonlinear trajectories. These constraints also introduce discontinuities into the dynamics.

In addition to this, even simple dynamical systems can exhibit extreme sensitivity to initial conditions (or chaotic behaviour). Combined with noise in observations this means dynamics for complex tasks are inherently unpredictable at a fine scale. Furthermore, even if the rules governing the interactions were accurately known, properties of the behaviour cannot be easily predicted or inferred without resorting to simulation. A consequence is that exact planning in the presence of uncertainty, which requires integration over all possible transitions, quickly becomes intractable.

Huge Action Spaces

While the state space of physical systems is large, the number of actions available to realistic control systems is typically even larger. This is especially true of motor control tasks, where the number of degrees of freedom of the agent is vastly larger than the physical degrees of freedom. For example, the human body has hundreds of muscles each of which contains hundreds of individually activated motor units, yet its skeleton admits less than a hundred degrees of freedom. Real-world systems are severely under-constrained. The immediate consequence of these huge action spaces is a combinatorial explosion of search, meaning any undirected exploration scheme will require an intractable number of samples.

The large action space is exacerbated by the coarseness of the reward signal. For example, in parameterised supervised learning approaches, the mapping from rewards to the actions that caused them is assumed given, and hence the learning amounts to mapping the actions (or outputs) to the parameters controlling them. In optimal control problems this mapping is confounded by the distal error problem which means that the reward signals are not explicitly related to individual actions.

In general, the effect of a group of actions is more complicated than a simple combination of the individual actions. This is especially important in motor systems, where temporal credit assignment is complicated by physical aspects such as momentum (for example, in a frictionless environment, forces may have an infinitely delayed effect on state) and structural credit assignment is complicated by redundancy as different actions may have similar effects.

Costly Sampling

In machine learning problems it is typically assumed that the data set is given as an input to the problem and is of fixed size. This means that the learning agent is freed from any decision about extending the data set.

In control tasks, however, the potential size of the training set is unbounded and the act of sampling is costly. This throws up complications for the agent, which has to deal with the selection of training data and the inevitable tradeoff

between exploration and exploitation. Optimal action selection in the presence of costly sampling is an sophisticated combination of information-gathering and reward-maximising behaviour.

In addition, from a practical perspective, with current technology the cost of sampling in both real and simulated physical systems is prohibitively expensive.

Any approach to learning optimal control in real-world tasks must contend with all of these issues. However, existing approaches ignore one or more of these complications. For example, adaptive control theory ignores the complexity and uncertainty of dynamics, traditional reinforcement learning ignores the huge state space and the need for generalisation, supervised learning ignores the distal error problem, and robotics typically ignores the large number of degrees of freedom.

2.3 Representation

Learning agents construct representations which bias future interactions and enable them to optimise their behaviour. While the potential complexity of real-world tasks appears overwhelming, these tasks must contain structure which enables them to be solved. In general, this structure can occur in many places such as in the dynamics, in the long-term value of states, or in the optimal policies. Intuitively, it would seem necessary for any tractable solution to a sufficiently complex task to use representations which coincide with the task's inherent structure. However, the location of this structure in various problem domains is difficult to pinpoint *a priori*.

Conceptually, approaches to the optimal control problem make an implicit assumption about the location of this structure. While, intuitively, it would seem that the problem is one of searching for the best controller, most approaches attempt to learn some prespecified intermediate representation, and treat estimating this structure as an end unto itself. These indirect optimisation methods potentially fall into the trap of solving a hard problem on the way to solving an easier one. They trivialise action selection by assuming that some facilitating intermediate representation is available or can be learnt. In the end, the use of any representation must be subordinate to the overall goal of optimising behaviour. From a decision-theoretic or Bayesian perspective, a specific representation is valuable only insofar as it would change an agent's actions[6].

Domain-specific knowledge may be useful in suggesting forms of representation. For example, if the reward signal was known to be highly correlated with the position and orientation of the agent, then a reasonable method would use stochastic filtering theory to derive an intermediate error signal which related to the certainty of the state, perhaps by reducing the entropy of the belief distribution over the hidden pose state. While useful from an engineering perspective in producing transparent control, such approaches rely on prior

[6]This theoretical perspective is beginning to be recognised in cognitive science in the form of *action-centred representations* [Cla97].

knowledge, are not applicable in the general case and are prone to anthropomorphic bias.

Approaches to learning optimal control can be categorised based on how they approach the distal error problem. They are typically divided into those that optimise by learning some intermediate representation (such as predictive models or value-functions) and those that directly optimise the controller. This distinction appears in adaptive control theory as indirect versus direct adaptive control methods and in reinforcement learning as model-based versus model-free learning.

2.3.1 Model-Based Control

Model-based approaches learn representations to better predict the dynamics of the environment. These representations are used to predict the observations (*forward models*), infer hidden state which allows better prediction (*filtering*), and learn which actions are needed to achieve desired dynamics (*inverse models*). These representations are used to tackle the distal error problem by transforming the distal error signals into the error vectors needed for traditional control. In essence, model-based approaches decompose the optimal control problem into the separate sub-problems of state estimation, prediction, and action selection.

Techniques used in model-based control (such as feedback control, filtering, regression and linearisation) are theoretically well-developed. In order to develop the theory, however, very limiting assumptions are necessary. For example, most adaptive control theory assumes a linear state space model with Gaussian noise (e.g. linear-quadratic-Gaussian control methods and Kalman filtering)[7].

A typical model-based approach is the inverse dynamics approach to feedforward control. This involves computing a desired trajectory which maximises the reward, and then solving explicitly for a force trajectory which constrains the system to follow that trajectory. This approach divides interaction with the environment into learning and exploitation phases and separates the processes of perception and action.

Apart from the difficulties of applying such an approach to complex dynamical systems, this approach is widely regarded as biologically implausible because the torque at each joint is not a directly controlled entity. This is because muscles may span multiple joints and generally operate unlike simple torque generators, as their mechanical behaviour is strongly state-dependent. In addition, most descending motor commands do not address individual muscles but muscle synergies (groups of muscles) [Lee84].

Fully model-based approaches are also generally intractable: even if accurate forward and inverse models were learnt, state estimation was solved, and there was complete knowledge of the underlying value-function, such methods applied to complex tasks require evaluating a high-dimensional integral

[7]Relaxation of these assumptions allowing for nonlinear, multi-modal probability densities are beginning to appear, such as the use of sequential Monte Carlo (particle filtering) methods for state estimation.

over future trajectories and over belief states.

2.3.2 Model-Free Control

An alternative to the use of models in control is model-free control. Model-free approaches recognise that the dynamics of physical systems are complex and largely unpredicable, and abandon any attempt to model them. Model-free control for realistic systems began with the relative successes of behaviour-based robotics [Bro86], which stressed the use of decentralised, reactive, robust and computationally cheap controllers.

Model-free control has several implications. In particular, these approaches assume that the overall behaviour of an agent is an emergent phenomenon, derived from the coupled agent-environment dynamics. Thus, the environment and the agent are considered inseparable and must be treated as a single system. Rather than viewing dynamics as something to compensate for, it is treated as something to exploit. This idea of emergence is a common viewpoint in connectionist and dynamical systems theory views of interaction. A corollary of this is that the role of perception is not to yield perfect information by estimating hidden processes and states of the world, but rather to quickly select the most suitable information for the task at hand.

The rest of this thesis assumes a model-free approach to learning optimal control.

2.4 Value-Function Learning

The usual approach to model-free control is *reinforcement learning* (although such methods are not always model-free). Confusingly, this term is used interchangeably to refer to several concepts: a problem (optimal control in small discrete MDPs), its typical solution (learning the optimal value-function) and the underlying philosophy (model-free). In this thesis, reinforcement learning will be used a synonym for the learning optimal control problem while *value-function learning* will refer to the traditional model-free solution of learning the optimal value-function.

The *value-function* of a policy is a mapping from a state to the expected long-term reward an agent receives if it starts in that state and follows the policy[8]. Value-function learning involves computing improved estimates of the *optimal value-function*. All value-functions for discrete MDPs have structure in the Bellman equations[9] which relate the value of states. Value-function learning methods exploit this structure tractably using the computational technique of dynamic programming (a means of solving constraint satisfaction problems through repeated substitution of estimates). For more information see [SB98].

Informally, value-function methods acquire a model of the relationship between observations, actions and long-term rewards; in effect, constructing an

[8] *Value-functions* here refers to any mapping onto long-term reward, so the term includes action-value (so-called Q value) and advantage functions.

[9] The equivalent in continuous systems are the Hamilton-Jacobi equations.

internal reward signal that is less delayed than the original, external one. Action selection is implicitly derived from this model and the selection of exploration policy (typically some heuristic based on greediness, novelty and stochasticity).

These algorithms rely on two strict assumptions: that observations are Markovian and that the value of each observation is represented explicitly in a look-up table. If these assumptions hold then convergence is guaranteed to the globally optimal policy (assuming each state is visited infinitely often). If not, for example if the state space is continuous or high dimensional therefore necessitating an approximate representation of the value-function, or if the task is partially observable, then reinforcement learning can fail catastrophically (diverging from an optimal value-function) even in small MDPs. Unfortunately, continuous, high-dimensional and partially observable processes are characteristic of real-world systems.

Heuristic attempts to scale up value-function learning to continuous systems typically involve either function approximation or adaptive discretising of the state and action spaces [MA95, tH01]. In general, this discretisation not only introduces hidden state, violating the Markov property and defeating any convergence guarantee, but the resulting number of states makes learning very slow. More principled approaches of representing continuous value-functions using Gaussian processes have recently been proposed [EMM03, RK03] (utilising continuous versions of temporal difference learning and policy iteration respectively) which model uncertainty in the value estimates. Again, however, convergence is not guaranteed. Also, representing the value-function using a nonlinear function approximator means that each action selection step involves solving a nonlinear optimisation problem.

Extensions to value-function methods to handle partial observability are also not promising. While, theoretically, it is possible to convert a discrete POMDP into a belief-state MDP such that optimising the MDP optimises the POMDP [Ber76], even monitoring such a belief-state requires knowing transition and observation models, and the resulting MDP is continuous and very high-dimensional. Heuristic techniques involving the use of a state estimator cannot guarantee to build a perfect Markovian state representation.

While value-function learning is model-free, by computing an intermediate representation it still amounts to an indirect optimisation. In essence, it solves a harder problem on the way to solving an easy one: acting optimally only requires the relative values of actions and not the absolute values. Intuitively, it should be simpler to determine how to act than to determine the value of acting.

2.5 Direct Policy Search

An alternative model-free approach is to abandon learning any intermediate representation and learn the controller directly. This is achieved by parameterising the policy and then optimising these parameters.

The choice of parameterisation yields a class of policies $\Pi = \{\pi_\theta : \theta \in \mathbb{R}^m\}$

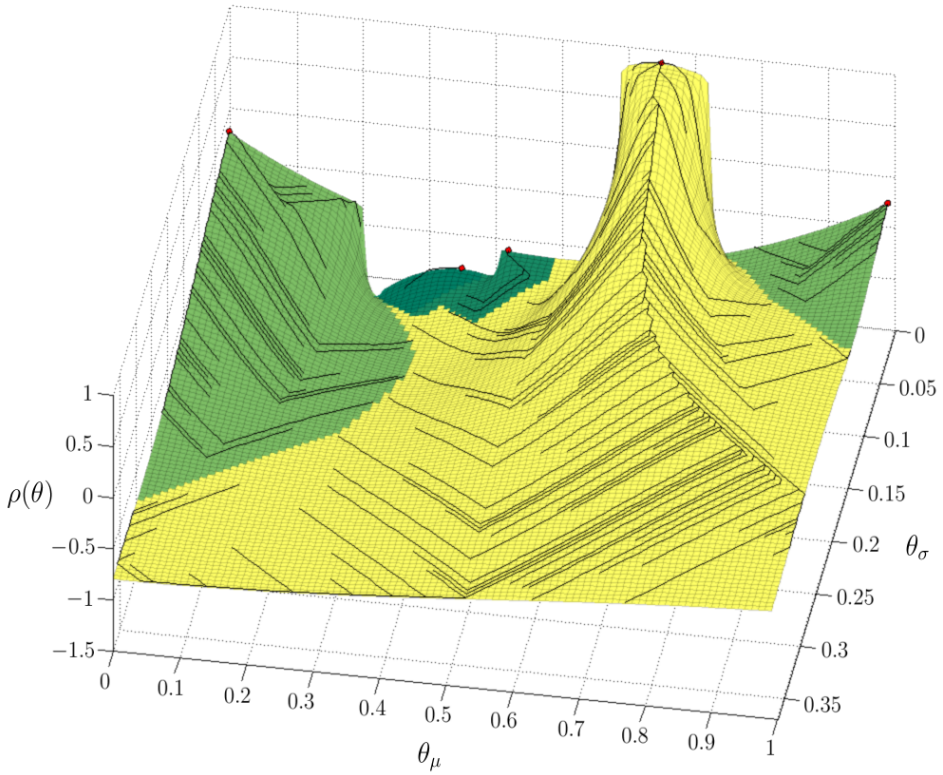

Figure 2.2: **Hill-Climbing Policy Search.** A notional example of policy search by hill-climbing in a one-dimensional problem. This diagram shows the objective surface derived from a Gaussian stationary distribution which is directly controlled by a two parameter policy (mean θ_μ and standard deviation θ_σ) and a deterministic reward function (which follows the surface at $\theta_\sigma = 0$). The lines on the surface show sample learning trajectories, with the colour of the surface indicating the performance of the converged optimum for each setting of initial parameters. This diagram illustrates the benefit in hill-climbing search of using initial parameters which are unbiased and have high entropy. It also shows convergence to a deterministic policy, i.e. $\theta_\sigma = 0$, where the entropy of the stationary distribution can be directly controlled.

parameterised by the vector θ. These parameters induce a distribution over trajectories $p(X|\theta)$ in the state space where each trajectory X is generated by sampling actions from the policy π_θ. The expected performance given by $\rho(\theta)$ is defined as the expected reward under this distribution, i.e.

$$\begin{aligned}\rho(\theta) &\equiv \mathbb{E}\left[r(X) \mid \theta\right] \\ &= \int_{\mathcal{X}} r(X)p(X|\theta),\end{aligned}$$

where $r(X)$ is the reward criterion which maps the sequence of individual reward signals received in the trajectory to a scalar value. Examples of reward criteria are the average and discounted reward functions, i.e. $r(x_{1:T}) = \frac{1}{T}\sum_{t=1}^{T} r_t$ and $r(x_{1:T}) = \sum_{t=1}^{T} \gamma^t r_t$, where γ is the discount rate. The parameters also induce a *stationary distribution* $p(\mathbf{x}|\theta)$ over the state space. Informally, this distribution can be interpreted as the fraction of time spent in state $\mathbf{x}$ along a trajectory under π_θ (or more correctly the limit of this as the length of the trajectory approaches infinity).

Learning optimal control becomes the problem of adjusting θ in order to maximise this high-dimensional integral. This has similarities to *Markov chain Monte Carlo* methods which evaluate high-dimensional integrals by simulating Markov chain transitions to generate samples from the stationary distribution. In direct policy search, however, the goal is to maximise this integral by modifying the transition parameters which control the stationary distribution. Figure 2.2 shows an example of policy search by hill-climbing.

There are several benefits to searching in the space of parameters. Firstly, the approach is applicable to POMDPs. Informally, this is because any choice of parameterisation can always be considered to be a restricted class of policies which only uses observable data from the underlying MDP. Secondly, unlike model-based or value-function methods, the computational complexity of policy search methods is not directly dependent on the size of the state space or the complexity of the dynamics. Thirdly, restricting the policy to a parameterised family of functions directly addresses large state and action spaces through generalisation. Finally, these methods allow the ability to bias learning by including prior knowledge into the parameterisation.

Parameterising the policy transforms a sequential decision problem into a function optimisation problem. Instead of the agent choosing which action to perform next, the choice becomes which θ to sample next. Casting optimal control as a function optimisation problem opens the door for the application of statistical learning theory. Two classes of function optimisation used for learning are *global optimisation* and *gradient-based optimisation* techniques.

2.5.1 Global Optimisation

In general, any change to the policy parameters will have a non-trivial effect on the stationary distribution and thus the performance of the policy. Therefore, it could be argued that the objective surface will have features that make it inherently difficult to search, i.e. it will be multi-modal at many length scales, with little predictable structure.

Global optimisation methods are a class of heuristic search methods which attempt to optimise such surfaces. They consist of evaluating a parameter vector θ by sampling one or more Monte Carlo trajectories to give samples of the objective function $\widehat{\rho}(\theta)$ which are used to propose new points to sample. These methods are generally variants of random search, genetic algorithms [Hol75] or simulated annealing [KGV83] and are essentially all instances of the general class of stochastic (and often parallel) non-gradient-based hill-climbing search algorithms. By using such methods, the problem is transformed into choosing a family of parameterised controllers and search operators. These methods typically require no constraints on the parameterisation or search method so can include controllers with memory, arbitrary changes of controllers, etc.

A recent example of global optimisation applied to a motor control is the simulated robot weight-lifting task in [RB01] where a simple random search algorithm was used with a set of deterministic proportional-derivative controllers. The search consisted of, after each episode, perturbing the current parameters with Gaussian noise and, with a small probability, taking a step toward the current best parameters seen so far. The authors attribute their success to the prior information built into the parameterised policy.

The *No Free Lunch* theorem [WM97] suggests that, in general, this will always be the case. It states that a search algorithm which does well for one particular class of problems necessarily must do poorly on others. Moreover, the theorem proves that it is not enough to observe that a particular problem contains exploitable structure: the existence of that structure does not justify a particular algorithm. The structure must be reflected in the choice of algorithm, i.e. in the prior information built into the parameterisation or the search operators. Thus, finding a parameterisation and set of search operations which readily admits a solution when applied with a particular global optimisation procedure will only succeed if prior knowledge of the task is reflected in this choice. Of course, it would be possible to search for good parameterisations and operators but this search process would continue in an infinite regress.

Global optimisation methods essentially bypass the credit assignment problem by ignoring the actions or observations in a particular trajectory. Thus, they make limited use of the available information. Intuitively, the number of samples needed by global optimisation is much greater than the true sample complexity of these problems. Another shortfall is their unsuitability to online learning in the case of non-episodic tasks with occasional or continuing reward.

While useful in producing sample solutions for a given task, global optimisation methods provide little insight into the processes behind online motor learning or adaptive behaviour. These heuristic methods are essentially pragmatic alternatives to brute force search. They have a usefulness in showing the existence of a solution rather than providing a viable explanation of its development.

A more principled global optimisation approach involves treating optimisation as a regression problem in which the samples are used to explicitly estimate a model of the objective surface. The use of Bayesian non-parametric techniques allows for both the judicious use of samples and the ability to es-

timate the uncertainty in the surface estimate. This uncertainty estimate can be used to actively explore the parameter space by choosing samples which greedily maximise the expected improvement in the best sample seen so far[10]. Whether such methods are superior to simpler local optimisation techniques is an open question. Similar ideas to explicitly model the uncertainty in parameters have been used in reinforcement learning in *Stochastic Real-Value Units* as described in Chapter 5.

2.5.2 Gradient-Based Optimisation

Gradient-based optimisation methods use estimates of the derivatives of a function to optimise it. Applied to tasks with no distal error with neural network function approximators, these methods result in the familiar *backpropagation* algorithm. Applied to optimal control problems, these methods are *policy-gradient learning* algorithms.

Policy-gradient learning algorithms rely on stochastic policies for undirected exploration to estimate the performance gradient. This is then used with some form of gradient ascent, resulting in an algorithm which is guaranteed to converge to a locally optimal policy. These algorithms for the first time allow a theoretically plausible treatment of complex control tasks.

They are not without problems, however. These include the long learning times resulting from the large variance of gradient estimates and the use of undirected exploration, the convergence to local minima, and typical parametric learning problems such as catastrophic forgetting and selecting a parameterisation with appropriate complexity.

2.6 Motor Learning

Animals are the only known examples of systems which are able to solve complex real-world control problems. Their remarkable ability to quickly learn robust, adaptive behaviour in complex environments serves as an inspiration for approaches to the optimal control problem. Biological motor control is concerned with the processes by which an organism produces physical forces. These forces are exclusively generated through the contraction of muscles and serve a range of purposes including moving joints, opposing static loads, and stabilising the body from unpredicted perturbances. From an optimal control perspective, the motor system solves the problem of learning to generate patterns of muscle activation that maximise task-dependent objective functions in the presence of unknown and time-varying parameters.

Relatively little is known about the mechanisms of biological motor learning. A number of fundamental questions, such as the relevant coordinate system in which sensorimotor adaptation occurs (whether in terms of desired trajectories or velocities, individual forces or torques, muscle parameters such as stiffness and viscosity or individual contractions), the types of objective criteria

[10] Marcus Frean, personal communication, July 2002.

implicitly used (whether kinematic, dynamic, or both), and the role of noise in learning (whether a hindrance or a help) are subject to wide debate.

General characteristics of the approaches that organisms have evolved to solve motor control problems are beginning to be reverse-engineered. These include utilising (rather than circumventing) the underlying environmental dynamics through passive compliance and preflexes [KF99], and the biasing of control through motor primitives such as muscle synergies [MIGB94], and open-loop activation patterns such as motion pattern generators and reflexes. Other natural processes which are traditionally viewed as limitations or constraints of the motor system may actually simplify the problem by biasing the search. These include development, selective attention, shaping, and coadaptation. A dominant view in cognitive science is that animals utilise action-centred representations wherever possible, utilising abstract intermediate representations as a last resort [Cla97]. Evidence for the use feedforward model-free solutions includes the fact that many motor skills are attained in the absence of explicit feedback about muscle contractions or joint angles [Bar02].

This biased and fundamentally coupled view of interaction suggests the use of model-free and, in particular, direct optimisation approaches for tractable control in real-world problems.

2.7 Discussion

This chapter has given a brief overview of the optimal control problem in real-world systems and introduces the broad classes of solutions. Real-world problems have many features which complicate learning control, and this chapter has argued that these serve as a justification for the use of policy-gradient learning.

From a physical perspective, model-based solutions rely on prior knowledge and are intractable in high-dimensional systems with complex dynamics. From a theoretical perspective, value-function approaches are inapplicable in continuous, high-dimensional, and partially-observable systems. Existing global optimisation approaches are inapplicable to online learning of controllers with a large number of parameters, and require a heuristic choice of search operators. Additionally, the biological perspective suggests the use of model-free optimisation where possible. Thus, for optimising control in complex physical systems, the only current principled approach which robustly learns is policy-gradient learning.

The next chapter discusses the derivation of a simple direct policy-gradient learning algorithm and the remainder of this thesis demonstrates the application of that algorithm to a variety of learning tasks which exhibit the complexities of real-world systems.

Chapter 3

Policy-Gradient Learning

Policy-gradient learning algorithms compute estimates of the gradient of the performance of a POMDP with respect to the parameters of a (typically stochastic) policy. This estimate can be used to optimise the policy using standard gradient-based function optimisation methods.

In general, gradient-based function optimisation consists of three separate steps: choosing a parameterisation and initial parameters, evaluating the gradient, and updating the parameters using this gradient.

1: **Parameterise**. Choose Π_θ and θ_0.
2: **Estimate Gradient**. Calculate $\widehat{\nabla}_\theta \rho$.
3: **Improve Parameters**. Set $\theta_{new} = f(\theta, \widehat{\nabla}_\theta \rho)$.

The steps of estimating the gradient and improving the policy repeat until θ has converged, i.e. until $\theta_{new} = \theta$.

This chapter discusses these steps in reverse order. A simple online gradient-ascent procedure to improve the policy will be presented, before the derivation of a commonly used algorithm which is able to estimate the gradient from a single sample trajectory (as developed in [BB99a, KYK95] and others). The choice of parameterisation is then discussed followed by the limitations and various extensions to the algorithm. For a history of policy-gradient algorithms and a more formal derivation see [BB01].

3.1 Policy Improvement

The simplest use of gradient information for optimising the parameters of an objective function is *gradient ascent*, where the gradient is used to estimate the direction of increasing performance. The simplest form of gradient ascent is *steepest-gradient ascent* which consists of updating the parameters by taking small steps in the direction of the first derivative of the objective function, i.e.

$$\theta_{new} = \theta + \alpha \widehat{\nabla}_\theta \rho(\theta),$$

where α is a small positive step-size (also known as the *learning rate*). If the

gradient estimate is reasonably accurate (within at least 90° of the true gradient) and α is small enough, then the parameters are guaranteed to converge to a region within α of a local maximum.

There are two main reasons for taking small steps in policy search. Firstly, the stochasticity in both the environment and the policy mean this is a noisy estimation problem, thus the gradient estimate must be averaged over several noisy values. Secondly, the policy parameters have a complex effect on which actions are chosen and the relationship between actions and rewards typically might be obscure. Thus, large steps might lead to decreases in the average reward.

There are two options for when to apply this update: one can either wait until the gradient estimate has converged, or apply the update after every interaction. The latter *online* learning algorithms slowly change the parameters with a moving estimate of the gradient. Thus, the parameters are updated at every time step, i.e.

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla}_\theta \rho(\theta_t).$$

The motivation for using online learning is that it is practically very simple: it uses a single free parameter α, and removes the need to detect convergence in the gradient estimate. Online learning is also known as *stochastic* learning as it introduces noise into the gradient. This random noise disrupts the deterministic convergence guarantee and means the parameter trajectory may jump between local optima. However, this noise may actually benefit the performance of the algorithm as it causes it to escape local optima.

In theory, choosing an appropriate step-size requires knowing not just the gradient, but also something about its higher-order derivatives such as the curvature of the objective surface in the vicinity of the current parameters. The step-size can be large when the gradient estimate is accurate and the objective surface is smooth. An appropriate step-size will depend on the scale of the reward signal, as well as other parameters of the learning algorithms. All of these issues mean that the choice of step-size becomes problem dependent and effectively must be set by trial-and-error.

More efficient gradient-ascent methods which make better use of gradient information, such as conjugate-gradient ascent and methods for higher order gradients have been applied to policy-gradient learning [BB99a, BB01]. While providing faster convergence they are still hill-climbing searches. As such, they are dependent on the nature of the objective surface.

3.2 Gradient Estimation

Methods for estimating the gradient of the performance function trace their history back to Williams' REINFORCE algorithm [Wil92]. The original estimator, which assumed immediate reward and episodic tasks, has since been extended to handle non-episodic tasks and partial observability. This section presents the derivation of such an estimator and discusses various extensions. For more detail see [BB01].

3.2.1 Gradient Estimator Derivation

This rather informal derivation of the gradient estimator is presented as a series of modifications to a basic estimator.

Trajectory Gradient

The performance of a policy is defined as the expected average reward received from executing that policy, i.e. the expected reward integrated over the trajectory distribution induced by the parameters. The policy parameters only impact the objective function via this distribution as the reward distribution is invariant to any changes in the policy. Thus, the gradient of the objective function is

$$\begin{aligned}\nabla_\theta \rho(\theta) &= \nabla_\theta \left[\int_{\mathcal{X}} r(X) p(X|\theta) \right] \\ &= \int_{\mathcal{X}} r(X) \nabla_\theta p(X|\theta),\end{aligned}$$

where each X is a trajectory. Directly evaluating this integral would require uniform sampling of trajectories. Although sampling from this uniform distribution is not possible, it is possible to sample trajectories by executing the policy, motivating the following form of the gradient:

$$\begin{aligned}\nabla_\theta \rho(\theta) &= \int_{\mathcal{X}} r(X) \frac{\nabla_\theta p(X|\theta)}{p(X|\theta)} p(X|\theta) \\ &= \mathbb{E}\left[r(X) \frac{\nabla_\theta p(X|\theta)}{p(X|\theta)} \mid \theta \right].\end{aligned}$$

In essence, this is a form of importance sampling where samples of gradients from the trajectory distribution are weighted by $1/p(X|\theta)$ to compensate for the difference between the sampling distribution and the uniform distribution.

Executing the policy (i.e. interacting with the POMDP using actions sampled from π_θ) generates trajectories and rewards which are samples from the trajectory distributions $p(X|\theta)$ and $r(X)$. A simple Monte Carlo method can be used to evaluate this expectation giving an unbiased estimator of the gradient, i.e.

$$\widehat{\nabla}_\theta \rho(\theta) = \frac{1}{N} \sum_{i=1}^{N} r(X_i) \frac{\nabla_\theta p(X_i|\theta)}{p(X_i|\theta)},$$

where i sums over the N trajectories.

Factoring the Trajectory

Due to the Markov property of the trajectories in an MDP, each trajectory X of length T can be factored into a product of individual state transitions, i.e.

$$\begin{aligned}p(X|\theta) &\equiv p(\mathbf{x}_0) p(\mathbf{x}_1|\mathbf{x}_0, \theta) p(\mathbf{x}_2|\mathbf{x}_1, \theta) \cdots p(\mathbf{x}_T|\mathbf{x}_{T-1}, \theta) \\ &= p(\mathbf{x}_0) \prod_{t=0}^{T-1} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \theta).\end{aligned}$$

This allows the likelihood ratio of the trajectory to be factored into the sum of the log gradients from each transition, i.e.

$$\begin{aligned}\frac{\nabla_\theta p(X|\theta)}{p(X|\theta)} &= \nabla_\theta \log p(X|\theta) \\ &= \nabla_\theta \log \left[p(\mathbf{x}_0) \prod_{t=0}^{T-1} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \theta) \right] \\ &= \nabla_\theta \log p(\mathbf{x}_0) + \sum_{t=0}^{T-1} \nabla_\theta \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \theta) \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \theta).\end{aligned}$$

Introducing an auxiliary variable $\mathbf{z}_t$ (the *eligibility trace*) allows this sum to accumulated, i.e.

$$\begin{aligned}\mathbf{z}_0 &= 0 \\ \mathbf{z}_{t+1} &= \mathbf{z}_t + \nabla_\theta \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \theta).\end{aligned}$$

Thus, $\nabla_\theta \log p(X^{(i)}|\theta) = \mathbf{z}_T^{(i)}$ and the estimator can be written

$$\widehat{\nabla}_\theta \rho(\theta) = \frac{1}{N} \sum_{i=1}^{N} r(X^{(i)}) \mathbf{z}_T^{(i)}.$$

Factoring the Reward

If the reward criterion (i.e. the mapping from the reward sequence $r_{1:t}$ to the overall value of the trajectory, $r(X)$) can be factored into a recursive form, i.e. there exists a function ϕ such that $r(\mathbf{x}_0, \ldots, \mathbf{x}_{t+1}) = \phi(r(\mathbf{x}_0, \ldots, \mathbf{x}_t), \mathbf{x}_{t+1})$, then the reward can similarly be accumulated:

$$\begin{aligned}r_0 &= 0 \\ r_{t+1} &= \phi(r_t, \mathbf{x}_{t+1}).\end{aligned}$$

Examples of recursive reward criteria are the average and discounted reward functions. This factoring of the reward function allows the estimator to be simply written

$$\widehat{\nabla}_\theta \rho(\theta) = \frac{1}{N} \sum_{i=1}^{N} r_T^{(i)} \mathbf{z}_T^{(i)}.$$

Biasing the Eligibility

This estimator relies on the ability to execute multiple trajectories, i.e. that the agent's interactions with the environment are episodic. This requires the agent to recognise when the trajectory has entered a recurrent state in order to update the gradient estimate. This is a problem for two reasons. Firstly, the variance of the estimator is related to the recurrence time (i.e. the time between

entering recurrent states) which typically grows with the size of the state space, rendering the approach inapplicable to high dimensional problems. Secondly, in partially observable problems the identification of a recurrent state cannot be assumed.

A key extension (introduced by [KYK95] and theoretically justified in [BB99a]) is to bias the gradient estimate. This both reduces the variance of the estimator and removes the need to identify a recurrent state, allowing estimation of the gradient of a POMDP from a single sample trajectory.

Techniques for biasing the gradient include truncating or discounting the eligibility trace. Discounting the eligibility is favourable because of the limited memory needed to store the eligibility trace. Discounting the eligibility trace changes the update equation to

$$\mathbf{z}_{t+1} = \beta \mathbf{z}_t + \nabla_\theta \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \theta),$$

where β is a pseudo discount factor called the *eligibility decay rate*. The intuition behind this discounting is that the *mixing time*[1] determines the effective recurrence time and, by using a discount factor, the gradient can be estimated over this effective recurrence time [BB01]. An intuitive interpretation of β is that it defines the time scale at which actions affect future rewards. In [BB01] it is suggested that β should be chosen such that the mixing time of the Markov chain is greater than $1/(1-\beta)$. A consequence is that for immediate reward episodic tasks where the mixing time is effectively 1, β can be set to zero and the eligibility trace simplifies to the likelihood ratio for the current transition.

Biasing the eligibility allows the estimator to be updated at every step, i.e.

$$\widehat{\nabla}_\theta \rho(\theta) = \frac{1}{T} \sum_{t=0}^{T-1} r_t \mathbf{z}_t.$$

Decaying the eligibility trace is essentially a weak assumption about the effect of actions on the world. The eligibility decay is not related to the discount rate used for the discounted reward criterion in value-function learning. While both exponentially reduce the assignment of reward to actions, here it is due to the decaying effect of actions on future reward as opposed to the discounted value of reward. In partially-observable environments, the reward is assumed to be mediated by hidden processes. This eligibility trace is similar to those used in temporal difference methods, but here it accounts for the eligibility of the parameters rather than of the states or actions.

Other possibilities for the eligibility filter exist. For example, it is easy to imagine POMDPs where actions tended to affect reward after a certain lag, in which case a reasonable eligibility filter would be a non-zero mean Gaussian. A meta-learning approach could be to parameterise the eligibility filter and learn these parameters.

The division of a trajectory into episodes which allows for the eligibility trace to be reset to zero can be viewed as an eligibility filter which makes the assumption that actions only effect the reward in their own episode.

[1] Informally, the mixing time is the average number of steps needed to be taken starting from any state such that the last state is distributed according to the stationary distribution.

Substituting Actions

While this gradient derivation has used $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \theta)$, this distribution is hidden to the agent as the agent does not have access to the distribution model and, even if it were available, the hidden state is not observable. However, the use of a stochastic policy creates a separate Markov chain dependent on the control parameters θ. Thus, the transition probability can be substituted throughout the derivation with the probability of the action $p(\mathbf{a}_t|\mathbf{y}_t, \theta)$, changing the eligibility update to

$$\mathbf{z}_{t+1} = \beta \mathbf{z}_t + \nabla_\theta \log p(\mathbf{a}_t|\mathbf{y}_t, \theta).$$

This is a crucial benefit of policy-gradient learning as it means optimisation is possible irrespective of any knowledge of the underlying trajectory, circumventing any dependence on the complexity of the dynamics or its observability.

Offsetting the Reward

It has long been shown empirically that the use of a reinforcement comparison scheme, i.e. offsetting the reinforcement signal with a constant baseline, can vastly affect learning performance in reinforcement learning problems [SB98, Wil92]. Using a baseline in this derivation changes the estimator to

$$\widehat{\nabla}_\theta \rho(\theta) = \frac{1}{T} \sum_{t=0}^{T-1} \delta_t \mathbf{z}_t,$$

where δ_t is the current reward offset by the baseline b_{t-1}, i.e.

$$\delta_t = r_t - b_{t-1}.$$

The effect of an appropriate choice of baseline is to reduce the variance of the estimator. The long-term average of reward (over all states and actions) was originally used in [KYK95] and has since been proved to be the optimal variance minimizing constant [WT01]. This average can be accumulated online as follows:

$$\begin{aligned} b_0 &= 0 \\ b_t &= b_{t-1} + \frac{1}{t}\delta_t. \end{aligned}$$

In fact, any function of state can be added to the reinforcement signal without biasing the gradient estimate. Using this fact, an extension of reinforcement comparison is to concurrently learn an approximate value function (a *critic*) and to use this as an adaptive reinforcement baseline. The resulting *actor-critic* methods aim to reduce variance. While the use of linear function approximators still maintains convergence [SMSM99], there is evidence that using linear function approximation may actually delay convergence speed [GU01].

Using a reinforcement baseline has another effect. In stochastic gradient ascent, as soon as the parameters approach a stationary point, the parameter trajectory starts zigzagging and convergence decreases. Thus, in order to obtain convergence the parameter updates must become smaller as the parameter vector approaches the optimum. This implies that either the gradients or

Given:
Eligibility discount rate $\beta \in [0, 1)$
Step size α
Trajectory length $T > 0$
Initial parameter values $\theta_0 \in \mathbb{R}^K$
Set $\mathbf{z}_0 = \mathbf{0}$ ($\mathbf{z}_0 \in \mathbb{R}^N$)
Set $b_0 = 0$
for $t = 0$ to $T - 1$ **do**
  Observe $\mathbf{y}_t$, generated from $p(\mathbf{y}_t|\mathbf{x}_t)$
  Generate action $\mathbf{a}_t$ from $p(\mathbf{a}_t|\mathbf{y}_t, \theta_t)$
  Observe r_{t+1} (where next state $\mathbf{x}_{t+1}$ generated from $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$)
  Set $\delta_{t+1} = r_{t+1} - b_t$
  Set $b_{t+1} = b_t + \frac{1}{t}\delta_t$
  Set $\mathbf{z}_{t+1} = \beta\mathbf{z}_t + \nabla_\theta \log p(\mathbf{a}_t|\mathbf{y}_t, \theta_t)$
  Set $\theta_{t+1} = \theta_t + \alpha\delta_t\mathbf{z}_t$
end for
return θ_T

Figure 3.1: **Policy-Gradient Learning Algorithm**.

the learning rates must vanish in the vicinity of the optimum. Using a reinforcement baseline has the side-effect of operating like a step-size reduction schedule. This is because, as learning proceeds, the baseline will be a strictly increasing constant. Thus, the constant baseline, combined with monotonic improvement in average reward, has the effect of reducing the learning rate as learning progresses.

3.2.2 The Resulting Algorithm

Figure 3.1 shows the resulting algorithm which is known variously as the *Stochastic Gradient Ascent* [KYK95] or OLGARB[2] [WT01] algorithm.

In practice, with continuous-time systems the lock-step updating implied by the above algorithm is not necessary. The algorithm can essentially be split into separate processes of updating the eligibility trace whenever a new action is sampled from the policy, updating the gradient and baseline whenever reward is received, and updating the parameters whenever the gradient is accurate (or, in the online version, at every interaction).

Intuitively, the algorithm can be understood as addressing credit assignment via heuristics. The conventional eligibility trace heuristics are *recency* (where more credit is given to more recent events) and *frequency* (where more credit is given to events that have occurred multiple times). The added heuristic with stochastic policies is *surprise* which gives more credit to those actions which are more unlikely to have been taken (this results from the weighting of the gradient by the inverse of the probability of the action).

[2] OLGARB is equivalent to OLPOMDP [BB99a] with a baseline.

Another intuitive interpretation is the physical view of learning, where the policy parameters represent the position of a particle, the eligibility trace is the velocity, the decay rate is a drag force and action selection consists of applying an impulse (an instantaneous change in velocity).

Policy-gradient learning can be considered a generalisation of backpropagation to tasks with distal error. Instead of immediately assigning the error signal to the parameters, a trace of which parameters caused the actions is maintained, and then used to assign credit to those parameters once reinforcement arrives.

3.3 Policy Parameterisation

There are several requirements of a policy for use in policy-gradient algorithms: the policy must be differentiable (and thus continuous) with non-zero support everywhere.

The choice of policy parameterisation is largely problem dependent. As with all parameterisations for search, the issue becomes one of choosing a suitable parameterisation which induces a performance surface with nice features for the particular search algorithm. In this case, the performance of gradient ascent depends upon the smoothness of the objective surface. If we are solely interested in the infinite-horizon performance of the learning algorithm, then a unimodal objective surface is desirable. In realistic problems, however, a trade-off between the performance of the final parameters and learning time will exist so a multi-modal surface with many lower local optima may be preferable. Due to the complex effects of the parameters on the surface, however, it is difficult to see how such characteristics of the objective surface could be predicted *a priori*.

Unless there is problem-specific knowledge to be included, an initial unbiased policy will be the maximum entropy policy which assumes the least about the distribution of actions. As a local optimisation algorithm is being used, the solutions it finds will depend on the choice of initial policy parameters. The stochasticity in the environment and the action selection and the use of online learning means that multiple runs with the same initial parameters will not converge to the same point.

3.4 Discussion

As a Monte Carlo method, policy-gradient learning restricts optimisation to reachable beliefs and concentrates effort where it is needed, allowing optimisation to occur in high dimensions. The other main advantage is the guaranteed convergence to a local optimum.

The main problems with policy-gradient learning are the high variance of the gradient estimate and the combinatorics of search in large state spaces. These result in a large numbers of samples needed to estimate each gradient

accurately making learning very slow[3].

3.4.1 Exploration

This slow learning time is related to the exploration strategy implicitly used by policy-gradient learning. Exploration is implicitly defined by the stochastic nature of the policy combined with the search method for the parameters. Gradient ascent ignores any exploration in parameter space, so any exploration is solely due to the random perturbation of the actions.

In general, this reliance on stochasticity results in undirected exploration which degrades to a uniform random walk unless some *a priori* information is known. This can take unacceptably long to discover interesting parts of the parameter space. In the initial stages of the search, undirected exploration is dominated by the *plateau problem*, where the surface consists of large level regions in which gradient ascent degenerates to exhaustive search. Acting randomly until some experience of the world is acquired presents problems where the reward function is largely uniform, with informative rewards in only a few states. In particular, a large class of optimal control tasks (those for which a random action is expected to worsen the performance) are intractable using any kind of undirected exploration. This is especially relevant to physical problems where a succession of random actions can have no discernible effect due to such features as inertia, momentum and friction.

The use of stochasticity for exploration is similar to the temperature parameter in simulated annealing. In policy-gradient algorithms, a high entropy policy will smooth the stationary distribution, allowing for gradient ascent to escape local maxima.

3.4.2 Extensions

Several extensions to the basic policy-gradient learning algorithm have been proposed.

For example, a better estimator is given by using the *natural gradient* [iA98] which takes into account the structure of the parameter space and has been shown to reduce convergence times [Kak02]. A recent result reveals a deep relationship between the natural gradient and the use of linear function approximation [PVS03]. Another approach is the PEGASUS algorithm [NJ00] which uses a deterministic random number generator to reduce noise in comparison of actions. However, relying on such a generator means the algorithm is only suitable for simulation and there is evidence that the policy can learn to rely on structure in the particular sequence of random numbers [Abe03]. Other extensions include the *Action Transition Policy Gradient* algorithm which reduces the variance in gradient estimates by limiting gradient estimation to action transitions [GU00a], and the use of policies with memory such as finite state controllers [AB02].

[3]The enormity of the learning times is highlighted by the fact that recently a supercomputer was built to perform policy-gradient learning simulation over days with agents in small, discrete spaces [Abe03].

While these (and other) policy-gradient formulations boast potential orders of magnitude faster convergence times, the consensus opinion is far from settled and the performance of the simple direct algorithm is far from understood.

3.4.3 Applications

As a relatively new class of learning algorithm, policy-gradient has been applied to relatively few tasks. These include standard control problems such as the mountain-car and call admission tasks [BWB99], robot tasks such as room entry and observation task [GKU03], autonomous helicopter control [BS01], games such as Tetris [Kak02], and simulated motor control tasks such as dart throwing [LCR03].

Chapter 4

Motoneuron Recruitment

While there is uncertainty about the nature of the adaptation processes underlying biological motor control, at the lowest level of the motor system it is known that each individual muscle is capable of generating a range of force magnitudes. This *force grading* process, which maps a desired force output to the activation of the individual neurons controlling muscle contraction, benefits both the accuracy and efficiency of movements.

While the physiological process controlling this force grading is well understood, and models emphasising the optimality of the process (in terms of error minimizing and information-maximizing properties) are well-developed [SWC+95], no work explaining the learning of the mapping appears to exist [Bar02]. This chapter proposes policy-gradient learning as a biologically plausible process for this adaptation.

The chapter begins with an overview of the relevant muscle physiology and a model of the force grading process. The need for adaptation is argued before showing how policy-gradient learning is a suitable model of this adaptation. A simple linear model is initially presented to demonstrate the process before testing on a more physically realistic model.

4.1 Background

All muscles in all organisms work in the same fashion. Each muscle is composed of many force generating elements: long, thin cells called *muscle fibres*. The fibres are arranged parallel to each other in *fibre bundles*, each of which is directly innervated by a *motoneuron* located in the motor cortex (the region of the brain that controls movement). A motoneuron together with its fibre bundle constitutes a *motor unit* and the aggregate of motoneurons controlling a muscle is the *motoneuron pool*. The link between motoneurons and fibre bundles means there is a direct relationship between motoneuron activity and force production.

The force generated by each fibre is proportional to its cross-sectional area. Because the fibre bundles are so small, the total force produced is only effectual when many motor units are activated at once. A simple discrete-time model

of the process by which these units are activated assumes that a pool of N motoneurons receives an excitatory input $y_t \in \mathbb{R}$ from higher areas in the motor cortex which is assumed to be positively related to the desired force of the muscle. This input reaches all motoneurons synchronously and activates a subset of the motor units. The force generated by the muscle f_t is a function of the weighted sum of the activation of its motor units, where the constant weights $\{w^{(1)}, \ldots, w^{(i)}, \ldots, w^{(N)}\}$ capture the different force-generating properties of the fibre bundles, i.e.

$$f_t = f(\sum_{i=1}^{N} w^{(i)} a_t^{(i)})$$

where $a_t^{(i)}$ is the activation of motoneuron i. The maximum force a muscle can apply is thus proportional to the summed cross-sectional area of all the fibres.

4.1.1 Force Grading

If any input were sufficient to cause simultaneous excitation of all motor units then the minimum force produced by the muscle would be too large for most purposes. In order to produce accurate movements and conserve energy, forces must be graded in response to the input to the motoneuron pool.

This force grading is the result of two processes: the large scale *recruitment* of motor units and the fine scale *firing rate modulation* of the motoneurons. Modeling each motoneuron as a traditional artificial neuron[1] with a soft threshold activation (such as the logistic sigmoidal function) captures these two processes. The thresholds of each neuron (i.e. the minimum excitatory input for activation to be significant) are distributed across the pool from small to large. This distribution of thresholds captures the recruitment aspect because the larger the excitatory input to the pool, the more elements become active. The linear region of the neuron captures the small range of force producible by the motor unit via firing rate modulation.

The combination of these two processes results in a continuously varying input signal producing a smoothly graded force response from the muscle as shown in Figure 4.1.

4.1.2 Adaptation

There are several reasons for why a plastic mechanism is necessary to adapt the parameters of the motoneurons.

Firstly, the optimal parameters are unpredictable from a genetic standpoint. The fibres are heterogenous (they differ in length, number and orientation) so the weights $\mathbf{w}$ (i.e. contribution of each bundle to the overall force) are neither uniform nor predictable. Furthermore, the optimal mapping is unpredictable as a number of state-dependent factors introduce nonlinearities into the mapping, distorting the view of the mapping being a fixed-gain linear amplifier of

[1] The activation of this artificial neuron encodes the average firing rate. This simplified encoding is justified because the level of activation of the muscle fibres is directly related to the frequency of the post-synaptic spikes.

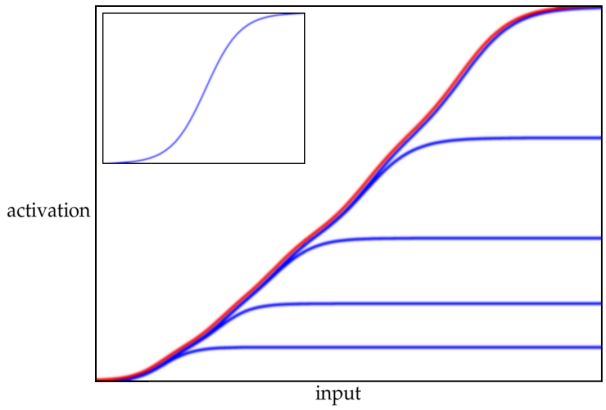

Figure 4.1: **Force Grading.** Smooth force grading in muscles is the result of motor unit recruitment and firing rate modulation processes. These processes can be modeled by the summation of individual sigmoidal units *(top left)* with different thresholds. This neural network can approximate any nonlinear amplifier of the input. Here, five units combine to give a near-linear response.

the input. For example, the net torque produced by a muscle depends on its mechanical advantage and thus varies with length. Thus, the optimal parameter settings depend on factors that can only be discovered through a process of trial-and-error interaction with the environment.

Secondly, even assuming the thresholds were predictable, the genetic information required to encode the parameters controlling the force grading for each of hundreds of motoneurons in hundreds of muscles would be huge and unnecessary considering they can be discovered by interaction with the environment.

Finally, even if the thresholds were predictable and genetically encoded, the optimal thresholds would be non-stationary. This is because many parameters of the system, such as body mass and muscle strength, change over time. Thus, the muscle parameters must be continuously adapting in order to maintain a robust mapping in the presence of a non-stationary environment.

4.1.3 Recruitment as Optimal Control

Motor control is often viewed as a hierarchy of systems from motor primitives down to motor units. The recruitment of motor units is an interesting behaviour of the motor system to investigate for several reasons.

Firstly, as the bottom layer in the hierarchy of motor behaviours, recruitment must form the basis for any motor learning system.

Secondly, because the motor units exist at such a low level the biological and physical constraints of the task are simple and well understood.

Thirdly, both motor units and muscles can be considered force-generating units. However, motor units avoid several complexities involved with the dy-

namics of multiple muscles. Muscle forces act along different axes, they may oppose each other, they have dynamic properties and they are heterogeneous. Conversely, motor units act along the same axis and always cooperate, simplifying the analysis of the task.

Finally, apart from any biological significance, the recruitment task is interesting as a study in arguably the simplest non-trivial learning task involving large number of degrees of freedom of an agent, as a study of how continuous control can result from the combination of discrete actions and as a task which is solved through the self-organisation of simple learning agents.

Tsetlin [Tse73] speculated that the motoneuron recruitment aspect of the force grading process could be modeled as a multi-agent reinforcement learning problem in which the task is to activate the right number of motor units to obtain a pull of a given force. In this task, the environment would evaluate the result of the collective behaviour of the entire pool of motor units based on the resulting force (see Figure [**?**]). Such a problem is free from any temporal credit assignment issues, as the reward is assumed to be related to only the most recent action. However the problem of assigning credit to each motor unit based on the collective behaviour still exists.

Policy-gradient learning is an appropriate method for solving this problem. The input is continuous so the alternative of using value-function methods would require the use of a function approximator, thus losing any optimality guarantees. In addition, the direct form of policy-gradient learning used in this thesis is biologically plausible. This has been demonstrated in [BB99b] where policy-gradient learning applied to integrate-and-fire neurons can be viewed as a form of Hebbian learning: a simple positive correlational rule of learning in which adjustments of weights are associated with nearly simultaneous firing of pre-synaptic and post-synaptic neurons.

4.2 Parameterisation

In order to test the use of policy-gradient learning for this force grading process, a number of assumptions are made to simplify the model while still capturing the main features.

The first assumption is that the finer force grading provided by firing rate modulation is ignored, allowing the output of each motoneuron to be represented as a binary quantity, i.e. $a_t^{(i)} \in \{0, 1\}$. In general, the use of a binary policy implicitly defines a decision boundary which splits the input space into two regions based on the probability of the policy's action. Learning amounts to adapting this decision boundary.

The second assumption is that the output of each neuron is stochastic and that this stochasticity is used by each neuron for exploration to find its optimal threshold. An appropriate function to capture this stochastic response is the logistic sigmoidal function. Thus, the probability of neuron i firing given excitatory input y_t is

$$p(a_t^{(i)} = 1 | y_t, \theta_t^{(i)}) = \frac{1}{1 + \exp[-\phi(y_t; \theta_t^{(i)})]},$$

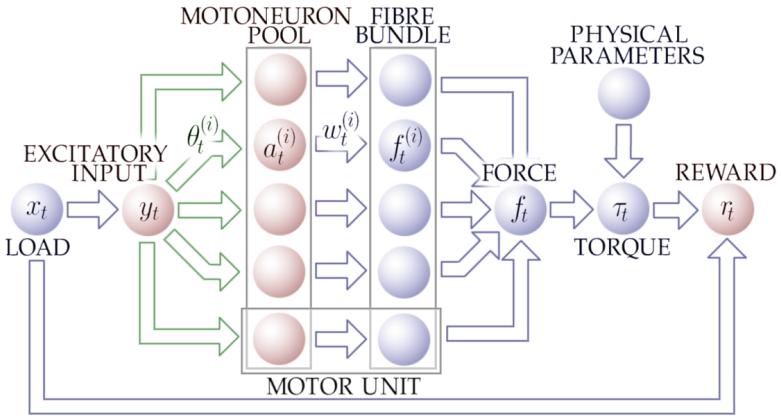

Figure 4.2: **Motoneuron Recruitment Task.** The motoneuron pool receives excitatory input which activates a subset of motor units in order to generate an appropriate force as defined by an external reward signal.

where $\theta_t^{(i)}$ are the parameters of motoneuron i and $\phi(y_t; \theta_t^{(i)})$ is the potential function of the neuron. For a single input neuron, this function is parameterised by two values, a gain k and threshold y^*:

$$\phi(y; \theta) = k(y - y^*). \tag{4.1}$$

The gain parameter (assumed positive to constrain the input to be excitatory) directly parameterises the level of exploration of the neuron. A zero gain yields a uniform probability of firing across the entire input space, while a large gain results in a near-deterministic hard threshold.

Because of this ability to arbitrarily approach a hard threshold, a network of these units (where the output is assumed to be the fraction of units that are activated) is general enough to approximate any deterministic one-dimensional mapping. As $k \to \infty$, the units lose their linear aspect and function as Heaviside step functions. Thus, the combined output approaches a piecewise constant function. Therefore, in theory, given enough units with a high enough gain, the family of functions defined by the ratio of neurons which are firing can approximate to any level of accuracy any deterministic increasing continuous function from $\mathbb{R} \to [0, 1]$.

The network as a whole, from excitatory input to force output, can be viewed as a single input, single output feedforward neural network with an adaptable stochastic sigmoidal hidden layer and a fixed deterministic output layer with unknown weights.

4.2.1 Factored Policies

This network is an example of a *factored policy*, i.e. a policy which can be partitioned into N different sub-policies, each depending exclusively on a subset of

the parameters, i.e.

$$p(\mathbf{a}|\mathbf{x},\theta) = \prod_{i=1}^{N} p(\mathbf{a}^{(i)}|\mathbf{x},\theta^{(i)}).$$

In general, the optimal behaviour for one subpolicy will depend upon the behaviour of the other sub-policies, because changing the parameters of one subpolicy effectively alters the environment's dynamics for the other sub-policies. Thus, in general, individually optimising sub-policies without regard for the others violates the assumption of non-stationarity in the environment.

However, the use of gradient ascent in policy-gradient learning circumvents this problem. This is because the gradient for each subpolicy with respect to the parameters of any other subpolicy is zero, i.e.

$$\frac{\partial}{\partial\theta_k} p(\mathbf{a}^{(i)}|\mathbf{x},\theta^{(i)}) = 0, \forall i \neq k.$$

This means that the gradient of the log probability is also zero and thus the policy-gradient for such factored policies is equivalent to the joint policy-gradient, i.e.

$$\begin{aligned}
\frac{\partial}{\partial\theta_k}\log p(\mathbf{a}|\mathbf{x},\theta) &= \frac{\partial}{\partial\theta_k}\log\left[\prod_{i=1}^{N} p(\mathbf{a}^{(i)}|\mathbf{x},\theta^{(i)})\right] \\
&= \sum_{i=1}^{N}\left[\frac{\partial}{\partial\theta_k}\log p(\mathbf{a}^{(i)}|\mathbf{x},\theta^{(i)})\right] \\
&= \frac{\partial}{\partial\theta_k}\log p(\mathbf{a}^{(k)}|\mathbf{x},\theta_k),
\end{aligned}$$

where $\mathbf{a}^{(k)}$ are the actions which depend on parameter k.

The consequence of this is that updating each subpolicy separately by gradient ascent is equivalent to jointly updating the entire policy (assuming the updates occur synchronously). Thus, each subpolicy can be optimised without communicating with the other sub-policies. In other words, a collective of agents updating their individual policies though gradient ascent learn to act so that the whole collective improves performance. This factoring of the policy gradient has recently been used for multi-agent coordination tasks [TBW01, PKMK00].

The network used here to model the recruitment task is an example of a factored policy as each parameter only influences the activation of a single motoneuron. Adding deterministic hidden units to this network, however, would induce a dependency between the sub-policies as the strength of each synapse between the input layer and the hidden layer would affect the potential of every output neuron. Thus, a feedforward network with a deterministic hidden layer is not a factored policy. Stochastic units breaks this dependence, allowing each stochastic unit to be optimised independently.

4.2.2 Gradient

It is easy to show[2] that the policy-gradient for a sigmoidal unit is

$$\frac{\partial \log a_A}{\partial \theta} = (A - a_1)\frac{\partial \phi}{\partial \theta}. \tag{4.2}$$

Thus, for a motoneuron with the potential function in equation 4.1, the policy-gradient is

$$\begin{aligned} \frac{\partial \log a_A}{\partial y^*} &= -(A - a_1)k \\ \frac{\partial \log a_A}{\partial k} &= (A - a_1)(y - y^*) \end{aligned}$$

Assuming $k > 0$, a positive reward will result in the threshold increasing if the neuron is off (depressing the probability of it firing in the future) and decrease the threshold if the neuron is on (making it more likely to fire). The gain parameter will increase whenever positive reward is received for input greater than the threshold and decrease otherwise.

4.3 Experiments

In order to test this parameterisation, experiments were initially performed on a simplified linear model before being applied to a more realistic physical model.

In both sets of experiments the effect of the output of the network is assumed to be instantaneous. Thus, the task is effectively an immediate reward task as the reward is independent of all actions except the current one. Therefore, the eligibility decay rate was set to $\beta = 0$.

In addition, the weights are assumed constant and of equal value (that is, the fibre bundles are considered homogenous), and the load is assumed to be a directly observable quantity, i.e. $y_t = x_t$ (an unrealistic simplifying assumption as the load is a hidden quantity to the motor system which can only observe height or strain).

4.3.1 Linear Mapping

In the simplest model of the force grading process, the optimal policy is explicitly defined as a linear mapping between the input load and the desired output force, i.e.

$$a_t^* = c_1 x_t + c_2$$

where a_t^* is the desired output and c_1 and c_2 are the parameters of the linear mapping. A quadratic reward function is used to penalise the controller's deviation from this input-output relationship, i.e.

$$r_t = -(a_t - a_t^*)^2.$$

[2] See Appendix A for details.

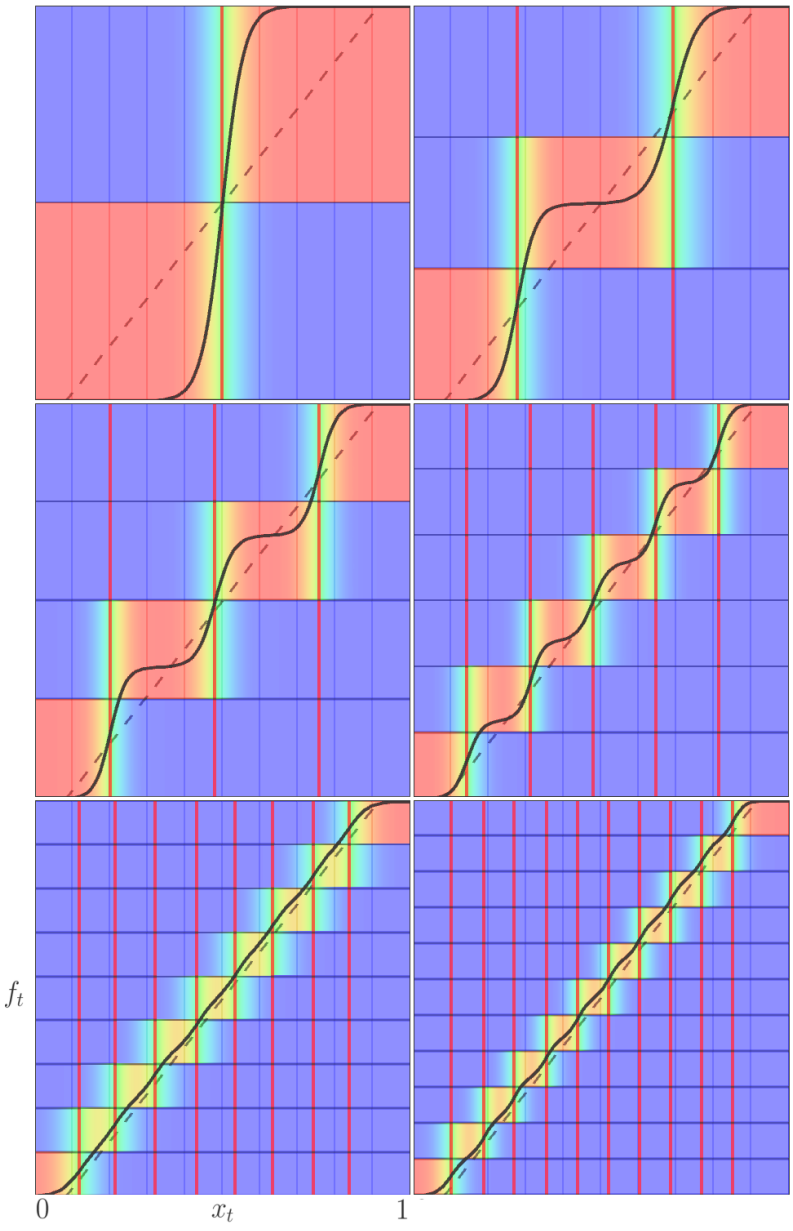

Figure 4.3: **Learnt Linear Policies.** This diagram shows the policies learnt to approximate a linear mapping given by the dotted line. The horizontal axis is the input, x_t, and varies from 0 to 1 and the vertical axis is the number of motoneurons firing. The red vertical lines indicate the motoneuron thresholds, the black curves indicate the expected output of the network, while the colour reflects the probability of that number of motor units firing (blue regions have zero probability, red have probability one and the green regions show the stochastic regions where the thresholds are being explored).

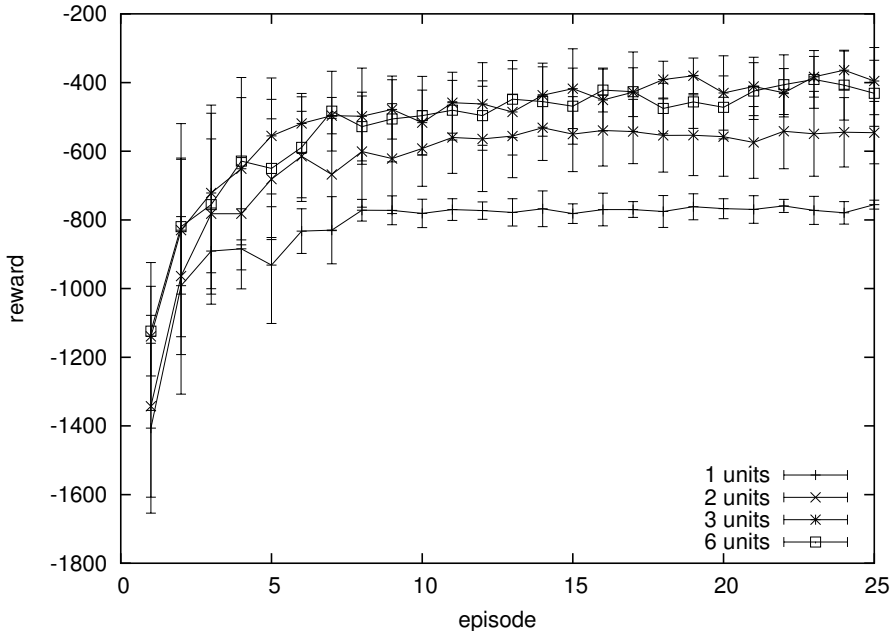

Figure 4.4: **Linear Mapping Performance.** This graph shows the mean of the performance for different numbers of units over 10 runs (the error bars show one standard deviation).

For these experiments, the desired mapping was given by $c_1 = -0.1$ and $c_2 = 1.2$. The initial parameters were set to $k = 0$ and the thresholds were randomly uniformly distributed in $[0, 1]$. The learning rate was set to $\alpha = 2 \times 10^{-4}$. At each interaction x_t was uniformly sampled from $[0, 1]$.

Figure 6.5 shows the policies converged to for a range of units. The policies converge to the optimal policy consisting of equally spaced thresholds with a moderate gain. Figure 4.4 shows the total reward for each episode during learning for different numbers of motor units.

4.3.2 Realistic Mapping

While the linear model demonstrates the ability of the policy to learn a mapping, it is not realistic for two reasons: it fails to model any complexity in the kinematics and dynamics of a real motor control task, and the reward signal is unrealistically informative. This section details the results of applying the same parameterisation to a more realistic model where the optimal mapping is implicitly derived from the dynamics of a realistic physical simulation and a plausible reward function.

This model consists of the simulation of a single degree of freedom arm which is actuated by a single muscle. The arm holds a load with variable mass and the goal is to apply the correct amount of force to lift this load. Applying

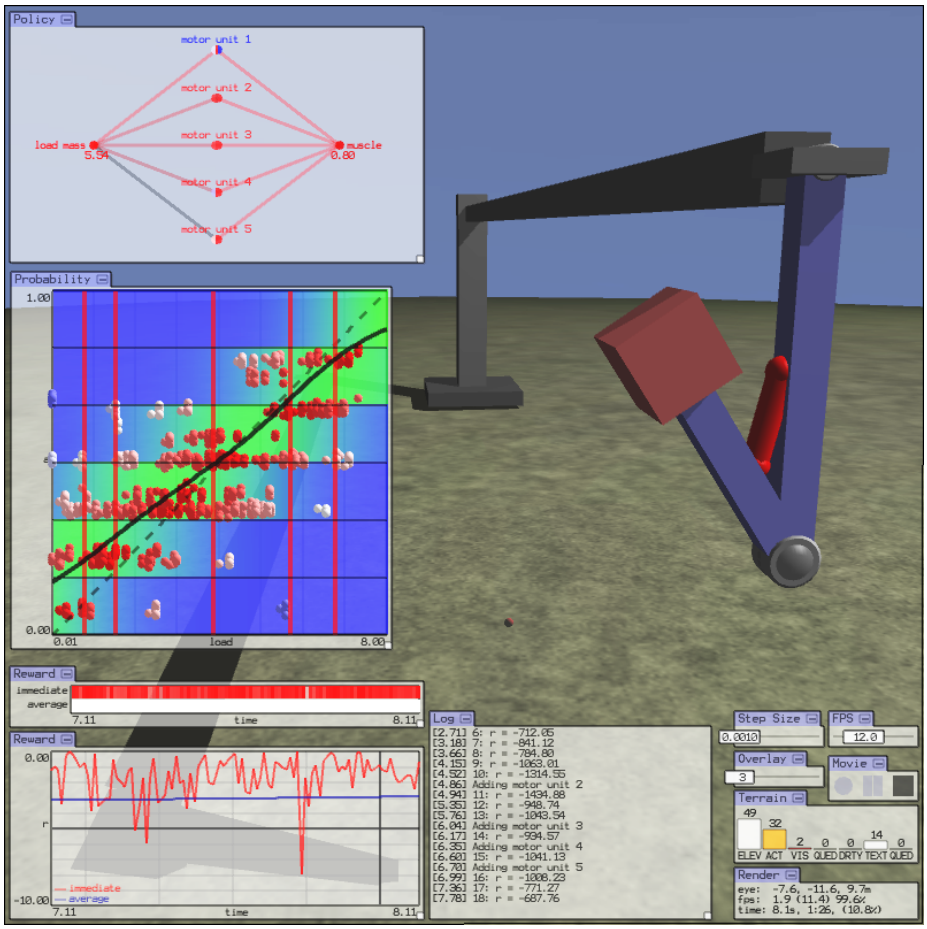

Figure 4.5: **Motoneuron Recruitment Simulation.** This diagram illustrates the configuration of the simulated arm used for the motoneuron recruitment task.

too little force fails to lift the load, while applying too much force is inefficient. A simple reward function which captures these goals is a weighted sum of the maximum height the load is lifted minus the force applied by the muscle, i.e.

$$r(x_t, a_t) = H(x_t, a_t) - C(a_t),$$

where $H(x_t, a_t)$ is the maximum height that load x_t is lifted given force a_t and $C(a_t)$ is a strictly increasing function of the muscle activation. Given this deterministic reward function, the optimal policy will balance the gain from lifting the load higher against the penalty for expending more energy to do so.

The simulation consists of two rigid bodies, representing the upper and lower arms, joined by a single degree of freedom revolute joint (see Figure 4.5). The upper arm is held by a harness at right angles which suspends it above the ground while the lower arm holds a load. A single flexor muscle is affixed to the skeleton at two points: the *origin* (half way up the upper arm) and the *insertion* (half way down the lower arm). As the muscle does not wrap around the joint, its path is simply a straight line between the origin and insertion attachment points. The force generated by the muscle is applied uniformly along this path. This is biologically plausible as the force generated by a muscle fibre is produced by a series of contractile elements called sarcomeres which are distributed approximately evenly across the length of the fibre.

The torque that results from the applied force depends on the tangential component of the force and the moment arm, i.e. the distance of the joint pivot point to the point where the force acts. The mechanical advantage varies with the sine of the angle between the moment arm and the force so that it is at its maximum when the joint is at a 90° angle and its minimum at the extremes of full flexion and full extension. The magnitude of the torque is always greater than zero as the muscle force is never parallel to the moment arm due to the offsetting of the origin and insertion points from the center of the skeleton[3].

Thus, the height function $H(x_t, a_t)$ depends on the mappings from the activation to the forces applied, from the forces to the torque and from the torque to the height lifted. In order to ensure a zero force output for non-excitatory input, negative values of x_t were included. The height function was defined for these inputs to output a constant value. Thus, the goal for a negative input is to minimise the number of motoneurons activated. Figure 4.6 shows samples of the height function for varying loads and activity levels. Figure 4.7 shows the reward surface and the corresponding optimal policy for the linear energy function $C(a_t) = 4a_t$. Because this task does not have delayed reward, this reward function completely determines the learning task.

The initial parameters for all motoneurons were set to $k = 0$ and the thresholds were distributed in $[0, 10]$. The learning rate was set to 2×10^{-4} for the gain and 4×10^{-4} for the threshold. The load was initially set to a minimum value of -2. At each interaction, the load was increased by an amount uniformly distributed in $[0.1, 0.3]$. Once the load was greater than 14, the load was reset to the minimum value and the parameters were updated based on the accumulated gradient estimate.

[3]In later experiments, this is also true of extensor muscles due to their wrapping around joint sockets.

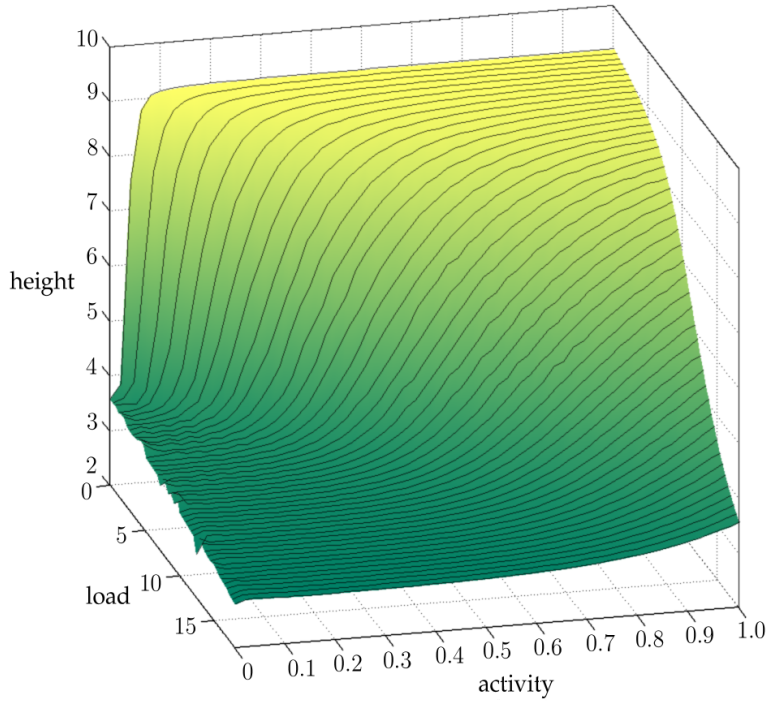

Figure 4.6: **Height Function.** Samples of the height function for varying load and muscle activity levels taken from the physical simulation of the arm.

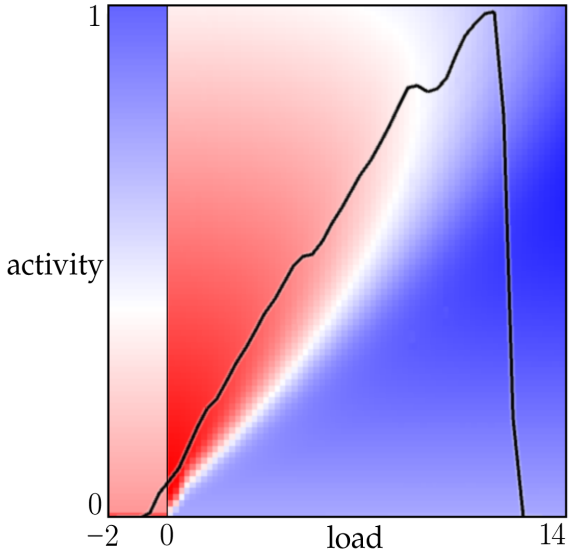

Figure 4.7: **Reward Surface.** The background colour shows the reward received for a range of loads and activity levels and the line shows the optimal activation for each load.

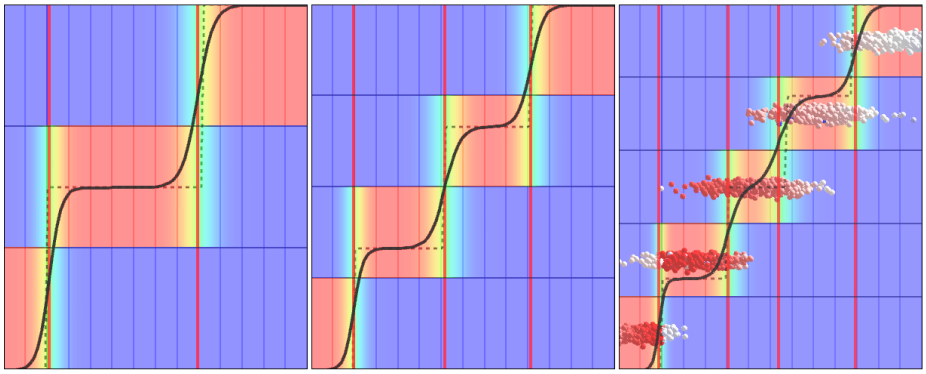

Figure 4.8: **Learnt Realistic Policies.** This diagram shows the policies learnt for the reward surface in Figure 4.7 for policies with two to four units. The optimal activations given by the dotted line is given by the maximum reward sampled by the policy. The right-most diagram shows recent samples of the reward taken under the policy (redder points correspond to greater reward). Other diagram elements are as in Figure 4.3.

Figure 4.8 shows the learnt policies for this task. Because the policy still contains stochasticity, the thresholds do not sit exactly on the line indicated by the sampled optimal mapping. This is because the reward on either side of the threshold is not necessarily equal (unlike the linear mapping where the reward function was stationary) so the optimal policy must take into account the risk of an unlikely action.

4.4 Discussion

The distribution of inputs has a large effect on the behaviour of the online learning algorithm. Two properties of the input distribution used helped prevent learning from converging to a suboptimal policy. These were ensuring that the loads in each episode covered the entire input space and waiting until the end of the episode to update the parameters.

Poor generalisation occurred if sampling was restricted to a subset of the input space. Specifically, if the region of negative input (corresponding to a desired force output of zero) was rarely sampled then one neuron would learn to stay permanently activated to ensure that some force is always output. This neuron would quickly achieve this by switching to a negative gain and then increasing its threshold. The result of these learning dynamics is that future sampling of negative input would not enable the neuron to correct as zero activation would never be output. In effect, this permanently restricted the range of activation.

Another problem associated with restricting input to large loads is that the policy would converge to a local suboptimal maximum at zero activation. The reason for this is that the addition of multiple binary random variables yields

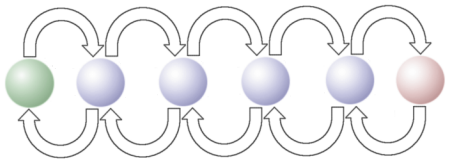

Figure 4.9: **Example** MDP**.** This deterministic MDP consists of a series of N discrete states (here $N = 6$) with two actions available in each state, enabling transition between adjacent states. An agent starting in the left-most state which uses an undirected random-walk exploration strategy will take exponentially many steps to sample the reward state on the right.

a binomial distribution with a variance inversely related to the number of variables. The consequence for a large pool of motoneurons is that, while each motoneuron may have a zero gain and be unbiased, the total force distribution will be sharply peaked and centred at 50% of the maximum muscle force. Therefore, any load requiring significantly more than 50% activation to lift is unlikely to sample the high reward received lifting the load, and thus learning will end up converging to zero activation.

This situation is prevented by ensuring the network is exposed to small loads, however. This is because the neurons learn a positive gain, meaning more excitatory input will result in a larger force, even if the larger input has never been received. This is a side-effect of the use of a global activation function.

The selection of an appropriate number of units is an important consideration. While including more motor units allows for a smoother force grading, it also results in slower convergence. From the perspective of each individual motor unit, its gradient estimate is being corrupted by the noise from the stochastic activation of other nearby units. This structural credit assignment is especially difficult in the initial stages of learning, where the entropy of each unit is maximised across input space. While it would be beneficial for each unit to update asynchronously with respect to each other, this would not yield the true gradient.

The potential impact of the use of a large number of units is illustrated by the MDP shown in Figure 4.9. This MDP consists of discrete states arranged in a line where a binary action transitions between adjacent states (either left or right) and reward is only received at the end of the line, i.e. if enough steps to the right are taken in a row. In this MDP random-walk exploration takes an exponential number of steps to receive reward. This MDP is essentially a sequential version of the recruitment task (in the motoneuron case, the length of the episode and size of the MDP depend on the force needed to lift the load). Each vector of neuron activations is equivalent to a fixed-length plan of action in the MDP as each uncorrelated random drawing of activations is equivalent to an episode of random walk behaviour in the MDP. This example demonstrates how temporal and structural credit assignment in the absence of any interme-

diate reward signal are equivalent, as timing information is not available to help with recency or causality.

A potential remedy to the combinatorial explosion of possible trajectories in this MDP is to use *macro-actions*, i.e. splitting the episode into a number of decisions less than the true number of interactions. The equivalent in the recruitment task is to use fewer units and make the output a ratio of the number of active units. This suggests that, because the appropriate number of units is not known *a priori*, a suitable strategy is to begin with a coarse resolution and keep increasing the resolution by adding more units. This method removes one free parameter (the number of units) and introduces a new one (when to add another unit).

Once the units have converged to an optimal policy, another unit can be added to improve the accuracy. Each additional unit forces the mapping to change less and less. For example, assuming a linear optimal policy, each unit in a network of N units only has to change its threshold by $\frac{1}{N(N+1)}$ of the size of the input space to accommodate an additional unit. While adding an extra unit initially affects the response globally across all input, as the gain of the unit increases, the effect is increasingly localised as shown in Figure 4.10. Assuming each unit has a high gain results in little noise from separated units. Thus, the output for each input is either near-deterministic or only has two possible activation levels. Thus, search is constrained to evaluating the relative performance of two possible actions compared with 2^N if each unit had zero gain.

While increasing the number of motor units and using the ratio of the outputs is not a biologically plausible process, it is equivalent to using a fixed number of units but forcing correlation between the activation of units (given a global learning rate and reward, they would update parameters similarly).

4.4.1 Extensions

There are several possible extensions to the models and learning algorithm used in this chapter.

Forcing Exploration

As learning progresses, the network converges to a deterministic policy, thus losing the ability to explore. As real-world tasks have physical parameters which change at long time scales, the policy will not be able to adapt to any change. Two options are available for artificially forcing exploration.

One option is to use a fixed step size, γ, for the baseline learning rate rather than using the true average weighting $1/t$, i.e. changing the baseline update to

$$b_t = b_{t-1} + \gamma\delta_t.$$

This would bias the learning to more recent experience which may prove useful in environments with a non-stationary drift of dynamics as it would track any change in average reward faster. A meta-learning approach might treat γ as a parameter to be optimised.

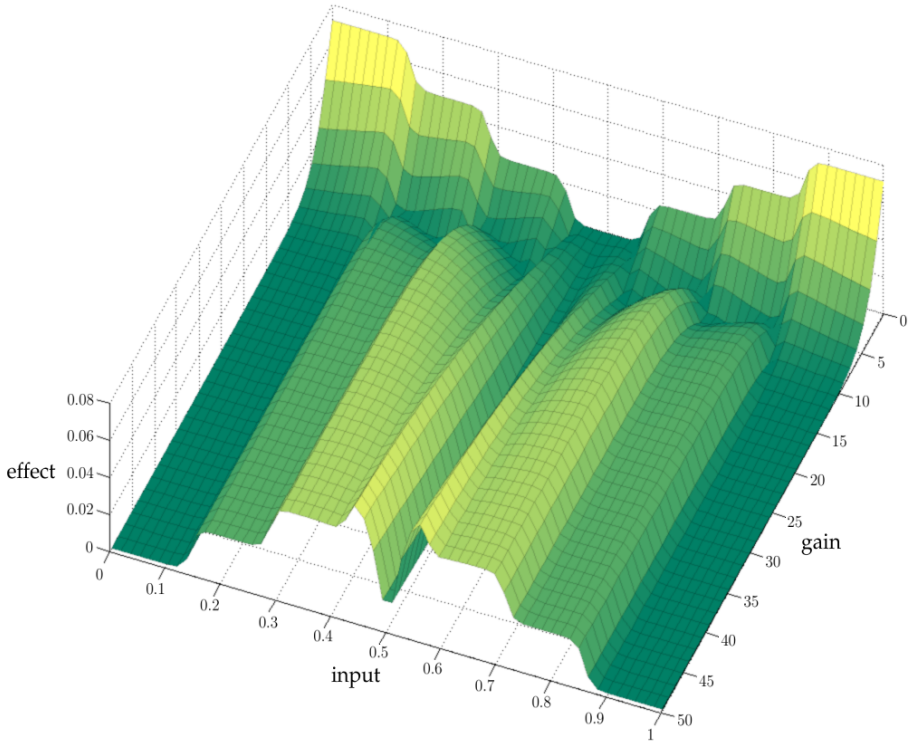

Figure 4.10: **Local Effect.** This diagram shows the effect on the policy of adding a new unit to a policy with 6 existing units. The effect of the additional unit is measured as being the absolute difference between the expected muscle activation without the unit and with the unit. At zero gain, the unit has the most impact at the extremes of the input space, but the effect becomes increasingly localised as the gain increases.

Another option would be to directly modify the parameters which control the stochasticity, i.e. for the recruitment task that would consist of slowing increasing the gain. This is analogous to the use of *weight decay* in supervised learning for complexity control.

Active Learning

In episodic tasks with a distribution over initial states, the trajectories which follow from some initial states will have converged whereas others will have not. For example, in the recruitment task, the load is sampled randomly but once the policy has converged to a deterministic response in a given region, receiving inputs in that region does not impact the learning. A possible extension which may decrease the learning time would be to allow the agent to select training examples by setting the distribution from which loads are sampled.

In these situations, learning would be improved by concentrating on the stochastic policy boundaries. This would allow it to restrict learning to unconverged areas thus decreasing the time to overall convergence. One option would be to sample initial states from the entropy of the policy, given by

$$\mathcal{H}(x_t; \theta_t) = -\sum_a p(a|x_t, \theta_t) \log p(a|x_t, \theta_t).$$

This relies on the heuristic assumption that stochasticity in the policy occurs in regions where the policy has not converged. This idea has similarities to the concept of boosting in supervised learning in which training examples are weighted in proportion to their current prediction error.

More Realistic Force Grading

The muscle force grading process was heavily simplified for the experiments in this chapter. A more realistic model of this process would include the effect of firing rate modulation in individual units. One way this could be achieved is by having each motoneuron output a continuous activation parameterising a linear gain by adding noise to a positive function of the input to the neuron.

Also, the experiments unrealistically assumed homogenous motor units, so permuting the units had no effect on performance. In reality, motoneurons come in an exponential distribution of sizes, where the excitability of a motoneuron (i.e. its threshold) is directly proportional to its size. The *size principle* describes the order of activation [SWC+95]. As the excitatory input to the pool grows, motoneurons are recruited in order by size from smallest to largest and de-recruited in the opposite order. A more realistic model would use heterogenous motor units with different force outputs to demonstrate the size principle resulting from the self-organisation of the units.

Chapter 5

Motor Learning Experiments

The previous chapter demonstrated the application of policy-gradient learning to a relatively simple immediate reward task. This chapter describes experiments of more realistic complex motor learning tasks featuring delayed reward, many inputs and multiple muscles.

Motor control tasks can be broadly categorised as either discrete movements such as reaching, lifting and grasping, or rhythmic movements such as running, swimming and scratching. The oscillatory dynamics of rhythmic movements suggests a suitability for the application of policy-gradient learning. In particular, the dynamics of such periodic tasks have a low mixing time allowing for frequent sampling of reward so that a gradient can be estimated quickly. They also typically lack any recurrent state which would admit episodic learning, thus suggesting the use of an algorithm which can learn from a single sample trajectory. In addition, the dynamics of such tasks are often complex and difficult to predict, suggesting the use of model-free learning.

This chapter begins with the extensions made to the muscle models to handle the simulation of such tasks, and then presents the results of experiments with learning the two periodic tasks of hopping and hexapod locomotion.

5.1 Muscles

The limitations of the muscle model used in the previous chapter include the simple geometry of the muscle path and the static force generating properties. This section describes the implementation of muscle wrapping and state-dependent force generating properties.

5.1.1 Muscle Wrapping

As muscles can only pull, they always operate in pairs in order to enable both flexion and extension of the skeleton. Pairs of monoarticular muscles (muscles which cross and affect only a single joint) consist of a flexor muscle which decreases the angle between bones on two sides of a joint, and an extensor muscle

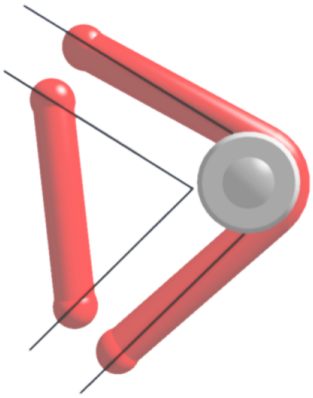

Figure 5.1: **Flexor and Extensor Muscles.** Models of monoarticulate flexor *(left)* and extensor *(right)* muscles.

which increases the angle (see Figure 5.1). Modeling the wrapping of the extensor muscle around the joint requires a more detailed muscle geometry than the straight-line path used for the recruitment task.

One such general muscle geometry is given by the *obstacle-set method* [GP00]. This method assumes that the muscle is affixed to the skeleton at the origin and insertion points and follows the shortest path between these points taking into account the various anatomical constraints such as bones and other muscles. In the case of a monoarticular muscle wrapped around a joint capsule, this results in a straight line between the origin and insertion except for highly extended positions in which it wraps around a spherical obstacle centred at the joint's center of rotation. For the experiments in this thesis, the muscles were always attached such that the shortest path lay in the plane tangent to the axis of joint rotation, thus simplifying the calculation to 2D and yielding a trivial 3D extension of the derivation in [Fag00]. For more details of the muscle geometry see Appendix A.

5.1.2 Lumped Parameter Force Model

In the recruitment task, the dynamic properties of muscles were ignored as the static force was the sole factor determining the immediate reward. In more detailed muscle force models such as lumped parameter models, however, the contraction force also depends on other factors such as the velocity of the muscle.

In these mechanical models, muscles contain elastic and passive elements that represent the stiffness of tendons and the passive tissue contributions to muscle force resulting in a spring-like force unit. The total force produced by the muscle is a sum of the force producing element and the passive elements, i.e.

$$f_t = f_t^{CE} + f_t^{SE}.$$

The elastic element generates a pulling force f_t^{CE} which decreases the further the muscle's length L_t is from its resting length r:

$$\begin{aligned} d_t &= \frac{L_t}{r} - 1 \\ f_t^{CE} &= a_t f_{max} \exp\left(-\frac{d_t^2}{k_{CE}}\right), \end{aligned}$$

where d_t is the normalised length of the muscle (positive when extended beyond resting length and negative when contracted) and k_{CE} determines the muscle shape. The experiments in this chapter used a constant $k_{CE} = 0.5$ and ignored any force grading in the muscle, thus assuming that the muscle acts as a binary actuator, i.e. $f_t^{CE} \in \{0, f_{max} \exp(-2d_t^2)\}$.

The passive element f_t^{SE} acts like a damper, resisting any change in length. The force it contributes is proportional to the rate of change of the muscle length $\dot{L}$, i.e.

$$f_t^{SE} = k_{SE} \dot{L}_t,$$

where k_{SE} determines the magnitude of the damping force.

Thus, a complete expression for the force is given by

$$f_t = a_t f_{max} \exp\left(-\frac{(L_t/r - 1)^2}{k_{CE}}\right) + k_{SE} \dot{L}_t$$

where expressions for L_t and $\dot{L}_t$ are given in Appendix A and f_{max}, r, k_{CE} and k_{SE} are the constant parameters that vary from muscle to muscle.

In such a model, the behaviour of the muscle is similar to a proportional-derivative controller with a set-point equal to the resting length of the muscle.

5.1.3 Muscles as Actuators

The use of muscles to provide actuation for a physically simulated agent yields several advantages over alternatives such as directly controlling the torque applied at a joint.

By ignoring any force grading muscles can be treated as binary actuators. Compared to using continuous actions, this simplifies the policy parameterisation. While simple to parameterise, the nonlinear, state-dependent torque output of muscles yields complex dynamics. In addition, any non-trivial task will require the coordination of many muscles raising the issue of structural credit assignment. Muscles allow the incorporation of prior knowledge into the control through adjusting the fixed parameters such as the points of attachment, the resting length and the coefficients. Understanding of the control is aided by the intuitive nature of muscles which helps in visualising the applied forces and the use of binary actions which helps in visualising the patterns of activation over time. Finally, the use of muscle-like actuators is biologically plausible.

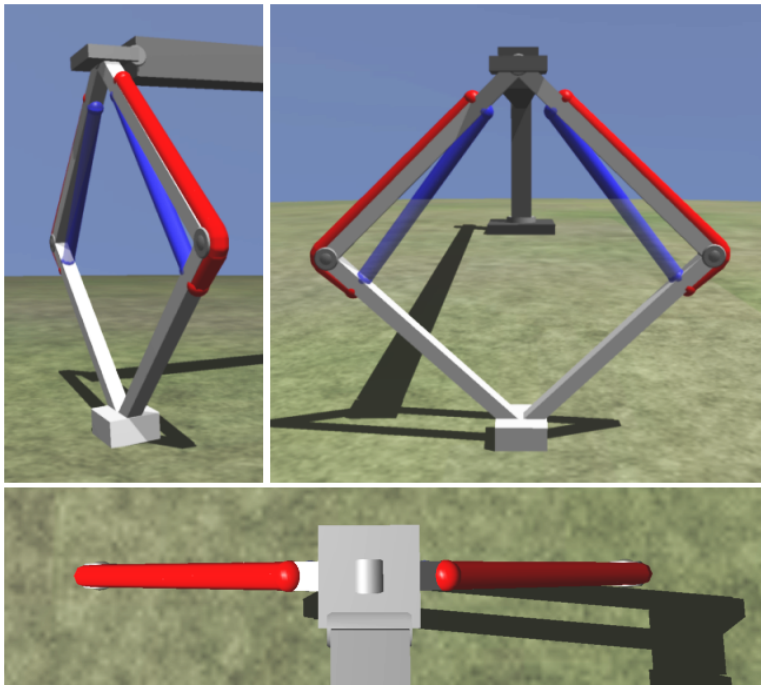

Figure 5.2: **Hopper.** This diagram shows the configuration of the agent for the hopping task.

5.2 Hopper

The first experiment was motivated by the desire to optimise a simple periodic low degree of freedom task which requires the use of opposing muscles. The task consists of an agent whose control allows it to contract or extend its body, allowing it to jump. The goal of the agent is to maximise the height jumped.

The structure of this hopping agent is a symmetrical skeleton consisting of four rigid bodies linked together in a closed loop via revolute joints (see Figure 5.2). The top bodies are joined to a harness which constrains its motion primarily to a plane. Thus, the agent effectively has a single controllable degree of freedom: the angle of its knees. This angle is indirectly controlled by a pair of opposing flexor and extensor muscles on each side of the body. Contracting the flexor muscles reduces this angle, while the extensor muscles act to increase this angle. A foot is attached to the bottom of the hopper to ensure flat contact with the ground.

The reward function used was a combination of the maximum height reached by the hopper on each jump and the muscle activity. Positive reinforcement proportional to the height was rewarded when the foot reached the apex of each hop. Negative reinforcement proportional to the number of muscles activated was given at each action selection step to penalise muscle activity. This was used as a form of complexity control to bias the policy search toward min-

imal controllers which generated fewer actions. In general, such a penalty acts as a cost function which forces the policy to generalise over smaller regions. The motivation for this is to ensure that the learnt policy did not consist of unnecessary muscle contractions. While a potential issue with this technique is that it may bias the learning toward a suboptimal policy which consisted solely minimising actions (i.e. rather than trying to jump, just sit still and conserve energy), this was avoided by ensuring the maximum penalty of 0.1 per muscle was much smaller than the reward for a good jump (over 10).

The agent's observations consisted of a 3-tuple $[c_t, \theta_t, \dot{\theta}_t]$, where $c_t \in \{-1, 1\}$ is a contact sensor attached to the foot, θ_t is the angle at the knee joint and $\dot{\theta}_t$ is its first derivative (multiplied by 0.02 to bring its effective range to within the range of the other inputs).

5.2.1 Policy

A simple 3-input, 2-output policy parameterisation was used, consisting of two sigmoidal hyperplanes, one for each pair of flexor and extensor muscles. This policy is effectively an extension of the policy used for the recruitment task extended to multiple inputs. The probability of the pair of muscles i firing is

$$a_1^i = \frac{1}{1 + \exp[-\phi(\mathbf{x}; \theta^{(i)})]},$$

where the potential function defines hyperplane i by weights w_i and bias b_i:

$$\phi(\mathbf{x}; \theta^{(i)}) = \sum_{j=1}^{M} w_{ij} x_j + b_i. \tag{5.1}$$

Using equation 4.2, the policy-gradient is thus

$$\begin{aligned} \frac{\partial \log a_A}{\partial w_{ij}} &= (A - a_1) x_j \\ \frac{\partial \log a_A}{\partial b_i} &= A - a_1. \end{aligned}$$

This simple policy has a limited representational capability as it can only represent linear decision boundaries. However, this was not a problem as the optimal behaviour is able to be generated by such a policy.

5.2.2 Results

The learning rate was set to 5×10^{-4}, the eligibility decay rate was 0.98, and the action period was 0.01 simulated seconds (the simulation time step-size was 0.001 seconds).

The choice of action period involved a trade-off: a short action period resulted in very slow learning (as it took a minimum number of successive extensor contractions to jump), while too long and the control was not able to respond quickly enough to a changing input. This trade-off is similar to the

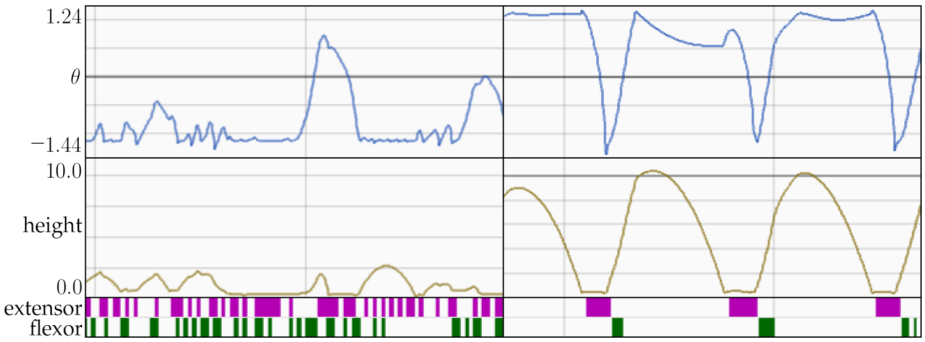

Figure 5.3: **Hopper Behaviour.** This figures shows the observations and actions of the hopper before *(left)* and after *(right)* learning. From top to bottom these are the hopper's angle, height and muscle activations.

problem of selecting the appropriate number of units in the motoneuron recruitment in order to balance the learning time and the accuracy of the mapping.

Figure 5.3 shows the difference in behaviour before and after learning. Figure 5.4 shows a post-convergence sample trajectory overlaying the probability of the flexor neuron firing. Figure 5.5 shows the sign of the weights for the policy that learning consistently converged to. The optimal behaviour consists of activating the extensor muscles while contracted to lift the load off of the ground, followed by activating the flexors to maximise the height of the jump's apex. Because there is some freedom in the lateral movement of the hopper, the activation of the flexor muscles has the added effect of ensuring that the foot makes contact with the ground at an appropriate angle.

5.3 Hexapod Locomotion

This section describes experiments with a more complicated motor control task involving legged locomotion. Locomotion is a natural motor learning task to investigate for several reasons. Firstly, an appropriate reward function (usually related to some measure of velocity or displacement) is intuitive, easy to measure and allows for frequent and smoothly graded feedback. Secondly, locomotion is a fundamental motor skill for animals and requisite for many other behaviours. Thirdly, locomotion allows for the study of adaptation under regions of differing dynamics by varying terrain.

The dynamics of locomotion in legged animals consists of raising the body above the ground and propelling it by a sequence of leg movements. During walking, each leg cycles between a *stance phase*, in which the leg is providing support and propulsion, and a *swing phase*, in which the leg is off the ground and swinging forward. Because the legs provide both support and propulsion and must be lifted after each stance, their movements must be coordinated so that the centre of mass of the body remains within a polygon of support formed

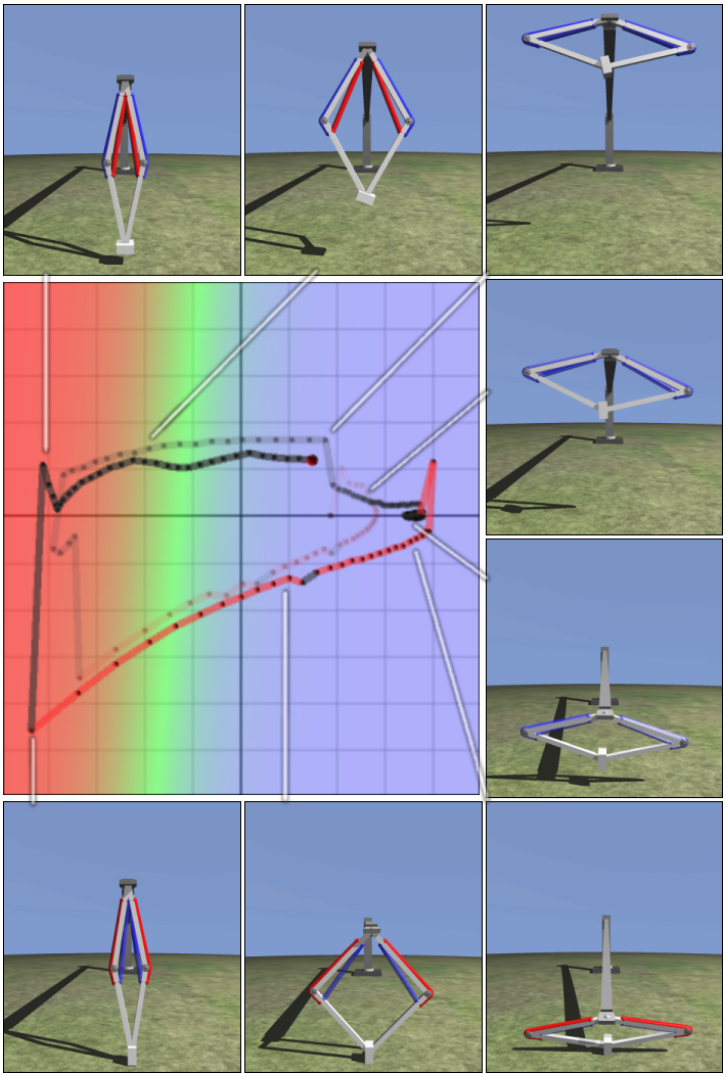

Figure 5.4: **Hopper Dynamics.** This diagram shows a typical sample trajectory of the hopper's observations after learning has converged. The horizontal axis is the knee angle, the vertical axis is the rate of change of the angle and red sections of the trajectory indicate contact with the ground. Each point on the trajectory is an individual transition in the trajectory. The background colour illustrates the probability of the flexor firing if the contact neuron is unactivated. The probability for the extensor is not shown as it is directly related to the status of the contact sensor. Likewise, the flexor is inhibited if the contact sensor is activated. Configurations of the hopper corresponding to the associated points on the trajectory are shown, with muscles shown red if activated and blue if unactivated.

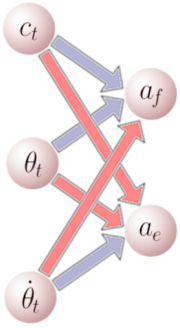

Figure 5.5: **Hopper Policy.** The learnt policy for the hopping task. Arrows indicate the weights between the sensors and the probability of the flexor and extensor muscles contracting (red for positive and blue for negative).

by the stancing legs. Another coordination problem is ensuring that adjacent legs do not interfere with each other.

5.3.1 Hexapod

The locomotion experiments used a symmetrical hexapod body which enabled the agent to remain statically stable for a variety of configurations. This removed the need to check for any stuck configurations (such as a collapsed body or interfering legs) and allowed learning from a single uninterrupted trajectory.

The hexapod model (see Figure 5.6) consists of three body sections of similar proportions connected via vertical revolute joint with $15°$ range. The legs are all equivalent except for their point of attachment to the body. Each leg is segmented into three parts: the first is connected to the body with lateral freedom with $60°$ range, the middle has vertical freedom with $70°$ range degrees of freedom and the foot segment has a $7°$ range. Each leg is controlled by four muscles (a flexor and extensor pair for each degree of freedom).

5.3.2 Gaits

The possible gaits for hexapod locomotion include tripod and pronking gaits.

In the tripod gait, the legs are divided into a pair of tripods. Each tripod consists of the front and rear legs of one side with the middle leg of the opposite side. Legs in the same tripod are always in the same (swing or stance) phase while legs in the other tripod are in the opposite phase. Thus, the tripod gait consists of the alternating movement of three legs.

The pronking gait consists of all legs cycling synchronously producing a jumping movement.

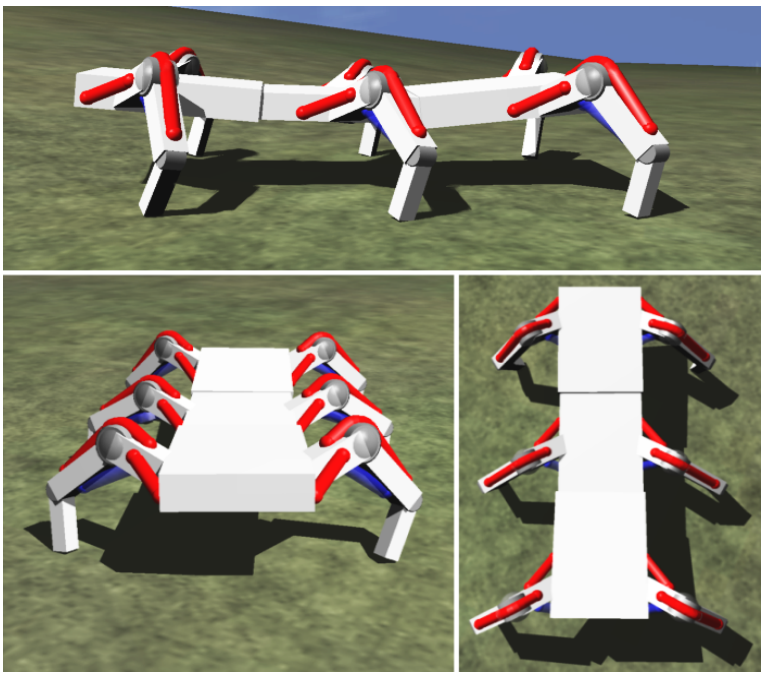

Figure 5.6: **Hexapod.** The hexapod model used for the locomotion experiments.

5.3.3 Finite State Controller

Initial tests of the ability of this model to walk discarded the use of the muscle actuator and instead used a fixed parameter finite state controller for each leg which resulted in a tripod gait by transitioning between stance, wait, swing and drop states as shown in Figure 5.7. The stance to wait transition occurred once the leg was sufficiently far back; the wait to swing transition occurred once opposing tripod had greater than one leg in the stance phase; the swing to drop transition occurred once the leg moved forward a given angle; and the drop to stance transition occurred once the foot made contact. Each state specified a desired angular velocity for the lateral and vertical degrees of freedom of the leg. The resulting controller was robust and consistently generated a tripod gait over varying terrain conditions.

Using prior knowledge to predefine a fixed controller was necessary because the hexapod's performance was heavily dependent on selecting appropriate physical, simulation and control parameters. These parameters included morphological parameters (such as segment size and density), joint parameters (location, degrees of freedom and range), material parameters (friction and elasticity coefficients), muscle parameters (placement, socket radius, maximum force, resting length and damping constant), simulation parameters (step-size, gravity and constraint solver parameters) and control parameters (action period).

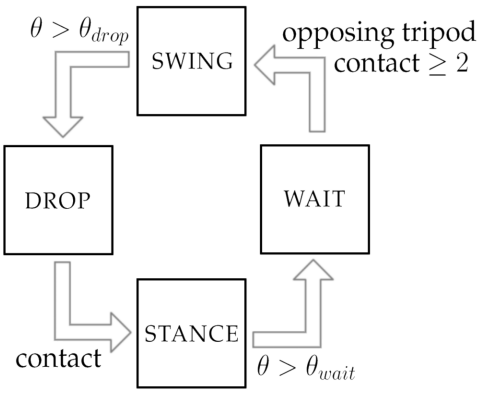

Figure 5.7: **Tripod Gait Finite State Leg Controller.** This diagram illustrates the finite state machine initially used to control each leg in the hexapod. The boxes represent individual states which dictate a desired angular velocity for each joint. The arrows show the preconditions for state transitions.

An attempt was made to use a simple random search algorithm to optimise these parameters. This involved generating new parameter samples by selecting from existing samples using a Boltzmann distribution over their average rewards, and perturbing a small random subset of the parameters with Gaussian noise. However, with a high-dimensional parameter space and lengthy sampling time, the approach was largely ineffective. Instead, it was found that using a manual trial-and-error procedure to tweak the parameters was sufficient. This involved measuring the maximum velocity of the tripod gait under the fixed finite state controller while observing how small perturbations to a subset of the parameters affected the dynamics. This effectively amounted to a manually guided hill-climbing search. While using a fixed controller is sub-optimal as it removes the ability to exploit the specific dynamics of the physical parameters, it reduced the time to sample the objective function by two orders of magnitude making the optimisation of non-control parameters possible.

This process resulted in a number of qualitative observations which were contrary to expectation. For example, a wide heavier base produced faster movement than a slimmer light one; using a vertical degree-of-freedom between segments was faster than a lateral degree-of-freedom; increasing the maximum force of the muscles past a point slowed it down; and increasing the power of the lateral muscles without a matching increase in the power of the vertical muscles slowed it down. An explanation for these observations seems to be that the slower adjustments all resulted in less apparent foot contact time, thus reducing the time available to apply the pushing force in the leg's stance phase. Specifically, increasing the base size reduced deviations of the frontal plane from the ground; a vertical degree of freedom allowed more contact with the ground; decreasing the maximum force reduced the risk of the hexapod becoming airborne; and the power ratio was necessary to allow the

foot to make contact before moving into the swing phase.

5.3.4 Gait Learning

After finding appropriate physical parameters, a reward function was set and policy-gradient learning was used to learn a gait. The reward function was proportional to the forward velocity of the hexapod, and the policy used were hyperplanes for each muscle where the inputs to the policy were the lateral angles of each leg. Initial experiments attempting to learn the activation of each of the 24 individual muscles were unsuccessful. It quickly became apparent that the structural credit assignment problem for such a large number of independent actions ($2^{24} > 10^7$ distinct actions) is intractable without incorporating prior knowledge, especially when sampling any abnormal reward required a number of correct consecutive actions.

To grasp the magnitude of the combinatorics of action selection, consider a task segmented into episodes, where in order for a non-zero reward to be given at the end of an episode (so that a gradient can be estimated), a certain fraction of the actions in each episode need to be correct or incorrect. Assuming each episode is only 5 interactions long, with 10 binary actions per interaction, then if at least 55% correct is required, it would take over 10^{10} episodes before it was more than likely to receive reward[1]. Thus, no unbiased stochastic search algorithm is tractable in such dimensions.

Instead of activating individual muscles, groups of coactivating muscles (or so-called muscle synergies) were used. This involved arranging the muscles into 4 groups consisting of the swing and stance phases for each tripod of the tripod gait, resulting in 16 distinct actions. Learning consistently converged to either one of two viable gaits. The learning parameters used were: $\alpha = 0.001$, $\beta = 0.97$, and the action period was 0.02 seconds. Figure 5.8 shows these two gaits. The pronking gait consisting of the coactivation of both tripods was learnt more or often than not. The tripod gait consisting of the alternating activation of the swing and stance synergies was also found.

5.4 Discussion

This chapter has presented preliminary results from applying policy-gradient learning to motor control tasks which require the use of multiple muscles.

Traditional approaches to optimal control are intractable with large numbers of input variables. For example, no value function with more than 6 input variables has ever been successfully approximated in non-trivial dynamic motor control tasks [Cou02]. In contrast, policy-gradient learning is useful for huge number of inputs as the computation of the gradient is a linear complexity operation in the number of inputs. Any inputs which do not correlate with abnormal reward have no effect on action selection.

[1]Calculated by integrating a Gaussian approximation to the corresponding binomial distribution.

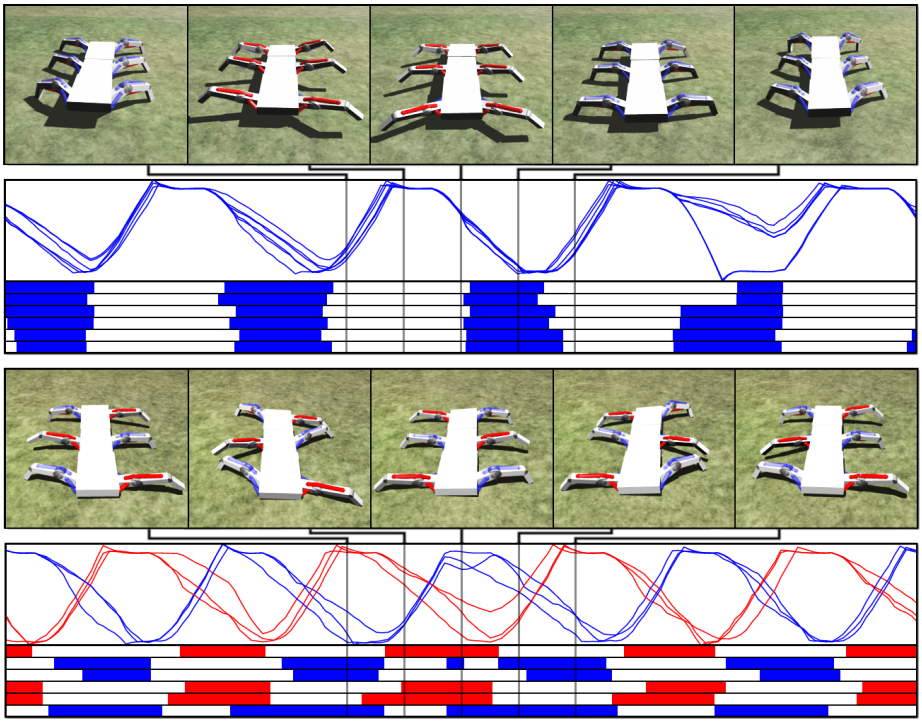

Figure 5.8: **Hexapod Gaits.** These diagrams show the two gaits learnt: the pronking *(top)* and tripod *(bottom)* gaits. The graphs show the lateral angle and foot contact for each leg (separated by colour into each tripod for the tripod gait).

While useful for problems with high-dimensional input spaces, the experiments in this chapter suggest that the structural credit assignment problem for high-dimensional control spaces combined with undirected exploration makes policy-gradient learning intractable without taking advantage of prior knowledge. Other difficulties include selecting an appropriate time scale for action selection and appropriate physical and simulation parameters.

5.4.1 Extensions

The experiments in this chapter suggest a number of possible future extensions.

Motor Primitives
Some means of tackling the combinatorics of action selection in realistic problems is necessary to scale learning to non-trivial tasks. This requires reducing the dimensionality of the control space without detriment to the optimality of learning.

One means of reducing the dimensionality of the motor control problem, motivated by neuroscience evidence, is through the use of *motor primitives* [MIGB94]. These building blocks of movement combine to form a variety of complex movements. Examples of such primitives include the muscle synergies used in this chapter.

The form that these primitives might take is the subject of much research. While the use of muscle synergies in this chapter was motivated by prior knowledge about the task, in general, little is known about the methods by which appropriate primitives might be generated. Any automated means of discovering useful primitives would seem to involve circular reasoning because, while intuitively it seems necessary to solve the task in order to find appropriate primitives, these primitives are necessary in order to solve the task. Recent work in attempting to automatically discover primitives in [TG03] uses unsupervised learning to extract regularities in the sensory-motor system.

In essence, motor primitives are a form of modular or hierarchical control. Assuming the primitives are given, policy-gradient algorithms are ideally suited to such controllers as they are applicable to partially observable processes. An interesting extension would involve simultaneously adapting the parameters controlling the movement specified by the motor primitive and the primitive's preconditions.

Biomimetic Hexapod
The hexapod model used for these experiments is a simplification of biological hexapod locomotion. More realistic biologically-inspired physical models would result in gaits which utilised the specifics of the dynamics such as the compliancy of legs.

Preliminary experiments with such a model were made (see Figure 5.9). These quickly illustrated the reliance on selecting correct morphological and physical parameters. For example, a number of issues relate to the fact that the legs have different functions. The force outputs of the middle-rear tripod pair

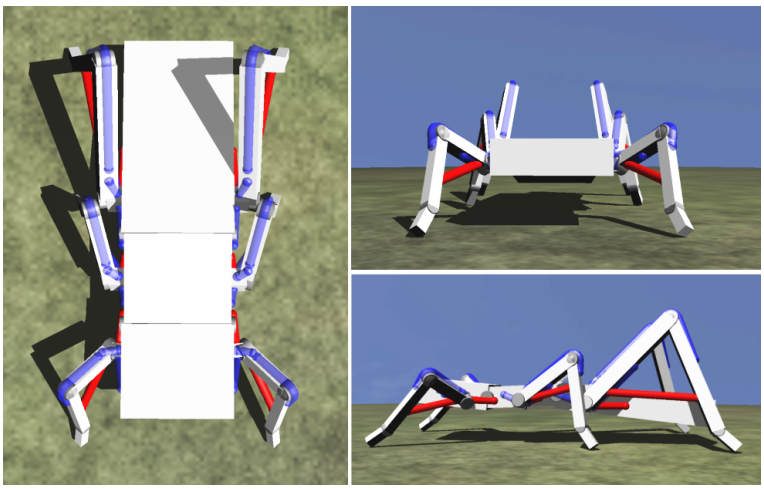

Figure 5.9: **Biomimetic Hexapod.** A biologically-inspired hexapod model.

need to be carefully timed in order to avoid the resultant force from imparting a rotation on the body and causing the agent to yaw. The front legs must have low friction and be clear during the swing phase so that they avoid interrupting the stride prematurely. Also, the friction in the middle and rear legs must be high enough to avoid slipping when the force is applied.

The complex relationships between the morphological parameters suggest that some form of outer loop (such as a global optimisation method) is needed in order to obtain a structure capable of producing the desired behaviour.

Stride Period Adaptation

A simple control theory approach for the stride period adaptation of running hexapods over varying terrain slopes is presented in [CKCC01]. This approach consists of assuming some desired period between footfalls and adjusting the stride period (i.e. time between swing and stance phases of the tripod gait) of an open-loop controller in order to slowly adapt this period and optimise the speed of the hexapod.

An attractive alternative to this hard-wired control law would circumvent prespecifying the desired footfall period and instead learn the control directly from the hexapod's experience across different terrain. This could be achieved by using policy-gradient learning to adapt the parameters of the stride period controller.

Chapter 6

Local Policy Learning

In any parametric learning problem, the particular class of representation used has a huge bearing on the behaviour and success of the learning algorithm. While the parameterisation used in the recruitment experiments was chosen to model a specific task, in general there are few constraints on this choice. The representational power of the parameterisation is seldom an issue as all but the simplest of function approximators are capable of representing arbitrarily complex nonlinear functions. However, attributes such as speed of convergence and applicability to high dimensional problems depend significantly on the parameterisation.

A relevant consideration is the property of *spatial localisation*. This chapter argues the benefit of using spatially localised policies in online control learning, introduces such a parameterisation which is shown to learn in moderately complex, linearly inseparable problems and applies a constructive algorithm to adapt the network architecture.

6.1 Local Policies

Spatial localisation is a characteristic of the region of response of a function approximator. *Local* approximators use activation functions which give a significant response only in a neighbourhood of the input space of the function. In contrast, spatially non-localised (or *global*) approximators use activation functions that are not limited to a finite domain.

A typical example of a global approximator is a feedforward neural network with a linear sigmoidal activation function. Examples of local approximators include the class of *radial basis functions*, which are the additive combination of a number of *radial functions*: functions where the response decreases monotonically with distance from a central point, i.e. $f(\mathbf{x}) \propto \phi(\|\mathbf{x}-\mathbf{x}'\|)$ where ϕ is a strictly decreasing function. A typical example of a radial function is the Gaussian, where $\phi(x) = \exp(-\alpha x^2)$. See Figure 6.1.

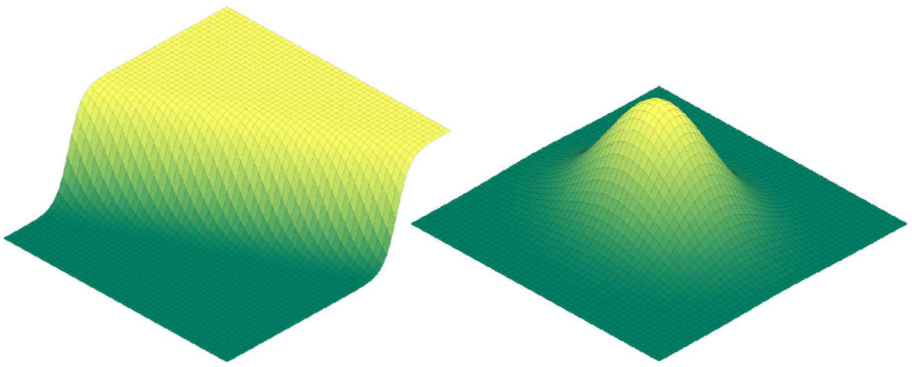

Figure 6.1: **Global and Local Approximators.** Examples of simple two dimensional approximators. *Left*: a global approximator (a linear sigmoidal boundary), *right*: a local approximator (an axis-aligned Gaussian).

6.1.1 Advantages of Local Policies

While most current research in machine learning is concentrated on global function approximation, there are a number of reasons to suggest that the use of a localised representation is better suited to online policy-gradient learning.

Catastrophic Forgetting

Global approximators generalise their response by extrapolating to areas outside of the observed data. A consequence of this extrapolation is that, in online supervised learning, global approximators are prone to *catastrophic forgetting* [Fre94]. Catastrophic forgetting is the phenomenon of unlearning which occurs when an approximator is trained on a restricted area of the input space and focuses too much on learning this area, forgetting to maintain previously learned areas. Catastrophic forgetting also occurs in control learning problems, and is potentially even more of an issue as the sampling distribution changes during learning. There is experimental evidence for catastrophic interference in human motor learning tasks [BKST95].

Techniques for fighting catastrophic forgetting in a supervised learning framework while retaining the use of global approximation include manipulating the training set (usually by permuting it), using batch instead of online learning or switching to a non-parametric method. These solutions are inapplicable to control learning.

Local approximators, on the other hand, learn through interpolation. They are not prone to generalising beyond their experience as learning in one region cannot affect another. Thus, they are not afflicted by catastrophic forgetting.

Representational Power

Even the simplest of local approximators are universal approximators. As such, given enough units they can approximate any function to arbitrary precision. Thus, unlike simple global policies such as the hyperplanes used in the

previous chapters, they are able to represent complex decision boundaries and solve linearly inseparable problems.

Transparency
Parameters have a more transparent meaning in local approximators than in global approximators. In global approximators such as multi-layer neural networks, the effect on the output of a change in parameters is not obvious as the response depends on the interaction of many parameters. In contrast, the response of a local approximator intuitively depends on only a few local parameters.

Because of this, it is both easier to incorporate prior knowledge into an initial local policy and extract intuition from a learnt local policy.

Biological Motivation
Local policies have an intuitive connection with the functioning of biological motor systems. A necessary component of generating coordinated movement is selectively sensitizing appropriate reflexes while suppressing others. One framework for such coordination is *schema theory*, in which motor control is explained as the coordinated coactivation of adaptable *motor schemas* or component behaviours which store how to react to a set of circumstances. The use of local factored binary policies represents the intuition of reflexes being activated in different regions of the input state and layering upon each other to modulate the existing dynamics. The task of motor learning amounts to adapting the preconditions which trigger the individual superposing reflexes.

6.1.2 Related Work

While the use of localised policies for policy-gradient learning is novel, the benefits of using local approximators and modular representations for learning are well known.

Local Approximators
Local function approximators have been used in value-function learning for their local generalisation properties with varying degrees of success [Sut96]. A popular approach is the CMAC local function approximation architecture [Alb75] which is a sparse, coarse-coded approximator, similar to a fixed radial basis function network. In [KA97] a similar approximator to the one used in this chapter is adapted by backpropagating the temporal-difference error. However, they observe that the method is not particularly effective as minimising this error is a different goal from optimising the control. In general, this is true of the use of any function approximator for value-function learning.

Local approximation has met with its greatest success in control problems in the learning of inverse models [AMS97].

Modular Controllers
Local policies can be interpreted as a modular controller where each basis function defines a sub-controller. The benefit of using modularity as a way to tackle the complexity of learning problems is well recognised. Several methods for

learning modular control in motor learning have recently be proposed based on the concept of *paired predictor-controller* architectures, e.g. MOSAIC [WK98] and MMRL [DSKK02]. These architectures assume the environment is partitioned into mutually exclusive *contexts*. More formally, the underlying state in the POMDP is assumed to include some latent discrete variable which is strongly correlated with both the observed dynamics and the optimal policy. In such environments, accurately inferring the hidden state should both simplify the control problem and address catastrophic forgetting by allowing the use of separate controllers for each context.

At each time step a distribution over the current context is inferred. This inference is performed by a fixed number of predictors (one for each context) which learn to predict the dynamics. The resulting belief distribution softly partitions the state space into regions according to how well each predictor performs. Each predictor is paired with a controller, and the error for each predictor weights the output of the corresponding controller in the final control signal[1].

While motivated by the desire to combine model-based and model-free methods in a Bayesian manner (and also by biological evidence [FVJW03]), there are several unaddressed issues with such methods. Firstly, the number of modules (i.e. the number of hidden states) is a free parameter and is impossible to know *a priori*. Secondly, the optimality of additively combining the output of several controllers is questionable. Thirdly, and most importantly, the validity of the heuristic assumption of the existence of a useful correlation between the predictability of dynamics and the optimal policy is not transparently obvious. In fact, regions of varying, unpredictable dynamics do not necessarily imply complex control. In these architectures, complex dynamics along a simple decision boundary will divide control between different modules, splitting an easy decision into pieces, thus decreasing the amount of training data for each controller and increasing convergence time (see Figure 6.2 for an example).

While it is difficult to argue that these methods are suboptimal as there is no loss of generality in dividing the control in this manner, the convergence time will be adversely affected by splitting the training data across multiple controllers as the variance of the learning algorithm will increase. In addition, the converse assumption, that predictable dynamics implies simple control, is not necessarily true.

These methods converge to a partitioning of the input space in which only one sub-controller is active at a time. Other such one-of-many approaches include the *boundary-localised* [GU00b] formulation of policy-gradient learning where predefined deterministic controllers are activated according to mode boundaries which are learnt. In contrast, the local factored policies used in this chapter are an example of many-at-once controllers. Such controllers have the advantage that the complexity of the response and the degree of exploration can vary over the input space.

[1]This partitioning is similar to a *Mixture of Experts* [JJNH91] architecture, except that instead of learning a gating unit the responsibilities are derived directly from the prediction accuracy.

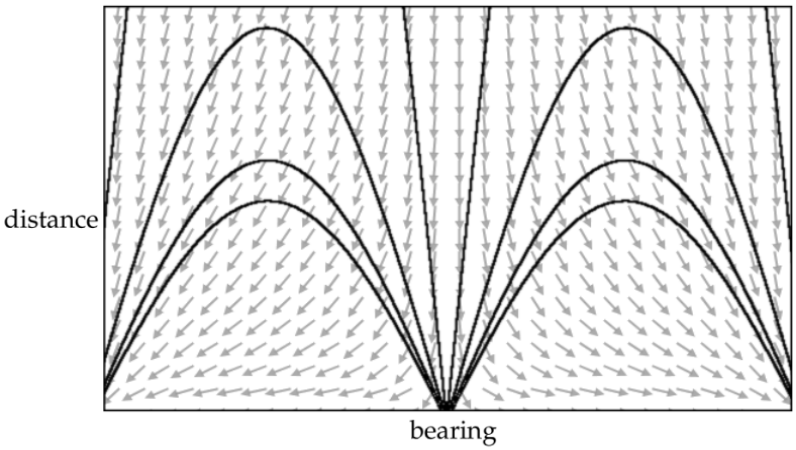

Figure 6.2: **Example of Input Space Partitioning.** This diagram shows the two-dimensional state space of an example task consisting of the distance and bearing of a target where the goal is to apply a binary action (such as braking) when the target is at a certain distance, i.e. the optimal policy is a simple horizontal linear decision boundary which ignores the bearing of the target. However, using N linear predictors partitions this boundary into N different regions. Here, 8 predictors (with equal change of distance and change of bearing equally distributed over $[-\pi/2, \pi/2]$) partition the input space into 8 regions (the arrows show the dynamics of the observations and the solid lines show the boundaries of the predicted contexts).

6.2 Parameterisation

The experiments with local policies in this chapter all use the same parameterisation consisting of a factored network of Gaussian units. This parameterisation is motivated by several considerations. The choice of basis function is constrained by the requirements of unbounded support and the need for an analytic gradient. Gaussians are a natural choice for this basis function because of their unboundedness and the simple gradient calculation. The use of a factored policy allows the activation and gradient of each basis function to be considered separately.

6.2.1 Gaussian Units

A multidimensional Gaussian function is defined by a covariance matrix, which determines its size and orientation, and a mean vector, which determines the centre of the region of response. In general, the covariance matrix can include non-zero off-diagonal terms. While allowing for such oriented Gaussians would yield a more general basis function, restricting the Gaussian to have a diagonal covariance matrix simplifies the gradient calculation by allowing the response to be factored into each dimension.

The probability that an axis-aligned Gaussian unit in m-dimensional space fires (where all variables are conditional on the time t and all parameters on the unit index i) is

$$p(a = 1|\mathbf{x}, \theta) \propto \prod_{j=1}^{m} \phi(x_j; \theta_j),$$

where $\phi(x_j; \theta_j)$ is the response for dimension j. The log response given by

$$\begin{aligned} \log p(a = 1|\mathbf{x}, \theta) &= \log(K \prod_{j=1}^{m} \phi(x_j; \theta_j)) \\ &= \alpha + \sum_{j=1}^{m} \log \phi(x_j; \theta_j) \end{aligned}$$

includes a scaling parameter α which controls the probability mass of the policy.

This axis-aligned Gaussian is parameterised by a vector of variances σ^2 and the centre μ. The response for each dimension is given by

$$\phi(x_j; \theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right).$$

Re-parameterising by $\lambda_j = \log \sigma_j$ removes the possibility of the variance becoming negative. The log response is thus

$$\log \phi(x_j; \theta_j) = -\frac{1}{2} \log 2\pi - \lambda_j - \frac{(x_j - \mu_j)^2}{2e^{2\lambda_j}}.$$

The size of the parameter vector θ for a policy with N units is $N(2m + 1)$, where m is the dimensionality of the input space.

6.2.2 Gradient

The gradient $\nabla \log a_1$ of the log probability for a positive action (i.e. $A = 1$) is given by:

$$\begin{aligned}
\frac{\partial \log a_1^{(i)}}{\partial \mu_j^{(i)}} &= \frac{x_j - \mu_j^{(i)}}{e^{2\lambda_j^{(i)}}} \\
\frac{\partial \log a_1^{(i)}}{\partial \lambda_j^{(i)}} &= \left(\frac{x_j - \mu_j^{(i)}}{e^{\lambda_j^{(i)}}} \right)^2 - 1 \\
\frac{\partial \log a_1^{(i)}}{\partial \alpha^{(i)}} &= 1.
\end{aligned}$$

This gradient can be used to calculate the gradient for non-firing actions (i.e. actions where $A = 0$) using a simple identity for binary actions[2]:

$$\nabla \log a_0 = -\frac{a_1}{a_0} \nabla \log a_1.$$

The effect of this gradient with policy-gradient learning is intuitive. The gradient of the unit's centre is a vector pointing directly toward (or, for a non-firing action, directly away from) the input point. Thus, if a unit fires and is followed by a positive reward then the unit will move toward the point where it fired with a step size proportional to how unlikely the firing action was.

The gradient of the scaling parameter simply makes actions more or less likely depending on whether the action is followed by positive or negative reward. A danger of using this parameterisation is that, if this scaling parameter becomes too large, then the probability of a firing action will saturate at 1 for a region close to the centre. Thus, the probability of a non-firing action will be zero, thus violating the assumption of unbounded support for all actions.

This can be avoided by either fixing the parameter to a nominal value or setting it so that the maximum response (at the centre of the unit) is a certain value, p_{max}. This value varies depending on the variance of the Gaussian:

$$\alpha(p_{max}; \lambda) = p_{max}(2\pi)^{m/2} \exp\left(\sum_{j=1}^{m} \lambda_j \right). \tag{6.1}$$

The gradient of the variance is positive when $|x_j - \mu_j| > \sigma_j$. The effect of this is that the activation region will expand on positive reward if the point is further than one standard deviation from the center and contract otherwise.

6.2.3 Stochastic Real-Valued Units

There is a history of using Gaussians in reinforcement learning problems [Wil92]. Their typical use is to add noise to the output of a deterministic neural network

[2]See Appendix A.

as a form of undirected exploration in order to learn continuous actions in immediate reward problems. These so-called *Stochastic Real-Valued* (SRV) units [Gul90] use the variance of the Gaussian to control exploration. As learning progresses, the units converge to a zero variance deterministic output (usually aided by some step-size scheme involving a function of the variance forcing the learning to slow as it reaches the convergence point).

While the gradient calculation is similar, the difference between these approaches and the method presented here is between using the Gaussian as a probability distribution for exploration of continuous actions and using it as a basis function to generalise over input for binary actions.

6.3 Experiments

This section details the results of using the local policy parameterisation on low-dimensional control tasks. The domain for the experiments consists of a two-dimensional particle simulation where the objective is to maximise a deterministic function of the particle's trajectory.

The state space is four-dimensional, consisting of the particle's position and velocity (the particle has no internal dynamics). The particle's state is numerically integrated with a step-size of 0.1. The particle is bounded by a square with sides of length 1. Collisions with this boundary have a coefficient of restitution of 0.5 and contact forces are modeled using a simple coulomb friction model. The particle has a mass of 1 and is subject to a constant downwards force of strength 0.25 and viscous drag of strength equal to the velocity. For more details of the simulation see Appendix A.

The initial position of the particle is distributed uniformly over an area along the top edge of the environment and the initial velocity is set to zero. As the particle hits the bottom boundary, the episode ends and the state is resampled from the initial distribution.

The agent is able to selectively apply any or all of fixed number N of horizontal forces of magnitude $F = 0.4$ in either direction, where each force is controlled by an individual Gaussian unit i which is parameterised as above by the 5-tuple $\theta^{(i)} = \{\alpha^{(i)}, \mu_x^{(i)}, \mu_y^{(i)}, \lambda_x^{(i)}, \lambda_y^{(i)}\}$. The forces are applied for two simulation time steps at a time, with action selection occurring every second time step.

The architecture is similar to that used in previous chapters: a single output network consisting of a plastic stochastic layer and a fixed deterministic layer with unknown weights (see Figure 6.3). While there are a number of possible combinations of forces, much as in the recruitment task, there are fewer actual distinct actions. While there are 2^{2N} possible firing combinations, because each force is homogenous and opposing forces cancel, there are effectively only $2N + 1$ distinct actions (representing the range of forces in $\{-FN, -F(N-1), \ldots, FN\}$).

The initial parameters were chosen so that 8 units (4 in each direction) were laid out in a grid-centred packing across the state space with a variance of 0.16^2 in both dimensions. The response at the centre of the region was fixed to 1 by

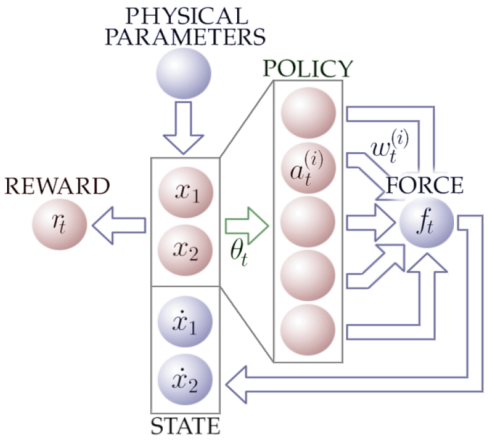

Figure 6.3: **Particle Simulation Policy.** The state of the particle (its observable position $\mathbf{x}$ and hidden velocity $\dot{\mathbf{x}}$) evolves according to the physical parameters and the force $\mathbf{f}_t$ applied. The policy stochastically generates a force output depending on the position of the particle and the current policy parameters, θ_t. The reward r_t is a deterministic function of the particle's position.

updating the scaling parameter $\alpha^{(i)}$ at each step using equation 6.1. Figure 6.4 shows sample trajectories from this initial policy as well as the sampled stationary distribution. The stationary distribution was sampled by accumulating the occupancy of the particle over 15,000 episodes in a 100 square grid laid over the input space. Each transition was interpolated across the grid and the cells which intersected the transition were convolved by a Gaussian with variance of 1 grid cell.

Experiments were run on several tasks with differing reward functions. The reward signal for each task consisted of the sum of a number of axis-aligned Gaussians. The tasks differed in the number and placement of the reward regions. No noise was added to the reward signal. The use of Gaussians for the reward results in a smooth reward signal, improving the accuracy of the gradient estimate.

The motivation for choosing this domain was to test the viability of the local policy parameterisation in tasks which had some of the characteristics of real-world problems. In particular, the simulation produces relatively complex dynamics, as evidenced by the trajectories sampled from the initial policy. Also, the task is partially observable as the policy has no access to the particle's velocity. This is particularly important as the effect of an applied force varies significantly depending on the velocity. Although the reward, transition, observation models are deterministic, complicating the task by adding noise should have little effect other than to slow learning.

An advantage of the dynamics of this task is that the episode length is independent of the actions. The longer the trajectory, the more zero reward is received. Thus, assuming the total reward of the trajectory is greater than zero,

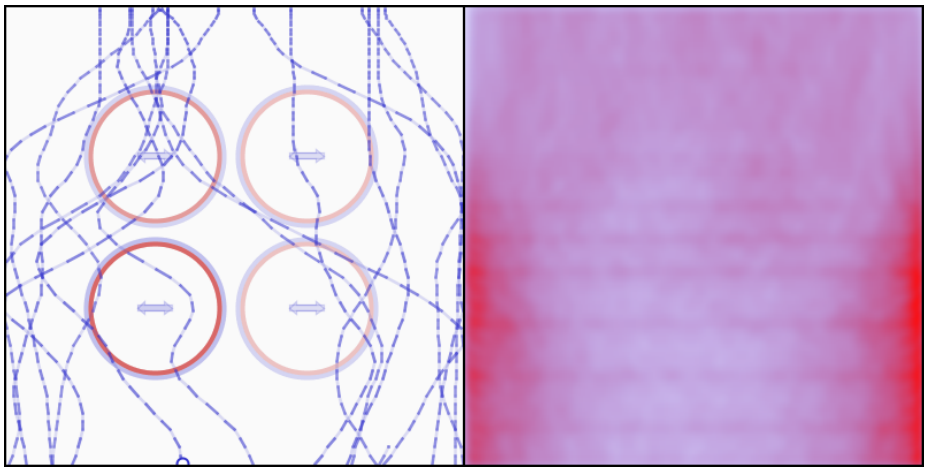

Figure 6.4: **Initial Tiled Policy.** The initial tiled local policy and sample trajectories *(left)* are shown along with its stationary distribution *(right)*. The dashes in the trajectory indicate the simulation step size and are proportional to the particle's velocity. The ellipses are positioned at the centre of each Gaussian unit with the length of the axes illustrating one standard deviation.

a shorter trajectory would result in greater offset reward. If the control were able to influence the length of the trajectory, then a policy which traded off the length of the trajectory for the amount of non-zero reward received would be optimal, resulting in unintuitive behaviour.

6.3.1 Results and Observations

Figure 6.5 shows the converged policies and the stationary distributions for two tasks. The learning rate was set to 2×10^{-4} for the centre parameters and 8×10^{-4} for the variance parameters. Figure 6.6 shows the probability of at least one unit firing for the policy learnt for the second task in Figure 6.5.

These is evidence that the random noise introduced by the online updating causes the parameter trajectory to jump from local optima. This occurs because a slight change in a single unit can disrupt the dynamics and performance, forcing other units to move. Also, as could be expected, there are many local minima for these tasks evidenced by the fact that the policies learnt for a symmetrical task are highly asymmetrical.

Because the sampling distribution changes during learning, some units can end up taking no part in the policy. These *junk* units can be seen in the first task in Figure 6.5: six of the units are effectively removed from the policy, three pushed outside the bounds of the environment. The other three were initially beneficial to the policy, serving to protect the particle from the negative region in the bottom left, but once a unit moved to the top left, these units are redundant. These junk units could be pruned to improve the computational

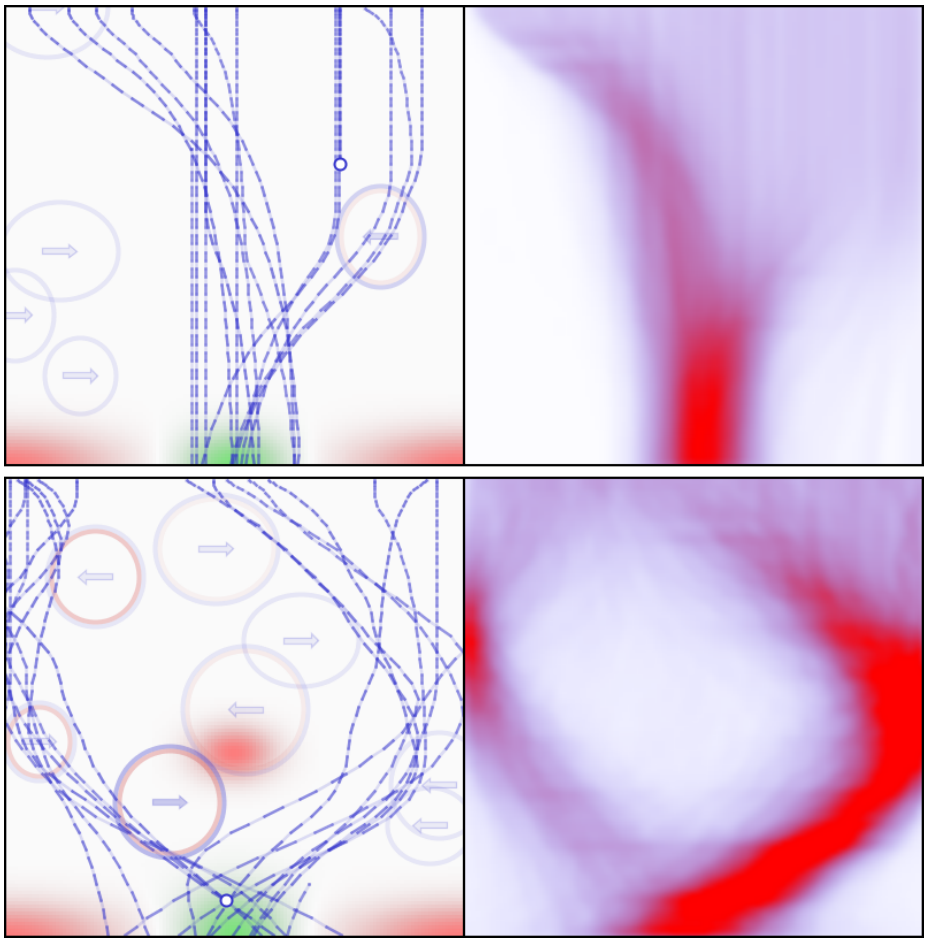

Figure 6.5: **Local Policy Learning.** These diagrams show the policies *(left)* and their stationary distributions *(right)* after $2,000$ episodes of learning for two different tasks. Non-zero reward is indicated by the green (positive) and red (negative) regions.

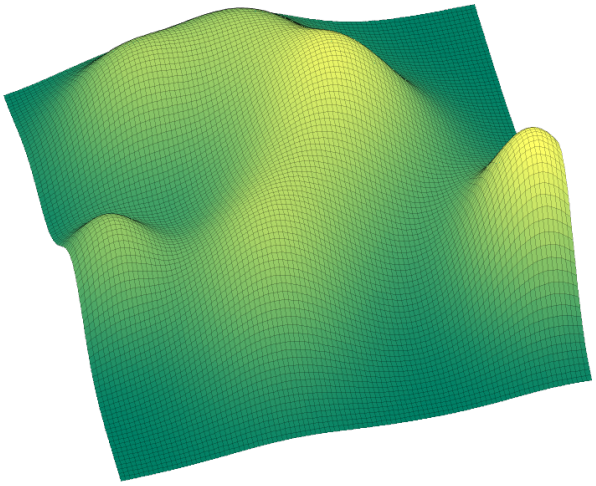

Figure 6.6: **Local Policy.** The policy for the bottom task in Figure 6.5. The height of the surface is the probability of at least one unit in the network firing.

efficiency[3].

From a distal perspective, the learnt policies exhibit interesting cooperative behaviour. For example, the policy in Figure [**?**] seems to utilise rebounding off of the boundary to both avoid negative reward regions and apparently to reduce the noise in the position for other units later in the trajectory. From the unit's point of view, the boundary is equivalent to a deterministic unit whose force is proportional to the normal (horizontal) component of the particle's velocity.

6.4 Constructive Algorithm

One aspect in which these experiments fail to capture the complexity of real-world problems is the dimensionality of the input space. Extending this parameterisation to high-dimensional problems is not straightforward, however. The main hurdle is specific to the use of local approximators. Local approximators suffer from the curse of dimensionality, i.e. the number of local units needed to tile a space is exponential in the number of dimensions. Thus, naïvely tiling the input space with local units (as was done in the initial experiments) is not possible in high dimensions.

While the observation spaces for real-world problems may be high-dimensional, it is possible that the dynamics may be locally low-dimensional. That is, the dynamics may actually lie on low-dimensional manifolds in the input space, and thus locally low-dimensional policy boundaries may be sufficient to provide optimal control. There is evidence to suggest that this is the case for motor

[3]However, this would have little impact on the time complexity if a space subdivision scheme was used to calculate the relevant units (see section 6.5.4).

control tasks. Firstly, the amount of data needed to learn mappings in globally high-dimensional space is orders of magnitude greater than is possible to experience in a lifetime. Secondly, empirical evidence suggests that real-world motor control problems are locally low-dimensional. For example, a local approximator[4] used to learn the inverse dynamics for a robotic motor control problem with 21 input dimensions found the distribution effectively only had locally 6 dimensions [AMS97]. If this is the case, then the number of units needed for optimal motor control will be sub-exponential.

However, this does not address the issue of choosing a suitable number of units in the architecture or the initial placement of these units. In general, deciding *a priori* a suitable level of complexity for a parametric representation is difficult, especially in online learning. Online learning of systems with unknown complexity is still very much an open issue in learning theory. The typical approach is to revert to trial-and-error procedures. Techniques for model selection in supervised learning (such as cross-validation and weight decay) exist as a form of complexity control to penalise overfitting and improve generalisation performance, and are inapplicable to learning control.

An alternative approach is to avoid guessing a suitable architecture *a priori* and use some form of *generative* architecture where the number of units (and thus the number of parameters) is dynamic. Two options for generating these architectures include pruning units from a complex network, or growing units from a smaller network. The latter *constructive algorithms* start from minimal networks and add new units during the course of learning to incrementally complexify the parameterisation.

6.4.1 Advantages of Constructive Algorithms

There are several advantages to using some form of constructive algorithm in online control learning over a static architecture. The main motivation is avoiding the need for an exponential number of units to tile the input space. However, avoiding this tiling also has the effect of improving convergence times by reducing the variance in the gradient estimate. This is because, in an unbiased tiling scheme, the overlapping activation regions result in noisy gradient estimates.

The difficulty of learning the parameters for each unit is complicated by the fact that every other unit is changing at the same time. As the units do not communicate with each other (only indirectly through the behaviour of the whole system), the performance function for each unit is constantly changing. The result is that, instead of each unit moving quickly and directly to assume some useful role, the units perform a complex dance that takes a long time to settle down. By using a constructive algorithm where the adaptation of the existing units is suspended when a new unit is added, the new unit sees a fixed problem. Thus, constructive algorithms can separate the effects of individual unit resulting in faster learning.

[4]The *Locally Weighted Learning* algorithm used is a local approximator using piecewise linear models weighted by fixed Gaussian kernels.

Other advantages of constructive algorithms include finding adequately sized networks resulting in more computationally economical approximators which have the potential for matching the intrinsic complexity of the task. Also, the use of fewer units is more transparent in extracting knowledge. Finally, there is physiological evidence that learning in the adult brain does involve growing new neurons [Gag02].

6.4.2 Related Work

Several methods exist which adapt network topology for classification tasks, e.g. the cascade-correlation [FL90] and upstart [Fre90] algorithms. However, methods for control tasks are more limited. Most such approaches use some form of genetic algorithm and have demonstrated good performance in standard control tasks such as pole-balancing [SM02]. More recently, the cascade-correlation algorithm has been applied to reinforcement learning tasks to approximate the value function [RP03].

Another approach to handle the curse of dimensionality is used by the CMAC architecture which uses tricks such as hashing and ignoring some dimensions of some of the tilings.

6.4.3 Implementation

Implementing a constructive algorithm is a straightforward procedure. The algorithm alternates between two phases: learning and adding. The learning phase consists of learning all of the parameters of all of the units in the network. The adding phase consists of adding a new unit to the network and slowing the learning of the existing units while learning the parameters of the new unit.

These phases introduce many new free parameters into the learning algorithm. The various issues raised include choosing the initial policy parameters, the learning step-sizes, and the adding and stopping criteria.

Initial Parameters

To avoid biasing the search, each additional unit needs to begin with a global search in order to find a place to fit. A global search results from an unbiased response over the entire input space, i.e. maximising the entropy of the unit. In sigmoidal policies, an unbiased response (where the probability of each unit firing is 0.5) results from the use of zero weight parameters. However, forcing a local approximator to have a global response is problematic. The entropy of a Gaussian distribution is proportional to its variance, so a Gaussian with a large entropy will have a large variance. Thus, if the response is normalised, then the result is the response will approach zero as the entropy approaches infinity. Scaling the response is necessary in order for a unit to have an effect on the dynamics.

Conversely, this scaling is also useful for dampening the probability of acting. In the domain of force-generating binary controllers where one of the actions has considerably more impact on the dynamics, the firing action may

overwhelm the dynamics if the probability of firing is too large. The scaling parameter can be manually adjusted to counter this effect.

Variance Reduction
In order to force each unit to fit a local region, during the adding phase the variance of the new unit is decreased by a constant factor each time step. The choice of this factor involves a trade-off between ensuring each added unit has the time to be able to travel to the bounds of the state space and the speed of learning.

Another issue is when to stop reducing the variance and switch to the learning phase. The simplest choice is to cease shrinking the unit once the variance reaches a certain cutoff.

Learning Rates
In order to improve the gradient estimate for each new unit, the learning rates of the existing units are decreased during the learning phase. This addresses the so-called moving target problem and allows the learning rate for the new unit to be increased. Thus, the algorithm now uses three different learning rates.

Adding Criterion
The natural precondition for another unit to be added to the network is to wait until learning has converged to a local maxima. The difficulty with detecting this is that the bias in the gradient estimate from the use of stochastic gradient ascent combined with the bias from the decaying eligibility means that the parameters will not converge to a point, but rather to a region. Thus, detecting convergence is not simply a process of measuring the velocity of the parameter trajectory.

A simple alternative was used here: the learning time was fixed to a constant number of time steps. A difficulty with this scheme is that the optimal setting is very problem dependent and grows with the size of the network due to the stochasticity of the units.

Stopping Criterion
Another related issue is when to stop adding units. The naïve criterion would be to stop once the improvement from adding the previous unit is negligible (or less than zero). This is problematic for two reasons. Firstly, the noise in the learning means that each unit may converge to a different local optimum. Secondly, the task may require the use of many units before any reward can be sampled.

In practice, the increasing size of the network is not a problem, unlike supervised learning where the generalisation performance is degraded by the use of overly complex networks.

6.4.4 Experiments

The constructive local policy learning algorithm was tested on a range of tasks in the particle domain. The evolution of the policy and the average reward for

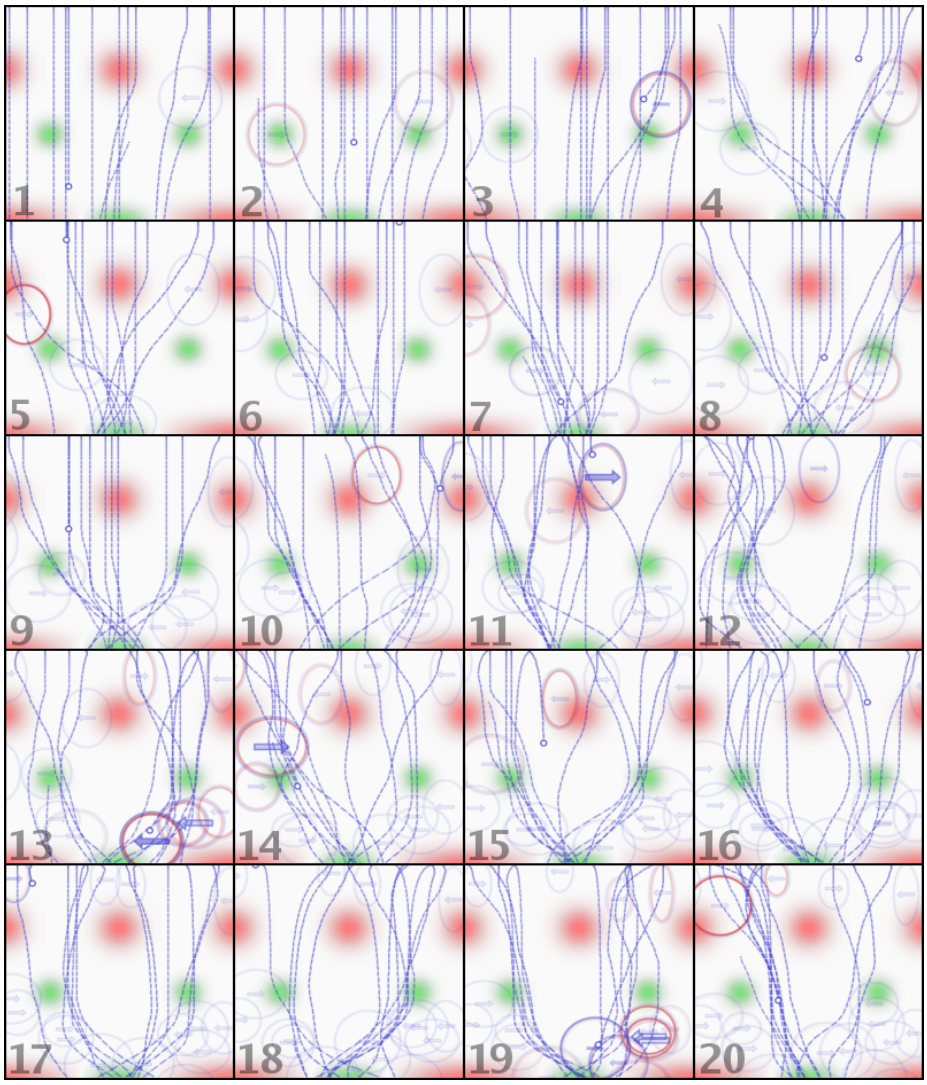

Figure 6.7: **Constructive Local Policy Learning.** These diagrams show the evolution of the local policy as new units are added over $8,000$ episodes. Each tile shows sample trajectories policy after a new unit has been placed.

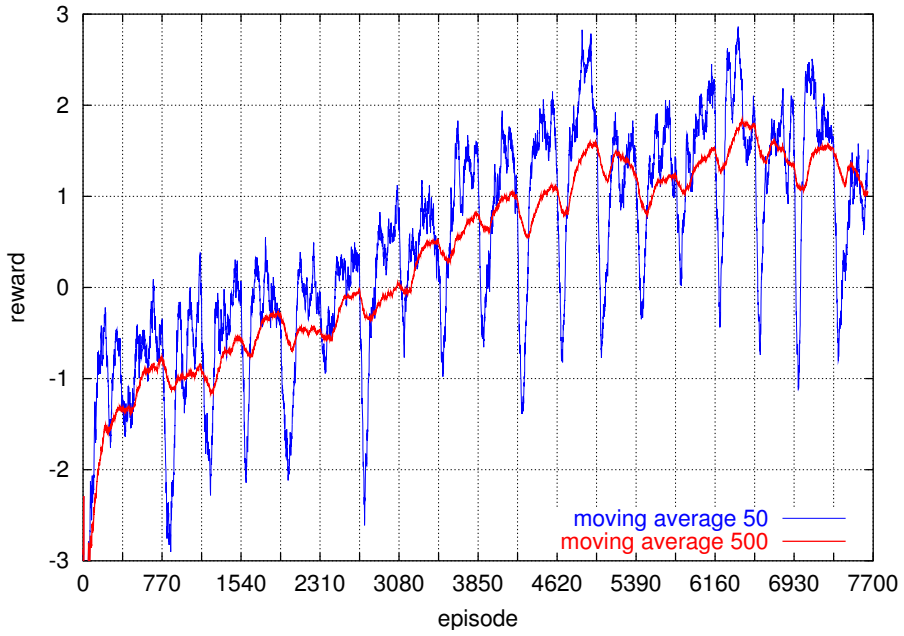

Figure 6.8: **Constructive Local Policy Learning.** This graph shows the increase in average reward for the task in Figure 6.7. The lines show the moving average of the reward over the previous 50 and 500 episodes. The vertical grid lines indicate when new units were added, and show a large drop in reward as each new unit disrupts the dynamics. This graph suggests that the performance peaked after approximately 13 units had been added.

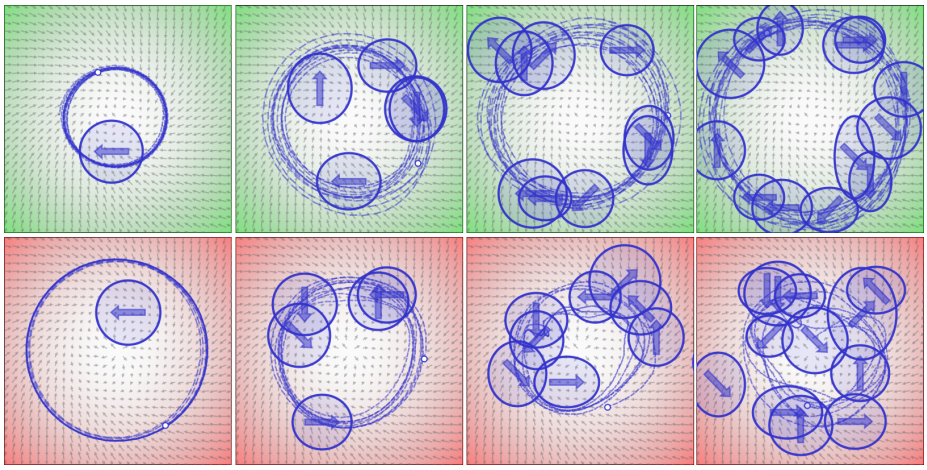

Figure 6.9: **Non-Episodic Tasks.** These series of images show the constructive algorithm learning in two non-episodic tasks with similar dynamics (created by a clockwise force field) but different reward functions. They show learning to maximise and minimise (*top* and *bottom*, respectively) the radius of the particle trajectory by positioning forces in 8 directions.

a moderately difficult problem are shown in Figures 6.7 and 6.8. The scaling parameter was fixed so that the response at the centre was 0.5. The initial centre was set to $(0.5, 0.5)$ and the variance for each unit was set to 1 in each dimension. The eligibility decay rate was 0.95. The base learning rate used during the learning phase was 1×10^{-4} for the centre parameters and 5×10^{-4} for the variance. During the placing of each new unit, the variance decayed at 0.99975 each time step and the learning rates were 8 times faster for the new unit and half as fast for the existing units. A new unit was added every $10,000$ steps. The variance reduction cutoff was set at $\sigma^2 = 0.15^2$.

The constructive algorithm was also tested on non-episodic tasks. A force field was setup which both pushed the particle in a clockwise orbit around a centre point and forced it toward an equilibrium radius[5]. The topology of these dynamics is effectively similar to the use of a gravity force with a vertical wrap-around. Instead of using just two horizontal forces, 8 different forces spaced at $45°$ intervals were consecutively added. The eligibility decay rate was set to 0.98. The reward function used was inversely related to the distance from the centre. Figure 6.9 shows the results with two runs of the algorithm with a reward function which penalised closeness to the centre and one which rewarded it.

[5]For details of the force field used see Appendix A.

6.5 Discussion

This chapter has introduced a modular policy and associated divide-and-conquer strategy for learning control which decomposes a complex problem into simpler subproblems. As opposed to paired predictor-controller networks, the resulting algorithm produces an action-centred representation where the input space is partitioned based on the needs of the control rather than on the predictability of the dynamics.

This section discusses the unintuitive aspects of this parameterisation, its limitations (including the deterministic response and the greediness of the constructive algorithm), and possible extensions.

6.5.1 Unintuitive Policies

The controllers learnt with this parameterisation rely on unintuitive consequences of the parameterisation which derive from the non-explicit encoding of the control. Specifically, the total force exerted over a period of time depends on several aspects of the policy.

Firstly, the use of factored policies means that the force output of each unit additively combines to produce the total force.

Secondly, if the scaling factor is fixed so that the probability mass is constant, then the region of activation becomes tied to the level of exploration. Thus, increasing the generalisation over input space by increasing the region of activation necessarily increases the exploratory behaviour of the unit. The consequence is that the area of response is inversely related to the entropy of the policy.

Thirdly, the stochasticity in the policy is not only used for exploration but also determines the control. If the agent moves through observation space in a continuous fashion, then the probability of executing action depends on the observation sampling period and the velocity of the observations. For short action periods and low velocities, the probability of a low-probability unit activating effectively approaches 1 even for low probability actions. Thus, the stochasticity in the policy is representing both the exploration region and the degree of response. This is due to using a discrete-time algorithm for continuous systems.

6.5.2 Deterministic Response

While sigmoidal networks are able to converge to deterministic functions, and tend to do so in learning as the magnitude of the weights grows to infinity and units approach a hard threshold, Gaussian basis functions are unable to converge to a deterministic response[6] so every unit will have some stochastic response.

Retaining this stochasticity in the policy may have some practical advantages. Firstly, stochastic units are more robust to changes in the policy, such

[6]Unless they converge to a trivial zero response with either zero or infinite variance.

as when a new unit is added. This is because a change in the policy architecture alters the performance surface for the existing units. Thus, units which retain their stochasticity are capable of exploring to find their new local optimum. Secondly, stochastic units are capable of reacting to a change in the environment's dynamics. Finally, as any learnt policy will have to cope with noise in the policy, the learnt policies will be more robust (i.e. cope with small deviations in the dynamics) than deterministic policies.

The problem with this enforced stochasticity is that the effective number of degrees of freedom of control is proportional to the number of exploring units. A unit which converges to a deterministic response effectively removes a degree of freedom from the control. Because units are not able to do this with this parameterisation, the noise in the gradient estimate will be directly related to the number of (non-junk) units due to the credit assignment problem of the simultaneous exploration of these units. This results in an increased convergence time.

A potential issue with allowing convergence to deterministic policies is the lack of exploration by a deterministic unit. As the performance surface for each unit changes due to the movement of other units, units which lack exploration will become stuck in suboptima as learning progresses. In incremental supervised learning techniques such as the upstart algorithm, another unit can come along and correct for a single error in the classification. Similarly, the use of a constructive algorithm allows units to correct for mistakes in the control and circumvents this issue.

One approach to allow for convergence to a near deterministic policy would be to use a policy which independently parameterised the size of the activation region and the degree of exploration. This could be achieved by adding stochastic noise to a deterministic response. In fact, any deterministic function defining an area can be made into a stochastic local policy by adding noise to the boundary.

An example of this would be to define a spherical activation region with stochastic boundaries, i.e.

$$f(\mathbf{x}) \propto \phi(k(\|\mathbf{x} - \mu\| - r))$$

where $\phi(x)$ is the logistic sigmoid function, r is the radius of the sphere, k is the hardness of the threshold and μ is the centre. However, while a simple expression for the policy-gradient of such a spherical unit exists, generalising to axis-aligned ellipsoids with different length axes is not possible as no analytical derivative exists (because computing the distance to an ellipsoid requires a numerical approximation). Another option without this difficulty would be to use oriented boxes with variable length axes.

The use of factored policies allows for the use of heterogenous policies, i.e. the sub-policies need not be the same, allowing the possibility of combining Gaussian units with other types of units such as global units.

6.5.3 Greediness

A potential limitation of the constructive algorithm is its greedy nature. Because each unit is optimised sequentially, the algorithm may learn policies which are less optimal for the number of units used than equivalent learning on a fixed architecture. For example, a task might require the use of several units in order to sample reward. If there was a penalty in using each unit then adding the units one at a time would remove them from the policy.

6.5.4 Extensions

The implementation of the constructive algorithm has many ad hoc free parameters such as the different learning rates, the variance reduction schedule and the adding criterion. Extensions to this algorithm should address the use of these simple heuristics by offering more principled treatment of these processes.

The constructive algorithm leaves open the question of selecting the actions of the units to add, i.e. the weights in the deterministic layer of the network. An obvious extension would be to use a SRV unit to learn the continuous parameters such as the magnitude and direction of the applied force. Recognising that each unit is functioning as a sub-controller with a static, deterministic policy, a more ambitious extension would be to extend the learning to these sub-policies, thus learning both to divide the state space into local regions and adapting the control in each of those regions. The recursive extension of this idea would yield hierarchical controllers where the output of each sub-controller is gated by its parents activation.

As local units only have an effect if placed near the dynamics, an extension could be to utilise an estimate of the distribution model to determine the placement of new units.

In addition, further work needs to be done in applying these policies to realistic motor learning problems, such as those in previous chapters. The direct analogue of these tasks would have the position of the particle corresponding to the configuration of the skeleton, and each binary unit controlling the activation of a single muscle. Another possible application is navigation. Flow fields are often used for emergent navigation [DJ00]. This local learning algorithm learn could be used to learn the flow fields for such tasks where negative reward is positioned at obstacles, and positive reward is given for goal states.

From a practical view, the use of local policies allows for the possibility of reducing the algorithmic complexity of the policy-gradient learning algorithm to below $O(n)$, where n is the number of units. This is because the magnitude of the eligibility for parameters of low-probability, unactivated units will be negligible. Utilising an efficient space subdivision data structure which mapped points in input space to units which had a probability of firing greater than some small constant would avoid the gradient calculation for these low probability units. Thus, the time complexity of the learning algorithm would depend only on the denseness of the units.

Chapter 7

Conclusion

7.1 Summary

This thesis has investigated the application of policy-gradient learning to tasks with characteristics of real-world control problems. In particular, simulated motor learning tasks were examined because they embody the complexities of such problems.

In supervised learning tasks, testing on real-world datasets such as handwriting recognition is relatively straightforward. However, designing tasks for learning optimal control is difficult because the connection to the real world is usually less than explicit. Motor learning tasks encompass all of the difficult aspects of optimal control in an intuitive way.

This thesis has focussed solely on the use of reinforcement learning for motor learning. However, the behaviour of motor systems cannot be explained through reinforcement learning alone. Other aspects of the motor system require the consideration of other machine learning techniques such as unsupervised learning for motor primitives, supervised learning for imitation learning and learning predictive models, and filtering for tracking features in the absence of sensory information.

The recurring theme in this thesis was the principle of divide-and-conquer. In particular, the distal error problem was tackled by the use of factoring, or decomposing joint distributions into the product of smaller local distributions. Specifically, the control was factored to estimate the effect of each parameter of each dimension of each subpolicy for structural credit assignment, while the observed dynamics were factored into each transition of each trajectory of each episode for temporal credit assignment. Another use of the divide-and-conquer strategy was the introduction of a constructive algorithm to decompose the problem of learning the parameters of several units at once into the simpler individual subproblems of learning each unit separately. Such pervasive application of divide-and-conquer techniques contrasts with typical global optimisation approaches to complex control tasks where both control and behaviour are considered indivisible.

From a computational perspective, policy-gradient learning algorithms are

very simple. Whereas model-based and value-function approaches become computationally intractable in high-dimensional problems, policy-gradient learning scales linearly. However, policy-gradient learning is expensive from a sample complexity perspective. This is due to gradient ascent's limited use of data and due to the long convergence times caused by both the use of undirected exploration and the high variance of the gradient estimate. In conducting simulated motor learning experiments, the large number of samples needed is exacerbated by the time taken to simulate realistic dynamical systems. Thus, the problems tackled by this thesis are only beginning to be approachable by available computational power.

It is difficult to compare policy-gradient learning to alternatives as there are no other general methods for learning stochastic optimal control in continuous spaces. While more suitable than other alternatives, there is still a lot of work to be done before policy-gradient techniques can be scaled up for general application to complex tasks without assuming prior information.

7.2 Contributions

The main contributions of this thesis are:

- The introduction and demonstration of a plausible process for the adaptation of motoneuron thresholds for motor unit recruitment.
- The introduction and demonstration of the use of a constructive algorithm for learning local factored policies for general optimal control tasks.
- The demonstration of the use of online reinforcement learning in a physically-realistic simulated agent.
- The development of an extensible simulation framework for the research of control in realistic dynamical systems.

7.3 Future Work

Several possible extensions to the work presented in this thesis and prospective research directions have been considered at the end of each chapter.

One consideration that has not been explored is the optimality of the underlying gradient ascent procedure. Gradient ascent methods effectively estimate a local linear model of how the performance changes with respect to the parameters. In general, however, optimising functions using point estimates of the gradient is wasteful of samples as each sample is used insofar as to estimate the local gradient and then discarded.

The use of parameterisation in optimising control allows for the application of statistical learning theory. While Bayesian approaches in recent years have dominated machine learning, the use of Bayesian techniques for learning optimal control has largely been ignored, possibly due to reinforcement learning's foundations in dynamic programming and discrete systems. Various methods

which introduce Bayesian techniques to control learning include inferring a distribution over possible environment models [DFR98], using unsupervised learning to learn motor primitives [TG03], or using non-parametric methods to model the value-function [EMM03, RK03]. However, there appears to be no use of Bayesian techniques for the direct optimisation of control.

Policy-gradient methods are suboptimal from a Bayesian perspective. The simple stochastic gradient ascent algorithm used throughout this thesis is non-Bayesian for several reasons: it does not use all of the data (the evolution of the parameters is the only trace of the previously seen data), the order of the data matters and it does not take into account any uncertainty. A more principled use of the samples than classical Monte Carlo would be to explicitly infer a posterior distribution of the integrand as in [RG03].

In addition, in a Bayesian formulation, the gradient samples should be used to estimate the objective surface and derive a search algorithm from the uncertainty underlying this model. For example, a non-parametric approach could use a Gaussian process to approximate the objective surface. As the derivative of a Gaussian process is another Gaussian process, estimates of the performance gradient could be used as samples. The error bars on the surface would allow the use of conjugate-gradient ascent or Markov chain Monte Carlo techniques to find peaks in the expected improvement from the next sample. An interesting observation is that gradient ascent would be expected to fall out of this process[1].

Such an approach may prove to be a good match with policy-gradient learning. Whereas policy-gradient learning needs many samples to estimate the gradient, Gaussian processes make good use of limited samples but its complexity is high.

The major problem with this approach is the computational complexity of the non-parametric regression ($O(n^2)$ with the number of samples). Practical approaches would need to use such methods as exploiting the sparseness of the covariance matrix or forgetting samples. In addition, applying such a techniques to a dynamic architecture such as the one presented in this thesis is not straightforward, as performing regression is difficult in the presence of changing dimensionality of parameter space.

While such a specific approach is unproven, in general the use of Bayesian techniques for model-free learning is an exciting direction in optimal control.

[1] Marcus Frean, personal communication, May 2003.

Appendix A

Miscellaneous

A.1 Policy-Gradient for Stochastic Binary Units

The relationship between $\nabla \log a_0$ and $\nabla \log a_1$ for a binary unit is given by

$$\begin{aligned} \nabla \log a_0 &= \frac{\nabla(1-a_1)}{1-a_1} \\ &= \frac{-\nabla a_1}{1-a_1} \\ &= -\frac{a_1 \nabla \log a_1}{1-a_1} \\ &= -\frac{a_1}{a_0} \nabla \log a_1. \end{aligned} \tag{A.1}$$

The gradient for a sigmoidal activation function with potential ϕ is calculated using the chain rule. The gradient for a firing action is given by

$$\begin{aligned} \frac{\partial a_1}{\partial \theta} &= \frac{\partial a_1}{\partial \phi} \frac{\partial \phi}{\partial \theta} \\ &= a_1(1-a_1) \frac{\partial \phi}{\partial \theta} \\ \therefore \quad \frac{\partial \log a_1}{\partial \theta} &= (1-a_1) \frac{\partial \phi}{\partial \theta}. \end{aligned}$$

Using A.1, the gradient for a non-firing action is:

$$\begin{aligned} \frac{\partial a_0}{\partial \theta} &= -\frac{a_1}{a_0}(1-a_1) \frac{\partial \phi}{\partial \theta} \\ &= -a_1 \frac{\partial \phi}{\partial \theta}. \end{aligned}$$

More simply, for action A the gradient is,

$$\frac{\partial \log a_A}{\partial \theta} = (A - a_1) \frac{\partial \phi}{\partial \theta}.$$

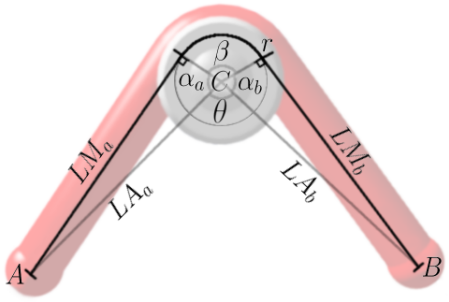

Figure A.1: **Muscle Wrapping.** The extensor muscle defined by points A and B wraps around a joint capsule centred at C with radius r. All distances and angles shown in the diagram (except for the wrapping angle β) remain constant throughout the range of joint rotation.

A.2 Muscle Geometry

The derivation of the muscle path is simplified by assuming the muscle lies in the 2D coordinate system of a plane. This is achieved by projecting all points in 3D space onto the plane defined by the centre of the joint socket C with normal in the direction of the axis of rotation.

The muscle is afixed to the skeleton at the origin and insertion, points A and B. The length of the lines between these points and the point of rotation, LA_a and LA_b, and the angles between each of these lines and the lines tangent to the socket capsule, α_a and α_b, are independent of the angle of the joint and only depend on the points and the radius of the capsule, r:

$$\begin{aligned} LA_a &= \|C - A\| \quad & \alpha_a &= \cos^{-1}\left(\tfrac{r}{LA_a}\right) \quad & LM_a &= LA_a \sin\alpha_a \\ LA_b &= \|C - B\| \quad & \alpha_b &= \cos^{-1}\left(\tfrac{r}{LA_b}\right) \quad & LM_b &= LA_b \sin\alpha_b. \end{aligned}$$

The joint angle at which the muscle begins to wrap, θ^c, is also constant

$$\theta^c = 2\pi - \alpha_a - \alpha_b,$$

where the joint angle is given by $\theta = \angle ACB$ and the muscle is wrapping if $\theta < \theta^c$. The length of the muscle for a given angle depends on whether it is wrapping or not. If it is wrapping, then the length of the wrapping section is given by $LM_w = r\beta$, where the wrapping angle $\beta = \theta^c - \theta$. Thus, the total length of the muscle is given by,

$$L = \begin{cases} LM_a + LM_w + LM_b & \text{if } \theta < \theta^c \\ \|A - B\| & \text{otherwise.} \end{cases}$$

Likewise, the rate of change of the length of the muscle also depends on whether or not the muscle is wrapping. If wrapped, then the velocity of the

muscle length is given by

$$\begin{aligned}\dot{L} &= \frac{d}{dt}(LM_a + LM_w + LM_b) \\ &= \frac{d}{dt}LM_w \\ &= r\dot{\beta} \\ &= -r\dot{\theta}.\end{aligned}$$

Otherwise, the velocity is given simply by the rate of change in the distance between the origin and insertion points, i.e.

$$\begin{aligned}\dot{L} &= \frac{d}{dt}\|A - B\| \\ &= \frac{D \cdot \dot{D}}{\|D\|},\end{aligned}$$

where $D = A - B$.

A.3 Particle Dynamics

The experiments in Chapter 6 used a simple particle simulation with the following physics. The motion of the Newtonian particle is governed by the equation $\ddot{\mathbf{x}} = \mathbf{f}/m$ where $\mathbf{f}$ is the force applied to the particle and m is its mass. Creating a variable $\mathbf{v}$ representing the particle's velocity splits this second order equation into a pair of coupled first-order ODE's $\dot{\mathbf{v}} = \mathbf{f}/m$ and $\dot{\mathbf{x}} = \mathbf{v}$ so that the particle's state, $\mathbf{q}$, is a point in phase space $[\mathbf{x}, \mathbf{v}]$ with first derivative $\dot{\mathbf{q}} = [\mathbf{v}, \mathbf{f}/m]$. The particle is constrained to a plane so $\mathbf{q}$ is 4-dimensional. The state is numerically integrated using a 4th-order Runge-Kutta integrator. Thus, the state at time $t_0 + h$ is given by the equations

$$\begin{aligned}c_1 &= f(\mathbf{q}_0, t_0) \\ c_2 &= f(\mathbf{q}_0 + \frac{c_1}{2}, t_0 + \frac{h}{2}) \\ c_3 &= f(\mathbf{q}_0 + \frac{c_2}{2}, t_0 + \frac{h}{2}) \\ c_4 &= f(\mathbf{q}_0 + c_3, t_0 + h) \\ \mathbf{q}(t_0 + h) &= \mathbf{q}_0 + \frac{h}{6}(c_1 + 2c_2 + 2c_3 + c_4),\end{aligned}$$

where f is the derivative function defining a vector field.

The forces acting on the particle are accumulated at each time step. These forces include gravity, which is a constant force $\mathbf{f}_g = m\mathbf{g}$ proportional to the particle's mass m in the direction of the gravity vector $\mathbf{g}$, and viscous drag which is a force proportional to the velocity of the particle, i.e. $\mathbf{f}_d = -k_d\mathbf{v}$ where k_d is the drag constant.

The circular force field used for the non-episodic experiment consists of two component forces centred at $\mathbf{c}$, both dependent on the particle's position (see

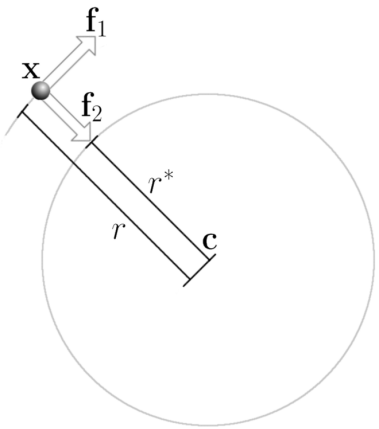

Figure A.2: **Circular Force.** The particle at $\mathbf{x}$ is pushed in a clockwise direction around the circle centred at $\mathbf{c}$ and towards radius r^* through forces $\mathbf{f}_1$ and $\mathbf{f}_2$.

Figure A.2). The force $\mathbf{f}_1$ pushes the particle in a clockwise path while the force $\mathbf{f}_2$ acts to push the particle toward the equilibrium radius, r^*. These forces are scaled by the constants k_1 and k_2 respectively to give the resultant force, $\mathbf{f}_w$:

$$\begin{aligned}
\theta &= \tan^{-1}\left(\frac{x_2 - c_2}{x_1 - c_1}\right) \\
r &= \|\mathbf{x} - \mathbf{c}\| \\
\mathbf{f}_1 &= (-r\sin\theta, r\cos\theta) \\
\mathbf{f}_2 &= (r^*\cos\theta, r^*\sin\theta) - \mathbf{x} \\
\mathbf{f}_w &= k_1\mathbf{f}_1 + k_2\mathbf{f}_2.
\end{aligned}$$

Boundary interpenetration is tested for after each time step. For a plane boundary defined by the point $\mathbf{P}$ and normal $\mathbf{N}$ interpenetration occurs if $(\mathbf{X} - \mathbf{P}) \cdot \mathbf{N} < 0$. Interpenetrations were initially handled by rolling back the system to the pre-interpenetration state and using a binary search stepping scheme to find the interpenetration time within a small constant, however it was found for the low velocities used in the experiments that it was acceptable to simply reposition the particle on the collision boundary. Collisions occurred when the magnitude of the normal component of the velocity, $\mathbf{v}_n = (\mathbf{N} \cdot \mathbf{v})\mathbf{N}$, was greater than a small constant. The collision response is an impulse which reverses the normal velocity, setting the new velocity to $\mathbf{v}' = -k_r\mathbf{v}_n + \mathbf{v}_t$, where k_r is the coefficient of restitution of the collision and $\mathbf{v}_t = \mathbf{v} - \mathbf{v}_n$ is the tangential component of the velocity. For resting contact where the particle is being pushed into the collision surface (i.e. $\mathbf{N} \cdot \mathbf{F} < 0$, where $\mathbf{F}$ is the total force applied to the particle), a contact force is generated $\mathbf{f}_c = -(\mathbf{N} \cdot \mathbf{F})\mathbf{F}$ which exactly cancels the normal force component. A simple linear friction model is used to apply a drag force, $\mathbf{f}_f = -k_f(\mathbf{N} \cdot \mathbf{F})\mathbf{v}_t$ that acts in the tangential direction.

Appendix B

Software Systems

All experiments described in this thesis were performed using software systems developed by the author. The design of these systems was intended to be as general as possible to allow for future use in research in machine learning and control. The systems consists of two applications: TIMSIM and TIMLAB.

B.1 TimSim

TIMSIM is a system for machine learning research in simulated motor control and robotic systems. Such a system must take into account several competing design requirements. Machine learning algorithms require fast simulation to accommodate large numbers of trials. As a research tool, the system must be extensible to allow for rapid prototyping and experimentation. To aid the diagnostic of complex physical systems, the system must have an interactive interface.

To achieve these goals, the core of the system is implemented in C++ while the user interface, control and learning algorithms are implemented in the Python scripting language[1]. The integration of these components is achieved through by building a Python extension library (using Boost.Python[2]) which defines a wrapper around all simulation and visualisation classes.

To reduce coupling in the design and to address the responsiveness of the application, the main functionality is divided into threads which are pre-emptively multi-threaded using the GNU portable threads library[3].

The interface consists of a 3D view of all simulated objects. The interface uses the OpenGL graphics library[4] for visualisation. Multiple camera angles are available, including walking, orbit and inspection camera functions with navigation being performed using either the mouse or keyboard. To visualise the muscles, the GLE library[5] is used to extrude pipes along the muscle path.

[1] http://www.python.org
[2] http://www.boost.org/libs/python
[3] http://www.gnu.org/software/pth
[4] http://www.opengl.org
[5] http://linas.org/gle

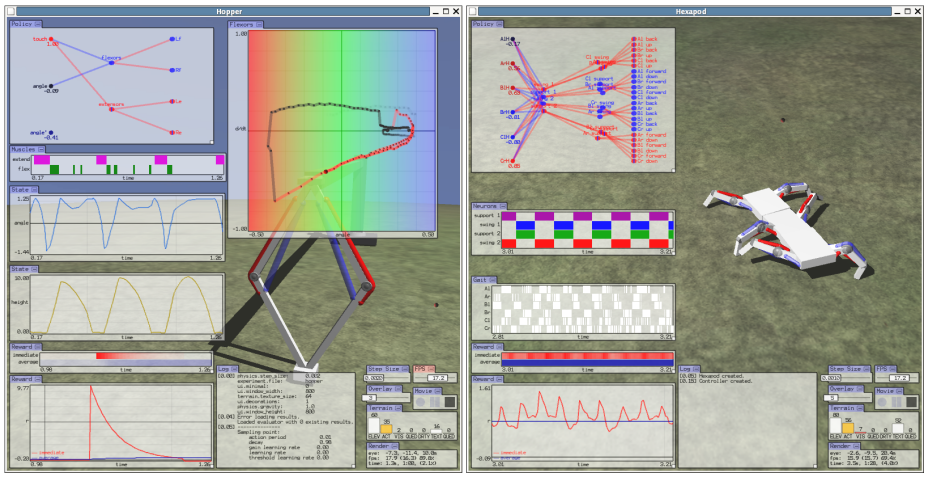

Figure B.1: TIMSIM. Screenshots of TIMSIM running the hopper and hexapod experiments.

The terrain is rendered with multitexturing for detail. Stencil-buffer shadow volumes are used for shadows between objects and the objects and the terrain. Video capture is performed by saving individual frames as raw PPM files and then encoding them into MPEG4 using transcode[6]. The overlayed user interface is constructed in Python by issuing 2D graphics commands which are stored in a C++ display list and then compiled to an OpenGL display list for reuse over several frames for faster rendering.

Terrain is modelled as an ellipsoid partitioned into a quad-tree with nodes storing heightfield elevation data. This hierarchical data structure allows an efficient level-of-detail selection using a view-space metric as in [LKR+96] as well as view-frustrum and horizon culling. Collision detection is simplified by restricting the modelled physical elements to boxes. The separating plane test is used for box collision. Box-terrain collision tests utilise the quad-tree to reduce the number of tests while edge to edge collisions are ignored. The rigid-body physics simulation, including constraint solving and state integration, is performed by the ODE engine[7].

The environment configuration such as global physical parameters (step-size, gravitational constant) and terrain parameters (ellipsoid radiuses, elevation heightfields) for experiments is parsed from configuration files.

The system (including experiment code) is around 5,000 lines of Python and over 20,000 lines of C++ code.

[6] http://www.theorie.physik.uni-goettingen.de/ostreich/transcode
[7] http://opende.sourceforge.net

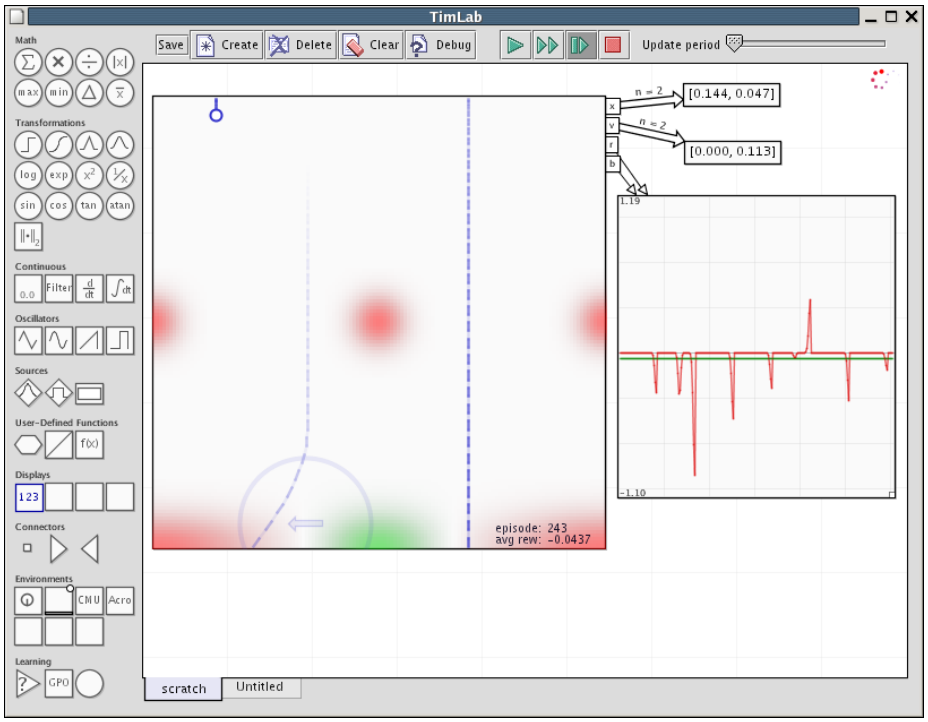

Figure B.2: TIMLAB. A screenshot from the TIMLAB application running a learning experiment.

B.2 TimLab

TIMLAB is a cross-platform application for building and simulating discrete-time continuous dynamical systems. The application is useful for building modular controllers and experimenting with dynamical system. It is implemented in Jython[8] to combine the rapid prototyping afforded by Python and the GUI capabilities of Java. It consists of around 10,000 lines of code.

[8] http://www.jython.org

Bibliography

[AB02] Douglas Aberdeen and Jonathan Baxter. Scaling Internal-State Policy-Gradient Methods for POMDPs. In *International Conference on Machine Learning*, Sydney, Australia, July 2002.

[Abe03] Douglas Aberdeen. *Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Computer Sciences Laboratory, Australian National University, March 2003.

[Alb75] J.S. Albus. A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Journal of dynamic systems, measurement and control*, 97(3), 1975.

[AMS97] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally Weighted Learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.

[Bar02] Andrew G. Barto. Reinforcement Learning in Motor Control. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 968–972. The MIT Press, Cambridge, MA, 2002.

[BB99a] P. L. Bartlett and J. Baxter. Direct Gradient-Based Reinforcement Learning: I. Gradient Estimation Algorithms. Technical report, Research School of Information Sciences and Engineering, Australian National University, July 1999.

[BB99b] P. L. Bartlett and J. Baxter. Hebbian Synaptic Modification in Spiking Neurons that Learn. Technical report, Research School of Information Sciences and Engineering, Australian National University, 1999.

[BB01] J. Baxter and P. L. Bartlett. Infinite-Horizon Policy-Gradient Estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[Bel61] Richard Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, New-Jersey, 1961.

[Ber76] Dimitri P. Bertsekas. *Dynamic Programming and Stochastic Control*. Academic Press, New York, 1976.

[BKST95] Tom Brashers-Krug, Reza Shadmehr, and Emanuel Todorov. Catastrophic Interference in Human Motor Learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 19–26. The MIT Press, 1995.

[Bro86] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. In *IEEE Journal of Robotics and Automation*, volume RA-2 (1), pages 14–23, April 1986.

[BS01] James Bagnell and Jeff Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proceedings of the International Conference on Robotics and Automation 2001*. IEEE, May 2001.

[BWB99] J. Baxter, L. Weaver, and P. L. Bartlett. Direct Gradient-Based Reinforcement Learning: II. Gradient Ascent Algorithms and Experiments. Technical report, Research School of Information Sciences and Engineering, Australian National University, July 1999.

[CKCC01] J. G. Cham, J. Karpick, J.E. Clark, and M. R. Cutkosky. Stride Period Adaptation for a Biomimetic Running Hexapod. In *10th International Symposium of Robotics Research*, Lorne, Victoria, Australia, November 2001.

[Cla97] Andy Clark. *Being There: Putting Brain, Body, and World Together Again*. The MIT Press, 1997.

[Cou02] Rmi Coulom. Feedforward Neural Networks in Reinforcement Learning Applied to High-dimensional Motor Control. In Nicol Cesa-Bianchi, Masayuki Numao, and Rdiger Reischuk, editors, *Proceedings of the 13th International Conference on Algorithmic Learning Theory*, pages 402–413. Springer, 2002.

[DFR98] Richard Dearden, Nir Friedman, and Stuart J. Russell. Bayesian Q-Learning. In *AAAI/IAAI*, pages 761–768, 1998.

[DJ00] Gregory Dudek and Michael Jenkin. *Computation Principles of Mobile Robotics*. Cambridge University Press, 2000.

[DSKK02] K. Doya, K. Samejima, K. Katagiri, and M. Kawato. Multiple Model-based Reinforcement Learning. In *Neural Computation*, volume 14, pages 1347–1369, 2002.

[EMM03] Y. Engel, S. Mannor, and R. Meir. The Gaussian Process Approach to Temporal Difference Learning. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

[Fag00] Andrew H. Fagg. A Model of Muscle Geometry for a Two Degree-Of-Freedom Planar Arm. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 2000.

[FL90] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation Learning Architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann, San Mateo, 1990.

[Fre90] Marcus Frean. The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. In *Neural Computation*, volume 2, pages 198–209, 1990.

[Fre94] Robert M. French. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 1176–1177. Morgan Kaufmann Publishers, Inc., 1994.

[FVJW03] J. R. Flanagan, P. Vetter, R. S. Johansson, and D. M. Wolpert. Prediction precedes control in motor learning. In *Current Biology 13*, pages 146–150, 2003.

[Gag02] Fred H. Gage. Neurogenesis in the adult brain. *The Journal of Neuroscience*, 22(3):612–613, February 2002.

[GKU03] G. Z. Grudic, Vijay Kumar, and L. H. Ungar. Using Policy Gradient Reinforcement Learning on Autonomous Robot Controllers. In *IEEE-RSJ International Conference on Intelligent Robots and Systems (IROS03)*, 2003.

[GP00] B. A. Garner and M. G. Pandy. The Obstacle Set Method for representing muscle paths in musculoskeletal models. In *Computer Methods in Biomechanics and Biomedical Engineering 3*, pages 1–30, 2000.

[GU00a] Gregory Z. Grudic and Lyle H. Ungar. Localizing Policy Gradient Estimates to Action Transitions. In *Proceedings of 17th International Conference on Machine Learning*, pages 343–350. Morgan Kaufmann, San Francisco, CA, 2000.

[GU00b] Gregory Z. Grudic and Lyle H. Ungar. Localizing Search in Reinforcement Learning. In *AAAI/IAAI*, pages 590–595, 2000.

[GU01] G. Z. Grudic and L. H. Ungar. Rates of Convergence of Performance Gradient Estimates Using Function Approximation and Bias in Reinforcement Learning. In *Proceedings of Neural Information Processing Systems 2001*, Vancouver, Canada, December 2001.

[Gul90] V. Gullapalli. A stochastic reinforcement algorithm for learning real-valued functions. In *Neural Networks 3*, pages 671–692, 1990.

[Hol75] John Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

[iA98] Shun ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 1998.

[JJNH91] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 1991.

[KA97] R. M. Kretchmar and C. W. Anderson. Comparison of CMACs and Radial Basis Functions for Local Function Approximators in Reinforcement Learning. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 834–837, Houston, Texas, 1997.

[Kak02] Sham Kakade. A Natural Policy Gradient. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[Kak03] Sham Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

[KF99] T. M. Kubrow and R. J. Full. The role of the mechanical system in control: a hypothesis of self-stabilization in hexapedal. In *Phil. Trans. Roy. Soc. London B. 354*, pages 849–862, 1999.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[KYK95] H. Kimura, M. Yamamura, and S. Kobayashi. Reinforcement Learning by Stochastic Hill Climbing on Discounted Reward. In *Proceedings of the 12th International Conference on Machine Learning*, pages 295–303, Yokohama, Japan, 1995.

[LCR03] Gregory Lawrence, Noah J. Cowan, and Stuart Russell. Efficient Gradient Estimation for Motor Control Learning. In *Conference on Uncertainty in Artificial Intelligence*, Acapulco, Mexico, August 2003.

[Lee84] W. Lee. Neuromotor synergies as a basis for coordinated intentional action. In *Journal of Motor Behavior*, volume 16, pages 135–170, 1984.

[Lit96] Michael L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, March 1996.

[LKR+96] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, and Nick Faust. Real-Time, Continuous Level of Detail Rendering of Height Fields. In *Proceedings of ACM SIGGRAPH 96*, pages 109–118, August 1996.

[MA95] A. W. Moore and C. G. Atkeson. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State Spaces. In *Proceedings of Machine Learning, 21(3):199-233*, 1995.

[MIGB94] F. A. Mussa-Ivaldi, F. A. Giszter, and E. Bizzi. Linear combination of primitives in vertebrate motor control. In *Proc. Nat. Acad. Sci. USA 91*, pages 7534–7538, 1994.

[NJ00] Andrew Y. Ng and Michael Jordan. PEGASUS: A Policy Search Method for Large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[PKMK00] L. Peshkin, K. Kim, N. Meuleau, and L. Kaelbling. Learning to Cooperate via Policy Search. In *Sixteenth International Conference on Uncertainty in Artificial Intelligence*, 2000.

[PTJ94] Singh S. P., Jaakkola T., and M. I. Jordan. Model-free Reinforcement Learning for non-Markovian Decision Problems. In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

[PVS03] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement Learning for Humanoid Robotics. In *Proceedings of the Third IEEE-RAS International Conference on Humanoid Robots*, Germany, September 2003.

[RB01] Michael T. Rosenstein and Andrew G. Barto. Robot Weightlifting By Direct Policy Search. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, volume 2, pages 839–844, 2001.

[RG03] C. E. Rasmussen and Z. Ghahramani. Bayesian Monte Carlo. In *Advances in Neural Information Processing Systems*, volume 15, Cambridge, MA, 2003. MIT Press.

[RK03] Carl E. Rasmussen and Malte Kuss. Gaussian Processes in Reinforcement Learning. Technical report, Germany, 2003.

[RP03] Francois Rivest and Doina Precup. Combining TD-learning with Cascade-correlation Networks. In *Proceedings of the Twentieth International Conference on Machine Learning*, Washington DC, 2003.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[SM02] Kenneth O. Stanley and Risto Miikulainen. Efficient Reinforcement Learning through Evolving Neural Network Topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, 2002. Morgan Kaufmann.

[SMSM99] R. S. 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*, pages 1057–1063. MIT Press, 1999.

[Sut96] Richard S. Sutton. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press, 1996.

[SWC+95] W. Senn, K. Wyler, H. Clamann, J. Kleinle, M. Larkum, H. Luscher, L. Muller, K. Vogt, and T. Wannier. Size principle and information theory. In *Biol. Cybernetics*, 1995.

[TBW01] Nigel Tao, Jonathan Baxter, and Lex Weaver. A Multi-Agent, Policy-Gradient approach to Network Routing. In *ICML 2001: 18th International Conference on Machine Learning*. Morgan Kaufmann Publishers, July 2001.

[Tes95] Gerald Tesauro. Temporal difference learning and TD-Gammon. In *Communications of the ACM*, volume 38(3), pages 58–67, March 1995.

[TG03] Emanuel Todorov and Zoubin Ghahramani. Unsupervised Learning of Sensory-Motor Primitives. Technical report, University of Califonia San Diego, and Gatsby Computational Neuroscience Unit, UCL, 2003.

[tH01] Stephan ten Hagen. *Continuous State Space Q-Learning for Control of Nonlinear Systems*. PhD thesis, Computer Science Institute, University of Amsterdam, The Netherlands, 2001.

[TL03] Emanuel Todorov and Weiwei Li. Locally Optimal Control of Continuous Stochastic Systems. Technical report, University of Califonia San Diego, 2003.

[Tse73] Mikhail L. Tsetlin. *Automaton Theory and Modeling of Biological Systems*. Academic Press, New York, 1973.

[Wil92] Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

[WK98] Daniel M. Wolpert and Mitsuo Kawato. Multiple Paired Forward and Inverse Models for Motor Control. In *Neural Networks 11*, 1998.

[WM97] David H. Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[WT01] Lex Weaver and Nigel Tao. The Optimal Reward Baseline for Gradient-Based Reinforcement Learning. In *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference*, pages 538–545. Morgan Kaufman Publishers, August 2001.