To appear in: *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*
David A. White and Donald A. Sofge (editors)
Van Nostrand Reinhold, Florence, Kentucky 41022 (publisher)

THE ROLE OF EXPLORATION IN LEARNING CONTROL

Sebastian B. Thrun

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213
Email: thrun@cs.cmu.edu

1 Introduction

Whenever an intelligent agent learns to control an unknown environment, two opposing objectives have to be combined. On the one hand, the environment must be sufficiently explored in order to identify a (sub-) optimal controller. For instance, a robot facing an unknown environment has to spend time moving around and acquiring knowledge. On the other hand, the environment must also be exploited during learning, i.e., experience made during learning must also be considered for action selection, if one is interested in minimizing costs of learning. For example, although a robot has to explore its environment, it should avoid collisions with obstacles once it has received some negative reward for collisions. For efficient learning, actions should thus be generated in such a way that the environment is explored and pain is avoided. This fundamental trade-off between exploration and exploitation demands efficient exploration capabilities, maximizing the effect of learning while minimizing the costs of exploration.

This chapter explores the role of exploration in learning control.

2 What are the Questions?

What are the fundamental questions to ask about the role of exploration in learning control?

Let us begin with the questions characterizing exploration and exploitation. Exploration seeks to minimize learning time. Thus, the central question of efficient exploration reads *"How can learning time be minimized?"*. Accordingly, the question of exploitation is *"How can costs be minimized?"*. These questions are usually opposing, i.e. the smaller the learning time, the larger the costs, and vice versa. But as we will see, pure exploration does not necessarily minimize learning time. This is because pure exploration, as presented in this chapter, maximizes knowledge gain, and thus may waste much time in exploring task-irrelevant parts of the environment. If one is interested in restricting exploration to relevant parts of the environment, it often makes sense to exploit simultaneously. Therefore *exploitation is part of efficient exploration*. On the other hand, *exploration is also part of efficient exploitation*, because costs clearly cannot be minimized over time without exploring the environment.

The second important question to ask is *"What impact has the exploration rule on the speed and the costs of learning?"*, or in other words *"How much time should a designer, who designs an active learning system, spend for designing an appropriate exploration rule?"*. This question will be extensively discussed, since the impact of the exploration technique on both learning time and learning costs can be enormous. Depending on the structure of the environment, "wrong" exploration rules may result in inefficient learning time, even if very efficient learning techniques are employed.

The third central question relevant for any implementation of learning control is *"How does one trade-off exploration and exploitation?"*. Since exploration and exploitation establish a trade-off, this question needs further specification. For example, one might ask *"How can I find the best controller in a given time?"*, or *"How can I find the best controller while not exceeding a certain amount of costs?"*. Both questions constrain the trade-off dilemma in such a way that an optimal combination between exploration and exploitation may be found, given that the problem can be solved with these constraints at all. Now assume one has already an efficient exploration and an efficient exploitation technique. This raises the question *"How shall exploration and exploitation be combined?"*. Shall each action explore and exploit the environment simultaneously, or shall an agent sometimes focus more on exploration, and sometimes focus more on exploitation?

All these question cannot be answered with a single, universal answer for arbitrary adaptive control problems. Optimal trade-off, optimal exploration strategies, as well as optimal combination techniques for exploration and exploitation strongly depend on the learning task at hand as well as the particular learning technique employed. This chapter addresses these issues of exploration, and describes and evaluates exploration techniques in the context of learning control with neural networks. A variety of exploration schemes are surveyed and compared based on a taxonomy of exploration techniques. The effect of exploration on the complexity of learning is analyzed, demonstrating how the choice of the exploration rule can affect learning time and costs in learning control. In addition, we will investigate the trade-off between exploration and exploitation and evaluate different mechanisms for combining exploration and exploitation.

The rest of the chapter is organized as follows. In Section 3 we will present an adaptive control technique based on modeling the environment with neural networks. In Section 4 we describe a taxonomy of exploration systems and survey some exploration heuristics often found in recent literature on adaptive neurocontrol and reinforcement learning. In Section 5 we discuss several ways for combining exploration and exploitation, including a mechanism for selectively switching attention between exploration and exploitation. Section 6 evaluates exploration in finite domains with discrete representations. This section analyzes the impact of the exploration technique on the complexity of learning by theoretical complexity results as well as some experimental results. In addition, the trade-off between exploration and exploitation is illustrated and discussed. In Section 7 we describe an exploration technique for real-valued domains based on neural networks. Here exploration utility is estimated by a neural network, and selective attention is used for combining exploration and exploitation. This description is followed by an empirical evaluation of this architecture on a simple real-valued robot navigation task.

3 Using Neural Networks for Adaptive Control

During the last few years, artificial neural networks have been successfully applied to a variety of adaptive control problems. Parameter estimation algorithms often used in neurocontrol include the backpropagation algorithm [43], CMAC [24, 32], and the Hopfield algorithm [13, 39]. An overview of the current state of the art in neurocontrol may be found in [33], as well as in this volume. In this chapter, we will focus on one particular control scheme: indirect control with adaptive predictive models [16]. In a nutshell, neural networks are trained with the backpropagation parameter estimation procedure to model the input-output behavior of the environment. Optimal actions are generated by search in the input space of these models. Indirect control facilitates the discussion of exploration, although the results presented here are transferable

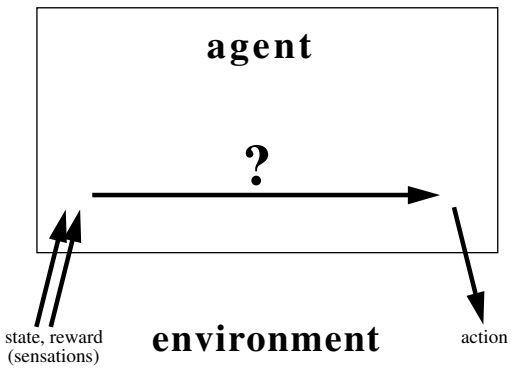


Figure 1: A learning agent observes at each time tick some *sensations* – *state* and *reward* – and has to select an *action*. The learning task is to select actions which maximize reward, i.e. a mapping from sensations to actions (reactive controller) is searched.

to other adaptive control schemes as well. The reader already familiar with this control scheme may skip this section.

3.1 The Environment

From the point of view of an intelligent agent, the most general definition of an *environment* (*world, plant, domain*) appears to be a mapping from *actions* $\times$ *sensations* to *sensations*, as depicted in Figure 1 [34]. That is to say, each time an agent selects an action, it will observe some sensations afterwards. These sensations are a function of the possibly unknown initial state of the environment, all actions performed so far, and some randomness. The task of the agent is to select actions such that it eventually will observe some *desired sensation*. For example, a mobile robot which is to hunt and collect mugs, tries to make the observation "all mugs are found and collected".

We will restrict this very general view of an active learning problem to a more useful one by refining the definition of the environmental mapping. At each time t, the environment is in a specific *state*, described by a state vector s^t. Each time an action a is performed, the environment changes its internal state to a subsequent state s^{t+1}, and gives some *reward* r^{t+1}. This reward, often referred to as reinforcement or payoff, is a scalar measuring the overall-performance of the agent: the larger the reward, the better the actions selected by the agent. Consequently, the desired sensation is exactly the *best possible reward* r^*, and the reward function itself defines the task to be learned. Learning from rewards is more difficult than supervised learning. Although a learning system always receives some immediate reward as response to an action, it will never get information concerning which action would have been the best, as it does in supervised learning tasks. Therefore an agent can figure out optimal actions only by comparison. But after an action is performed, the environment might be in a different state, which keeps the agent from trying other actions in the same state.

For simplification, we assume that both states and rewards are completely observable, avoiding problems of incomplete and noisy perception (see [3, 9, 28, 42, 46, 52, 60] for approaches to incomplete perception). In order to apply neural networks using the backpropagation training procedure [43], we further assume that the environment behaves sufficiently deterministic, i.e. that a deterministic function is capable of modeling the environment. This is an important restriction which is typically found in neurocontrol literature. Note that we do not require the inverse model to be unique. There may be many different optimal actions which lead to the same state and reward.

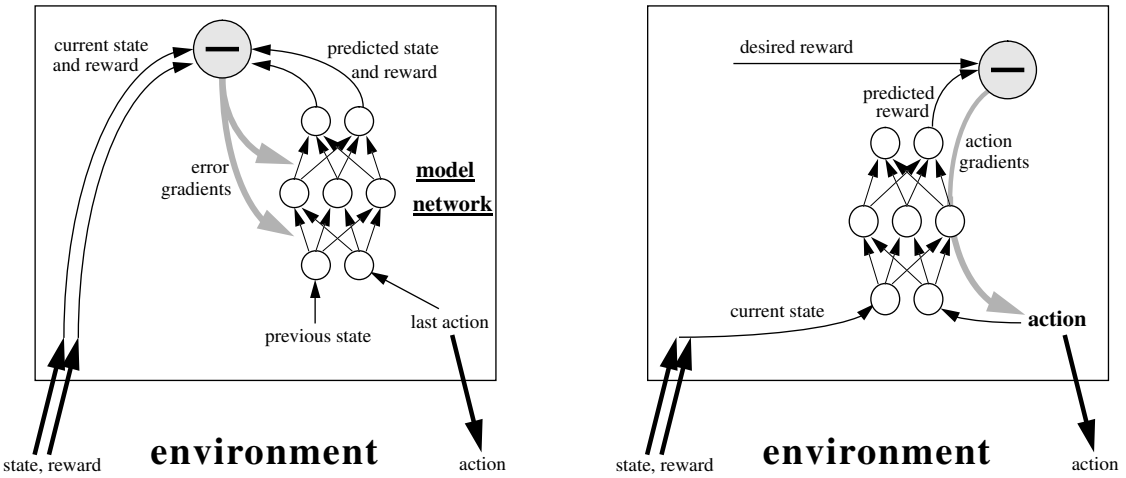


Figure 2: (a) The training of the model network is a system identification task. The model network predicts state and reward, which are compared with the environmental sensations. In order to progressively improve the model of the environment, weights and biases of the network are adapted by gradient descent using the backpropagation algorithm (grey arrows). (b) Using the model network for generating actions (exploitation). Starting with some initial action, gradient descent through the model network progressively optimizes actions with respect to the predicted reward.

3.2 System Identification with Backpropagation

The control scheme used in this chapter is based on adaptive predictive models. Each time an action a is performed, the resulting next state s and reward r are predicted by a multilayered backpropagation network [43], in the sequel referred to as *model network* (Figure 2a). These predictions $\hat{s}$ and $\hat{r}$ are then compared with the real state and reward using the squared *prediction error function*

$$E_{\text{pred}} \quad = \quad (\hat{s}-s)^T(\hat{s}-s) \quad + \quad (\hat{r}-r)^T(\hat{r}-r) \tag{1}$$

Here ^{T} denotes the transpose of a vector. Since backpropagation networks are differentiable, gradients of E_{pred} with respect to the internal parameters of the model, namely weights and biases, may be computed. These gradients are used for changing the weights and biases of the model network in small steps, in order to decrease the prediction error. Given a sufficiently small stepsize, the model approximates the environmental mapping and allows more or less accurate predictions. The reader may note that this process may give only suboptimal solutions for various reasons, e.g. local minima in parameter space of the model, non-uniform distributions of training examples, wrong size of the model network, randomness in the environment and/or the observations, non-stationary environments, and so on. However, non-linear system identification using the backpropagation algorithm has been shown empirically to be applicable to adaptive control, e.g. [16, 17, 19, 20, 40, 41, 44, 56] as well as several chapters in [33].

3.3 Indirect Control with Adaptive Models

In indirect control, actions are generated by gradient descent search in the action space of the predictive model. Given a current state and an initial action (e.g. randomly generated), the model network is employed for predicting the immediate next reward $\hat{r}$ (Figure 2b). The deviation of $\hat{r}$ from the best possible reward r^* is measured by the so-called *exploitation function*

$$E_{\text{exploit}} \quad = \quad -(\hat{r}-r^*)^T(\hat{r}-r^*) \; .$$

Gradients of E_{exploit} with respect to the action are computed and used for changing the action in order to progressively maximize reward, very similar to the way in which weights and biases are changed by backpropagation for system identification. Given a sufficiently small stepsize, this process yields a suboptimal action, namely the suboptimal action next to the initial action, which typically is found by gradient descent optimization. Actions obtained by maximizing E_{exploit} are said to *exploit* the environment.

One nice property of using backpropagation for system identification is that it usually yields considerably smooth functions with few local minima. As a result, action selection by gradient descent usually yields considerably good actions. However, other search techniques such as genetic algorithms [11, 12] are also applicable. A second nice property of backpropagation networks stems from the fact that backpropagation networks with small weights are almost linear. Weights are usually initialized with very small values. Therefore, in the beginning the error function E_{pred} has only one global minimum, which will be found by gradient descent, as long as the weights are small enough. However, with growing weights the network becomes more and more non-linear, and of course gradient descent is not guaranteed to find the best solution possible.

3.4 On-line Look-ahead Planning

So far, actions are optimized with respect to the immediate next reward. However, this control scheme can easily be extended to larger finite horizons by iteratively concatenating copies of the model network. The algorithm for optimizing actions with respect to the next N rewards is basically the same. Let the actual time be denoted by t. Given an initial plan of length N, i.e. N initial actions for time t till $t+N-1$, the model network is used for predicting N future rewards. First, the current state s^t and the first action a_1 of the plan are used for predicting the state $\hat{s}^{t+1}$ and the reward $\hat{r}^{t+1}$, as described in the previous section. This procedure is then iterated N times, using the predicted state and the corresponding action as an input for the next prediction. Actions are now optimized with respect to an N-step look-ahead exploitation function

$$E^N_{\text{exploit}} \quad = \quad - \sum_{\tau=t+1}^{t+N} (\hat{r}^\tau - r^*)^T G(\tau)(\hat{r}^\tau - r^*)$$

Here G is a non-negative diagonal matrix which weights the optimization. Look-ahead planning is able to deal with delayed reward, if this delay is bounded above by a known upper bound N.

In [20] and [56], look-ahead planning based on neural networks is successfully applied to real-time control of a robot arm. For example, in [56], the task is to touch a rolling ball with a robot arm. Recurrent neural networks [10, 15, 61] are trained to predict the robot arm behavior as well as the movement of the ball, and look-ahead planning with $N = 5$ allowed for touching the ball in real-time in about 70% of the cases. However, planning with larger horizons suffered significantly from local minima due to gradient descent, as well as large prediction errors for future rewards, since prediction errors are cumulative.

4 Exploration Techniques

Having described a particular *exploitation* scheme based on using neural networks for forward modeling, we will now focus on the main topic of this chapter: exploration. Throughout this chapter we will distinguish between two major families of exploration schemes: *undirected* and *directed exploration* [53]. *Undirected exploration techniques* explore an environment based on randomness. The most uninformed undirected exploration technique is *random exploration*, where actions are generated randomly with uniform probability distribution, regardless of expected exploration costs. For example, a robot using random exploration will collide with obstacles over and over again, regardless of earlier experiences. Other undirected exploration techniques described here take exploration costs into account by modifying the probability distribution with

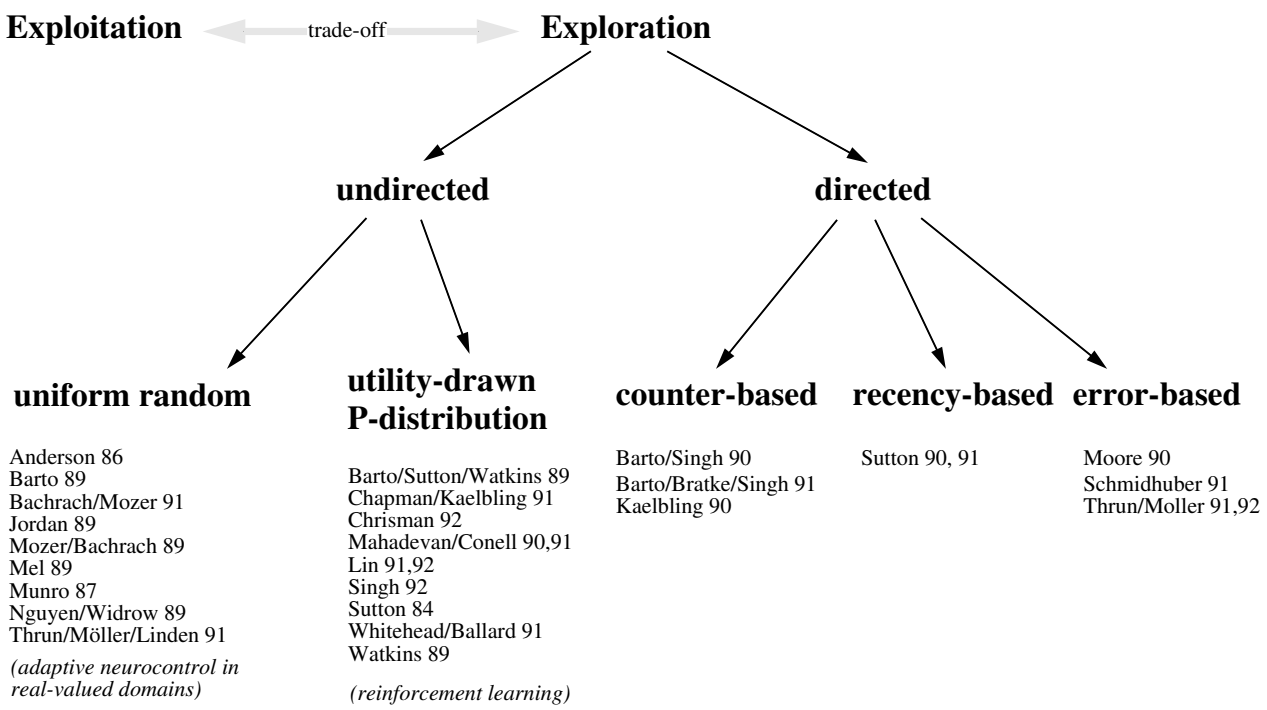


Figure 3: The exploration scene

which actions are selected. Costs are gradually avoided by making more expensive actions less likely. *Directed* exploration differs from undirected exploration in that it utilizes some exploration-specific knowledge for guiding exploration. Rather than selecting actions randomly, actions are selected such that the expected knowledge gain is optimized. The underlying exploration-specific knowledge captures how well parts of the domain are explored, and is used for assessing the exploration utility of actions. Directed exploration techniques thus allow for directly selecting actions which maximize the exploration effect. Although the superiority of directed exploration has been recognized by several authors [6, 18, 36, 50], undirected techniques such as random walk are often found in recent literature.

4.1 Undirected Exploration

Undirected exploration techniques are characterized by using randomness for exploration. The most basic and uninformed undirected exploration technique is called **random exploration**. Actions are selected randomly with uniform probability distribution, and the environment performs a random walk in state space. Random exploration does not take costs or rewards into account at all. In many applications of adaptive control, one distinguishes between a *learning phase* and a subsequent *performance phase* (exploitation phase), and costs count exclusively in the latter phase. Random exploration is frequently found in recent literature to adaptive neurocontrol, e.g. in [1, 3, 16, 17, 31, 33, 37, 38, 40]. However, there are undirected exploration techniques that take costs into account during learning. These typically use the actual controller for assessing the expected costs (or rewards, respectively) of actions. Actions are randomly selected, but the expected rewards are used for drawing the probability distributions with which actions are selected: the better an action is expected to be, the more likely it is to get selected. We will describe two basic ways for modifying probability distributions, both taken from recent literature on reinforcement learning.

Assume we have a controller that determines which action a^* maximizes exploitation. **Semi-uniform distributed exploration** selects actions with a slightly modified probability distribution: a^* is selected with a distinct, large probability P_{best}, and with the remaining probability $1 - P_{best}$ an arbitrary action is selected with uniform probability distribution. Consequently, semi-uniform distributed exploration performs exploitation with probability P_{best}, and random exploration otherwise [9, 29, 30, 60].

A second way for modifying probability distributions is used in **Boltzmann-distributed exploration.** Many learning algorithms used in adaptive control, e.g. indirect control with forward models, or reinforcement learning based on policy iteration methods [8], allow for estimating the exploitation utility of each action separately. Instead of having a black box controller which generates the best exploiting action a^* only, the utility $u(a)$ of every action a is evaluated and compared: a^* maximizes $u(a)$. In Boltzmann-distributed exploration these utility estimates are used for weighting exploration and exploitation, taking the utility of *all* actions a into account:

$$P(a) \quad = \quad \frac{e^{\vartheta^{-1} \cdot u(a)}}{\sum_{a' \text{ is action}} e^{\vartheta^{-1} \cdot u(a')}}$$

Here $P(a)$ denotes the probability for selecting action a, and ϑ is a positive constant value, often referred to as *temperature*. Thus, the higher the exploitation utility $u(a)$ of action a, the more likely a is to get selected. Boltzmann-distributed exploration is often found in reinforcement learning literature, if the number of actions is finite [5, 27, 47, 57].

In general, undirected exploration techniques select actions stochastically, whereas actions with a larger exploitation utility are equally or more than likely to get selected than actions with a smaller exploitation utility. Undirected exploration techniques cover a large spectrum of stochastic action selection mechanisms, which monotonically map exploitation utilities to action selection probabilities. Exploration based on non-uniform probability distributions, such as semi-uniform distributions or Boltzmann distributions, still relies on randomness, but costs are partially avoided. These techniques are often found in domains with finite actions spaces – it is more difficult to realize modified probability distributions in real-valued domains. In exploration techniques based on modified probability distribution exploration is achieved by randomness only, and exploration is thus undirected. As we will show in Section 6.1, undirected exploration may be inefficient in learning time under certain circumstances, meaning that the expected learning time may scale exponentially with the size of the state space.

4.2 Directed Exploration

Directed exploration techniques utilize some exploration-specific knowledge for guiding exploration search. Instead of selecting actions randomly, the exploration rule directly determines which action to take next in order to explore the environment best. The ultimate goal of directed exploration is to select actions which maximize the improvement of the resulting controller over time. However, this is impossible to determine since one cannot know in advance how an action will improve the performance of an agent in an unknown or partially unknown environment. For this reason, the directed exploration techniques described in this chapter are heuristic in nature – they use heuristics for optimizing knowledge gain. In particular, exploration may be achieved by selecting actions and/or states which have been selected less frequently [6], less recently [50], are assumed to have a high prediction error [36], have previously shown high prediction error [45], or are a combination thereof. We claim that directed exploration techniques are usually more efficient than any undirected technique in terms of both learning time and learning costs. In Section 6.1 we will present a theorem which shows the superiority of directed exploration for many finite deterministic domains. A connectionist generalization of directed exploration to real-valued domains is described in Section 7.

Counter-based exploration relies on an adaptive map $c(\cdot)$, counting the occurrences of each state s. This map is used for driving an agent to less explored states. A simple instance of this exploration scheme is given by the rule "go to the least visited neighboring state". For example, counter-based exploration may be achieved by always selecting the action which maximizes

$$E^{\text{counter}}_{\text{explore}}(a) \quad = \quad \frac{c(s^t)}{E[c(s^{t+1})|s^t,a]} \quad = \quad \frac{c(s^t)}{c(\hat{s}^{t+1}(s^t,a))} \; . \tag{2}$$

Here s^t denotes the current state, $E[\cdot|\cdot]$ denotes the expected value, and the next state $\hat{s}^{t+1}$ is predicted by the next-state function of the predictive model. Note that maximizing $E^{\text{counter}}_{\text{explore}}$ realizes the rule "go to the least visited neighboring state".

Kaelbling [18], p. 147, suggests a way to perform directed exploration by pure exploitation, similar to the idea in counter-based exploration. Her key idea, based on *Q-learning* by Watkins [57], a control scheme which allows estimating the exploitation utility of each state-action pair individually, is to initially overestimate the exploitation utility of each state (or state-action pair, respectively). Exploitation will then prefer unexplored actions (and thus states) due to their large utility estimate. The same effect can be achieved by assigning negative reward (costs) to each action, and initialize the exploitation utilities with 0 [5], as it is done in Markov decision problems [14, 22]. In a similar fashion, the *interval estimation algorithm* by Kaelbling [18], pp. 55-77, is also based on overestimating of the utility values. In this algorithm, exploitation utilities are expressed as *upper bound of the expected reward*, conversely to Q-Learning, where exploitation utilities approximate the *mean* of the expected reward. The statistic used for estimating the utilities ensures the correctness of this upper bound with some pre-given high probability. Due to this definition, the upper bound is initialized with the largest possible reward. The statistical function which re-estimates utilities is in part a function of the *number* of experiences, and utility values are carefully decreased on the basis of experience. In this algorithm, exploration is also achieved by initially overestimating utilities, and the resulting exploration effect is a combination of the rule "go to the least visited neighboring state" and error-based exploration, as described below. All these techniques merge exploration and exploitation knowledge. Consequently, after some time, the utilities approach the true exploitation utilities, and directed exploration is gradually reduced. Therefore these techniques are often combined with a second exploration technique.

A straightforward extension of counter-based exploration is **counter-based exploration with decay**. At each time tick, every counter is multiplied with a fixed decay $\lambda \lesssim 1$:

$$c(s) \quad \longleftarrow \quad \lambda \cdot c(s) \qquad \forall \text{ states } s$$

Decay improves the efficiency of exploration, if the exploration effect of visiting a particular state tends to vanish over time, as it is the case for most learning systems that have some learning and/or generalization dynamics. Counter-based exploration with decay also takes the *recency* of occurrences of states into account, by weighting more recent occurrences of states more strongly.

A more rigorous way to take the *recency* of occurrences of states as a basis for exploration is realized by **recency-based exploration**. This technique favors states for exploration which occurred least recently, e.g. by maximizing

$$E^{\text{recency}}_{\text{explore}}(a) \quad = \quad \sqrt{E[\rho(s^{t+1})|s^t,a]} \quad = \quad \sqrt{\rho(\hat{s}^{t+1}(s^t,a))} \; . \tag{3}$$

The function ρ measures the *recency* of a state, i.e. the time during which a state did not occur. Equation (3) may be found in Sutton's work [50, 51], referred to as *exploration bonus*. Recency-based exploration does not take the number of occurrences into account.

Error-based exploration: Some directed exploration techniques aim at estimating the prediction errors of an adaptive controller directly. Exploration is established by driving the agent into regions in state space for which the error has been estimated to be large. Error-based exploration schemes make the assumption that

states or regions in state space with large error are little explored and demand further exploration. This is achieved by maximizing

$$E^{\text{error}}_{\text{explore}}(a) \quad = \quad E[\hat{\varepsilon}(s^{t+1})|s^t, a] \quad \stackrel{\text{e.g.}}{=} \quad \varepsilon_{last}(\hat{s}^{t+1}(s^t, a)) \; .$$

$\hat{\varepsilon}(s)$ denotes an error estimate of state s. Clearly, the efficiency of this exploration rule depends on the heuristic employed for estimating errors. For example, Moore [36] describes a built-in heuristic for determining a confidence value, which can be viewed as error estimate (see below). If the learning algorithm at hand requires several visits of a state, and the error is sufficiently smooth in time, errors $\hat{\varepsilon}(s)$ may be estimated as a function of previous prediction errors $\varepsilon_{last}(s)$ at state s, as e.g. suggested by Schmidhuber [45]. This is usually the case for backpropagation networks. Note that error-based exploration is a promising candidate for exploration in dynamic, non-stationary environments.

Local versus non-local exploration: An important distinction of directed exploration techniques is given by the amount of global knowledge used for exploration. *Local* exploration techniques select actions only with respect to the immediate expected next sensations, i.e. subsequent states and rewards are not explicitly taken into account. The time for action selection does not depend on the size of the state space (usually it is at most linear in the number of possible actions, if this number is finite). Conversely, *global* exploration takes future effects with a larger horizon into account. For example, Sutton [50] shows how in finite domains exploration knowledge from the whole state-action space can be used for exploration by using dynamic programming. In the same way, the exploration techniques described above may be used for global exploration. Non-local exploration techniques are more complex in time, because they require planning. In this chapter, we focus exclusively on local exploration, since these are applicable in real-time.

Adaptive versus non-adaptive exploration: Directed exploration techniques may also be grouped into adaptive and non-adaptive techniques. Moore [36] describes a non-adaptive exploration rule which is directly based on the control knowledge. He uses memory-based storage techniques, i.e. all experiences are explicitly stored using a tree-structure for efficient retrieval. For each action he evaluates the *confidence* in the prediction by the euclidian distance to the nearest experience in the memory where this particular action was used. The action which minimizes confidence is selected for exploration. Moore demonstrates that his exploration scheme outperforms random exploration.

In Moore's scheme, as well as in many other hardwired exploration schemes (see e.g. [25]), the exploration value of an action is evaluated by a pre-given, non-adaptive rule, very much related to the internal representation of experiences and the learning task at hand. However, the techniques described here rely on an adaptive exploration map. This map is trained to make predictions about the controller itself, and exploration is achieved by stating the *goal of exploration* (i.e. an exploration function E_{explore}) rather than the selection criterion itself. However, the distinction between adaptive and non-adaptive exploration techniques is vague, evaluations as well as a precise definition are missing, and it is therefore not yet clear how meaningful this distinction is.

5 Combining Exploration and Exploitation

One main issue in this chapter is exploration and its interaction with exploitation. Up to this point, we have investigated exploration and exploitation mostly independently. This raises the question how to combine exploration and exploitation in order to deal with this trade-off efficiently. We have already described a solution for undirected exploration, namely modified probability distributions. Since the emphasis in this chapter is on directed exploration techniques, we discuss how to combine directed exploration and exploitation.

5.1 Static Linear Combinations

Static linear combinations establish the most basic and straightforward way for combining exploration and exploitation. Actions are selected to optimize a fixed, linear combination thereof. Recall that E_{exploit} is used for evaluating the exploitation utility of an action. Exploration is achieved by maximizing the *exploration function* E_{explore}, with E_{explore} being e.g. $E^{\text{counter}}_{\text{explore}}$, $E^{\text{error}}_{\text{explore}}$, or $E^{\text{recency}}_{\text{explore}}$. For combining exploration and exploitation, actions are now optimized with respect to both target functions by maximizing

$$E \;=\; (1-\Gamma)\cdot E_{\text{explore}} \;+\; \Gamma \cdot E_{\text{exploit}} \;. \tag{4}$$

Here Γ ($0 < \Gamma < 1$) is a fixed gain, determining the portion of both target functions in action selection. Note, in Equation (4), both exploration and exploitation are expressed in terms of optimizing a target function. Both can be viewed as optimization tasks and searched with the same search technique.

5.2 Dynamic Combinations

Exploration and exploitation are often conflicting. The main disadvantage of static combinations is that whenever exploration opposes exploitation, the resulting actions often neither explore nor exploit. For example, a tourist in Africa who suddenly finds himself standing in front of a lion has to decide between exploitation, i.e. running away, or exploration, i.e. approaching and closer examining the lion. If exploration and exploitation are combined with a fixed gain, he might stay where he is, which is clearly the most ineffective solution to the problem. Generally speaking, it might happen that the exploration utility E_{explore} balances the exploitation utility E_{exploit}, and no meaningful action may be found by optimizing E with a fixed gain Γ. It therefore makes sense not to keep the gain Γ constant during learning, but to switch between exploration and exploitation, depending on the expected knowledge gain and costs of an action.

In dynamic combinations, the trade-off parameter Γ is determined dynamically during learning. Dynamic switching mechanisms are based on the "desire to explore", denoted by e_{explore}, and the "desire to exploit", e_{exploit}. If these desires are to reflect expected costs and knowledge gain, the most straightforward definition is:

$$\begin{aligned} e_{\text{explore}}(a, s^t) &= E[E_{\text{explore}} | s^t, a] \quad \text{and} \\ e_{\text{exploit}}(a, s^t) &= E[-E_{\text{exploit}} | s^t, a] \;. \end{aligned} \tag{5}$$

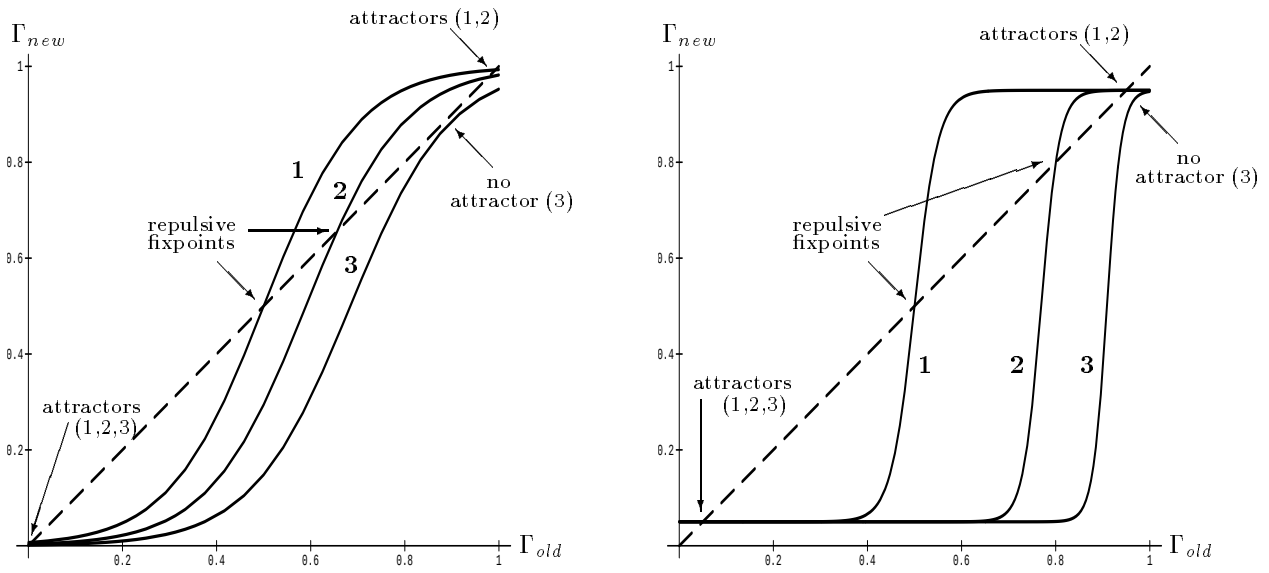
e_{explore} measures the expected exploration utility by selecting a at state s^t, and e_{exploit} measures the expected costs, i.e. negative reward.

Given the desires e_{explore} and e_{exploit}, exploration and exploitation can be dynamically combined by using these values for trading off exploration and exploitation. The most straightforward way is to focus exclusively on one target function:

$$\Gamma \;\longleftarrow\; \begin{cases} 1, & \text{if } e_{\text{exploit}} \geq e_{\text{explore}} \\ 0, & \text{if } e_{\text{exploit}} < e_{\text{explore}} \end{cases}$$

At each time step of action search, the agent either explores or exploits, based on a comparison of both desires. In terms of the above example, the agent compares his desire to flee with his desire to approach the lion, and will then select an action by maximizing the target function which corresponds to the larger desire. After the resulting action is executed by the environment, the agent will decide again whether to explore or to exploit. This simple, greedy rule is only one of many possible implementations for dynamically trading off exploration and exploitation. For example, E_{explore} and E_{exploit} may be multipied with a polynomial function of the desires with high degree. The agent would then switch smoothly between exploration and exploitation.

All these rules have in common that the decision, which function to optimize next, is independent of earlier decisions. In terms of the above example, this may result in the following behavior: If the desire to explore



(a) $c = 3$, $m_{\text{exploit}} = 0$, $m_{\text{explore}} = 0$

curve	f(e_{exploit})	f(e_{explore})
1	1.0	1.0
2	0.8	1.15
3	0.6	1.3

(b) $c = 20$, $m_{\text{exploit}} = 0.05$, $m_{\text{explore}} = 0.05$

curve	f(e_{exploit})	f(e_{explore})
1	1.0	1.0
2	0.6	2.0
3	0.3	3.0

Figure 4: Graphs of the update function for Γ, given by Equation (6) and (7), with stability factor (a) $c = 3$ and (b) $c = 20$. Fixpoints are on the dashed line. Note that graphs 1 and 2 have two attractors each. Updating Γ will thus not switch attention. However, if e_{exploit} and e_{explore} differ too much (curve 3), eventually one of the attractors disappears, and Γ will switch, if it was in its attractor basin.

exceeds the desire to flee, the tourist will move one step towards the lion. At this point the desire to flee may be larger than the desire to explore, and thus the tourist will run away one step. Generally speaking, an agent might start oscillatory behavior by constantly switching between exploration and exploitation.

5.3 Selective Attention

Selective attention [55] is a way for dynamically switching between exploration and exploitation which takes earlier decisions into account. The main difference to the dynamic switching method described above is that Γ is used as *focus of attention*. At each step of action search, both e_{explore} and e_{exploit} are compared *under the current focus of attention*, and the stronger value, weighted by the attention Γ, determines the behavior of the agent as well as the new Γ. The following updating rule for Γ enables switching attention between exploration and exploitation, based on the "desires" e_{explore} and e_{exploit}. With $f(\cdot)$ being a positive, monotonic function (e.g. $f(z) = e^z$), the value κ compares exploration and exploitation *under the current focus of attention*,

$$\kappa \longleftarrow \Gamma \cdot f(e_{\text{exploit}}) \; - \; (1-\Gamma) \cdot f(e_{\text{explore}}) \tag{6}$$

and the new Γ is obtained by squashing κ:

$$\Gamma \longleftarrow \frac{1 - m_{\text{explore}} - m_{\text{exploit}}}{1 + e^{-c \cdot \kappa}} + m_{\text{explore}} \tag{7}$$

Here $c > 0$ is a fixed gain, and $m_{\text{explore}} \geq 0$ and $m_{\text{exploit}} \geq 0$ with $m_{\text{explore}} + m_{\text{exploit}} \leq 1$ are the minimum portion of exploration and exploitation, respectively, in action selection: $\Gamma \in (m_{\text{explore}}, 1 - m_{\text{exploit}})$. For

example, $\kappa > 0$ implies that Γ is closer to m_{exploit}, and the system focuses more on exploitation. Likewise, if $\kappa < 0$, Γ is closer to m_{explore} and the system focuses more on exploration. The focus of attention switches only if κ changes its sign. By taking the current focus of attention Γ into account, the behavior of the agent is stabilized over time. This is because most of the time the rule for updating Γ in Equation (6) and (7) establishes two attractors, close to m_{explore} and m_{exploit}, respectively. This is illustrated in Figure 4. The scalar c serves as *stability factor*: the larger c is, the closer is Γ to its extremal values, and the larger the switching factors $\Gamma \cdot (1-\Gamma)^{-1}$ (taken from Equation (6) with $\kappa = 0$). Note that Γ is initially set to 0.5.

Although selective attention may also yield some multi-step oscillations, there is evidence [53] that dynamic weighting schemes based on selective attention outperform static schemes in terms of exploration costs. However, the reader may note that the switching mechanism described here is only one among many possibilities.

6 Excursion to Finite Domains

In this section we will discuss some exploration results obtained for finite domains, i.e. discrete domains with finitely many states and actions. Such domains have two advantages over real-valued domains, which make them favorable for studying the impact of the exploration rule on learning. First, it is easier to derive theoretical results, and we will summarize some important complexity results for deterministic domains stated elsewhere [53, 59]. Second, it is easier to evaluate and compare exploration heuristics in finite domains, especially those lacking the real-valued counterpart. In finite domains, exploration can be studied without and thus independent of generalization, which necessarily is required for learning in real-valued domains. However, asymptotic analysis of finite domains can be misleading since they allow for complete exploration in the limit.

6.1 Some Complexity Results

The aim of this section is to sketch some theoretical results showing the superiority of directed over undirected exploration in some finite deterministic[1] domains. For simplicity reasons, we restrict the results to state spaces with one explicit *goal state* (absorbing state). The task is then to find a controller yielding the shortest path to the goal state.

Whitehead [58, 59] showed that every learning technique based on random walk is inefficient in time, meaning that the learning time is expected to scale exponentially with the size of the state space, given that

1. the environment is deterministic,
2. there is no a priori information available to guide exploration, the system is *zero-initialized*,
3. the agent receives reward only upon entering the goal state. There is no intermediate information available for guiding the search,
4. random actions change the distance to the goal only by $+1$, 0, or -1, and are expected to *increase* the distance to the goal,
5. the state space is *ergodic*, i.e. for all states s and s', there is a possibility to get from s to s' with non-zero probability (this prevents the agent from getting trapped in a part of the state space which cannot be left and cycling there forever), and

[1] A domain is said to be *deterministic*, if for each state and action the next state and the reward is unique.

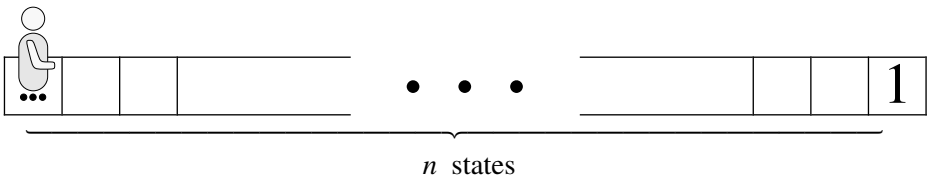


Figure 5: A one-dimensional navigation task. If actions a_1 and a_2 carry the robot to the left, and action a_3 carries it to the right, all *undirected* exploration techniques are expected to take $O(2^n)$ time, while *counter-based exploration* takes $O(n)$ time for finding the goal state.

6. the size of the state space is polynomial in the largest possible distance to the goal state, the *depth* of the state space (this holds for most state spaces studied in literature, e.g. grids of arbitrary dimensionality).

Theorem 1 *(Whitehead, Complexity of reinforcement learning using random walk exploration) Under above conditions, the expected time needed to find the goal state using random walk exploration is bounded below by an expression exponential in the size of the state space*[2]

Since we assume that there is no a priori and no intermediate information provided to guide the initial search until the goal is found (conditions 2 and 3), any undirected technique degrades to random walk in the first exploration run, i.e. before the goal is found and reward is received. Therefore this theorem holds for arbitrary undirected exploration techniques, making it interesting for our considerations. Theorem 1 claims the inefficiency of every undirected exploration technique for the domains it applies to. In Figure 5, such an environment is depicted. The state space is one-dimensional, and the robot has three actions to choose from: one action, a_r, carrying it to the right and two actions, a_{l1} and a_{l2}, carrying it to the left (except on the outermost states). The robot is initially set to the leftmost state, and the goal is to find the state on the right, where and only where positive reward will be received. Whitehead's theorem applies to this environment. In this particular environment, the expected time to reach the goal state is $3(2^n - n - 1)$ with n being the number of states. For example, this time is as much as $3.38 \cdot 10^{15}$ for $n = 50$ states.

However, a simple version of *counter-based exploration*, following the rule: "go to the least visited neighboring state" reduces complexity tremendously. In our example, the robot starts at the left, takes one step to the right, takes another step to the right, and so on, and finally reaches the goal after $n - 1$ actions. Note that this exploration rule demands a predictive model which enables one-step look-ahead[3]. This result can be extended to a more general theorem, stating a polynomial worst-case bound for finding a goal state in arbitrary finite, ergodic, deterministic domains – these domains cover all domains for which Whitehead's theorem is valid:

Theorem 2 *(Complexity of finding a goal state using counter-based exploration with a predictive model) Using the exploration rule "go to the least visited neighboring state", a goal state is always found within the first $O(l \cdot n^2)$ steps, given that the agent has access to a predictive model. (Here $l < n$ denotes the depth of the state space.)*

If we lack a predictive model, system identification may be done on-the-fly, still yielding a polynomial worst case complexity:

[2] We changed the terminology of the theorem in order to fit this framework. See [59] for the original version, the exponential expression and the proof.

[3] This generally undesired condition will be dropped in Theorem 3. Note that Watkins' Q-Learning [57], a learning paradigm for reinforcement learning with delayed rewards, also does not require a predictive model. In fact, several exploration techniques studied here have been proposed in combination with Q-Learning.

Theorem 3
(Complexity of finding a goal state using counter-based exploration without a predictive model)
Using the same exploration rule, a goal state is always found within the first $O(dln^2)$ steps, and a predictive model can be identified in $O(d^2ln^2)$ steps.(Here d denotes the number of actions.)

The proof of Theorem 1 may be found in [59], and the proofs of Theorem 2 & 3 may be found in [53]. It should be noted that all these theorems are valid for deterministic domains only. Similar results do not exist for general stochastic domains, since it can be shown that there are malicious stochastic domains where *any* exploration technique will take exponential time to find a goal state [53].

The theorems demonstrate the potential role of exploration in learning control. The choice of the exploration rule can have a significant impact on the expected search and learning time: Depending on the environment, this time may scale exponentially with the complexity of the environment, but more appropriate exploration techniques may easily prevent this. We believe that these theorems are important for understanding the fundamental difference between undirected and directed exploration. Undirected exploration may spend substantial time in already well-explored regions in state space, and exploration basically happens by chance, whereas directed techniques seek to explore less explored regions by choice. Theorem 1-3 clearly demonstrate that this difference may matter for the complexity of learning.

6.2 Exploration in Reinforcement Learning: A Robot Navigation Task

To finish the excursion to finite domains, we summarize some simulation results obtained on a finite reinforcement learning task with delayed rewards [8, 48, 57]. These results are described in more detail in [53].

Refer to Figure 6a. The task is to navigate the robot from its initial position, i.e. the circle on the left, to the goal position, the circle on the right, with as few steps as possible. States are represented by (x, y)-coordinates – there are roughly 4,500 states. Actions are represented as $(\Delta x, \Delta y) \in \{-1, 0, 1\}^2 - \{(0, 0)\}$, corresponding to the eight nearest neighbors in state space. The task is non-deterministic: actions will be executed correctly with 0.9 probability, but with the remaining 0.1 probability the agent will be carried to a randomly selected neighboring state. Positive reward 1.0 is only received upon the goal state, and negative reward -0.1 is received whenever the agent collides with a wall. Whenever the agent enters the goal state, it is reset to its initial state. Note that a one-step predictive model is given to the agent in advance.

The basic idea of reinforcement learning with delayed rewards [8, 5, 48], as used in this experiment, is to approximate a model of utility of states, where utility is defined as the *cumulative discounted future reward* one expects to receive by entering a state s. In our implementation, the exploitation utility U is defined as:

$$U(s) \quad = \quad \begin{cases} \sum_{\tau=1}^{\infty} \gamma^{\tau} \cdot E[U(s^{t+\tau})] & \text{if no reward is received at state } s \\ r(s) & \text{if reward } r(s) \text{ is received at state } s \end{cases}$$

Here $t + \tau$ denotes future time ticks, $E[\cdot]$ denotes the expected value, $s^{t+\tau}$ refers to the expected actual state at time $t + \tau$, and γ $(0 \leq \gamma \leq 1)$ discounts the future reward over time (usually $\gamma < 1$). Note that this definition of utilities U slightly differs from the standard definition found in reinforcement learning literature [8], since we explicitly distinguish between zero and non-zero reward. Given an accurate estimate of U, optimal future reward will be achieved by always selecting the action which maximizes

$$E_{\text{exploit}}(a) \quad = \quad E[U(s^{t+1})|s^t, a] \; . \tag{8}$$

Thus, a controller can be described in terms of a U-table. Note that this control scheme is equivalent to the control scheme described in Sect 3, except that it is finite and future cumulative reward is modeled instead

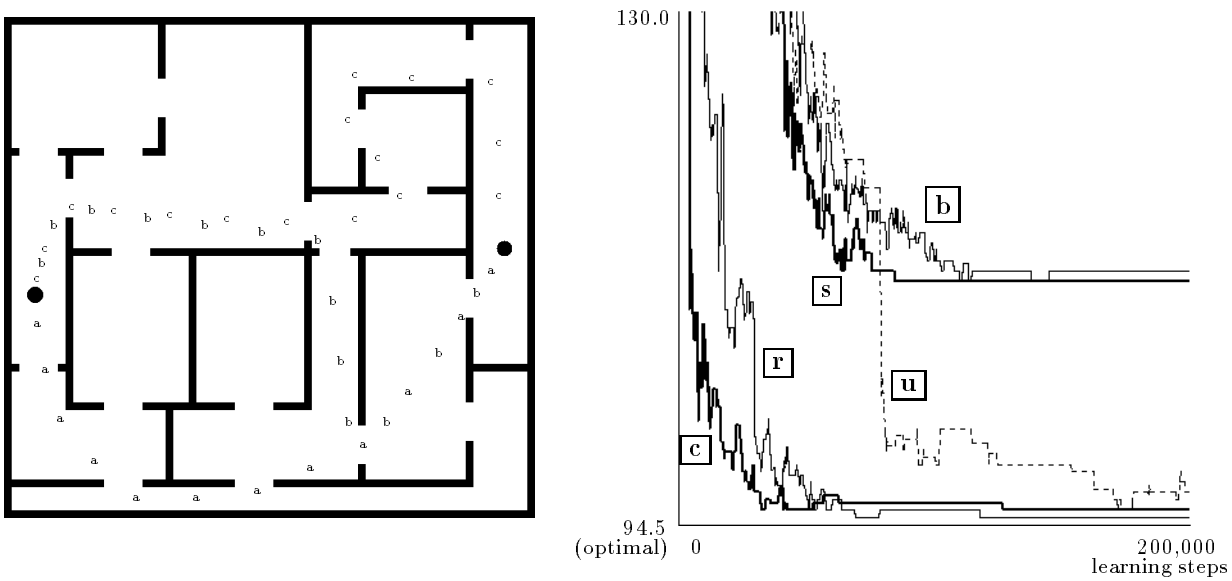


Figure 6: **(a) Task:** The task is to navigate the robot (left circle) to its destination (right circle) on the shortest possible path. Three locally optimal paths are marked: a. with length $L = 94.5$, b. with length $L = 105.8$, and c. with length $L = 117.6$ (euclidian distance, corresponds to the number of steps). **(b) Learning curves** for some exploration techniques. The horizontal axis measures learning time, and the vertical axis the current performance of the controller, evaluated by the shortest path generated by this controller. Undirected techniques: **u** random exploration, **s** semi-uniform distribution, and **b** Boltzmann distribution. Directed techniques: **c** counter-based exploration, **r** recency-based exploration – other directed exploration techniques not shown here performed nearly equivalently.

of the immediate next reward. In order to identify an optimal U-table, we applied a reinforcement learning rule based on real-time dynamic programming [5, 48, 49]:

Step 1:	Initialize: $U(s) \longleftarrow 0$ for all states s.
Step 2:	Select an action, e.g. by maximizing Equation (4).
Step 3:	If reward $r^t \neq 0$, $U(s^t) \longleftarrow r^t$. Goto Step 2.
Step 4:	$U(s^{t-1}) \longleftarrow \max\{U(s^{t-1}), \gamma \cdot U(s^t)\}$ $U(s^t) \longleftarrow \max\{U(s^t), \gamma \cdot U(s^{t-1})\}$ Goto Step 2.

Note that the first update rule in Step 4 establishes asynchronous dynamic programming, aligning reward values in the U-table. The second update rule however inverts this alignment. In the learning task at hand, this is valid since the environment is *symmetric*, i.e. for each action a leading from state s to s', there is an action a' leading from s' to s. Learning was further accelerated by temporarily memorizing each trajectory from start to the goal state (a *"trial"*), and after reaching the goal, replaying it in reverse order. This technique, known as *experience replay* [26, 27], has been shown to be extremely powerful in aligning utility estimates through the U-table.

Results are summarized in Table 1 and Figure 6b. Parameters were carefully optimized by hand for all exploration techniques. The gain Γ, trading off exploration and exploitation, was set such that results were comparable, and the discount factor was $\gamma = 0.995$.

EXPLORATION TECHNIQUE	(a) first trial: average steps (averaged over 50 trials)	(b) costs (average steps per trial)
undirected exploration		
random exploration	43,000	43,000
semi-uniform distribution	43,000	1,000
Boltzmann distribution	43,000	1,760
directed exploration		
counter-based exploration	4,800	520
recency-based exploration	8,900	500
counter-based exploration with decay	7,300	330
counter-/error-based exploration	4,800	530
counter-b. exploration, selective attention	4,800	340

Table 1: Results of all exploration techniques on the finite robot navigation task. (a) In the middle column, the average number of steps of the first trial (finding the goal state) averaged over 50 runs is shown. (b) In the right column, the costs of exploration – here average steps per trial – is depicted. These costs are taken from and directly correspond to the experiments depicted in Figure 6b.

- **Search:** Table 1a demonstrates the significant difference between undirected and directed exploration in terms of average time required for finding the goal (first trial). Undirected techniques, all performing random exploration in the first trial, spent significantly more time until the goal was found.
- **Learning:** Figure 6b shows learning trajectories obtained for some directed and undirected exploration techniques. Note that there is a significant gap between undirected ("b", "s") and directed ("c", "r") exploration, which also exists for all other directed techniques not plotted here. In all experiments we performed, directed exploration yielded much faster learning and always succeeded in finding close-to-optimal paths, whereas undirected exploration either mostly failed in identifying an optimal solution, or, as in the case of random exploration ("u"), succeeded only with tremendous exploration costs, (Table 1b).

6.3 Trading off Exploration and Exploitation

In this section, we evaluate the trade-off between exploration and exploitation on the reinforcement learning task described above. Recall that exploration and exploitation are traded off by the parameter Γ (c.f. Equation (4)). For $\Gamma = 0$, the agent exclusively explores the state space, and for $\Gamma = 1$, it exclusively exploits. This parameter has a strong impact on both expected learning time and costs, which reflect the minimization goals of efficient exploration and exploitation.

In Figure 7, this impact is illustrated on counter-based exploration with decay (with decay $\lambda = 0.9999$ and target function $E = (1-\Gamma) \cdot E^{\text{counter}}_{\text{explore}} + \Gamma \cdot 10 \cdot E_{\text{exploit}}$, c.f. Equations (2), (4), and (8)). This exploration technique was empirically found to perform best in this particular domain. Figure 7 plots (a) learning time and (b) average learning costs as a function of Γ. The most important result of this experiment is shown in Figure 7a: optimal learning time is achieved for $\Gamma^* \approx 0.74$, which divides the domain of Γ into two intervals:

1. **Saturation interval:** In $[0, \Gamma^*]$, increasing exploration will increase both learning time and learning costs. This undesired effect stems from the fact that too much exploration causes a waste of learning time in task-irrelevant parts of the state space. Consequently, Γ^* is better than any $\Gamma \in [0, \Gamma^*]$ in terms of both performance measures.

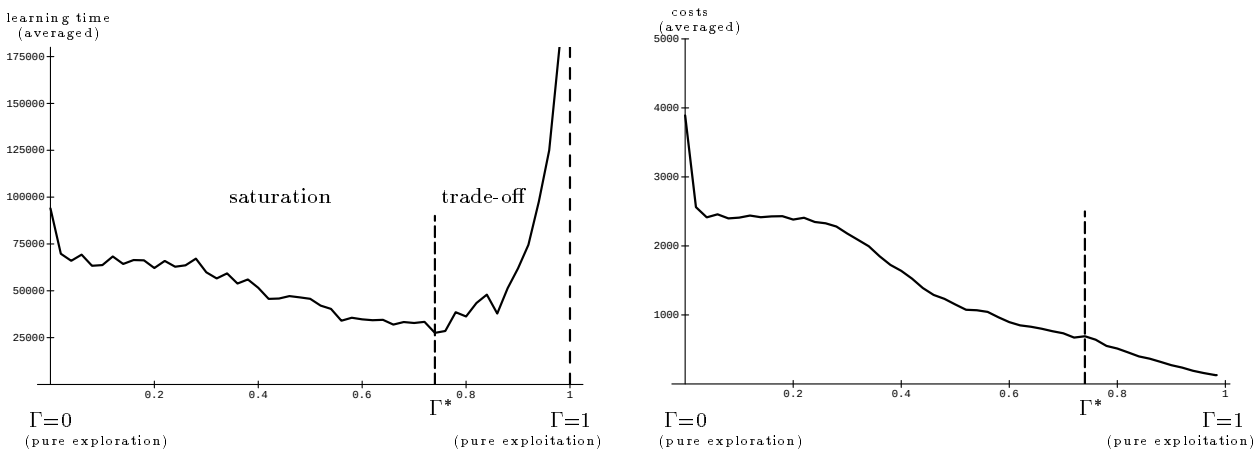


Figure 7: The trade-off between exploration and exploitation. The x-axis measures Γ in both diagrams. (a) **Average learning time**. Learning was finished, if the path found did not exceed the length of the optimal path by 5% (other stopping criteria yielded similar results.) (b) **Average costs per trial**, defined as the average number of steps used for entering the goal state. The cost function is monotonically decreasing in Γ: The more exploitation, the smaller the costs. All results are averaged over 25 runs.

2. **Trade-off interval:** In $[\Gamma^*, 1]$, increasing exploration does improve the learning time, but at the expense of increasing the costs. Learning time and learning costs are thus traded off by Γ only in this interval.

If one is interested in purely minimizing the expected learning time, $\Gamma = \Gamma^*$ is the best choice. Likewise, if learning time does not count and (average) costs are to be minimized only, Γ close to 1 will best achieve this. The performance measure to minimize is often a combination of both learning time and costs, resulting in an optimal value for Γ in the trade-off interval $[\Gamma^*, 1]$. In either cases, Γ should never be in the saturation interval $[0, \Gamma^*]$. This example clearly demonstrates that the exploration goal – namely minimizing learning time – is achieved only by also taking exploitation into account, in order to focus the exploration search on relevant parts of the state space.

These observations raise the question how to determine Γ^* in practice, and how to choose an appropriate Γ. Clearly, Γ^* depends on several factors, some of which are unknown in advance. These include the environment itself, the learning technique applied, and the particular exploration technique at hand, including all their parameters. It seems unfeasible to determine an optimal trade-off Γ directly. However, if a model of the environment is available, an optimal Γ could be estimated in advance based on simulations using this model. Γ could also be optimized during learning, e.g. by some estimation procedure based on stochastic gradients.

7 Exploration with Neural Networks

The general principle of the directed exploration schemes presented in this chapter is to select actions in such a way that the resulting observations maximize the expected knowledge gain in order to optimally improve the controller. In the last section we focused exclusively on finite domains. Such domains are considerably easier to study than domains with real-valued states and actions. If we counted occurrences of states in *real-valued domains*, almost every predicted next state (i.e. with probability 1) would have never occurred before, thus none of the directed exploration rules, as described in Section 4.2, would provide any useful notion of exploration utility. Instead, we need some generalization to real-valued domains which allow transferring exploration knowledge (e.g. counters, recency-values) to local neighbors in state space. In this

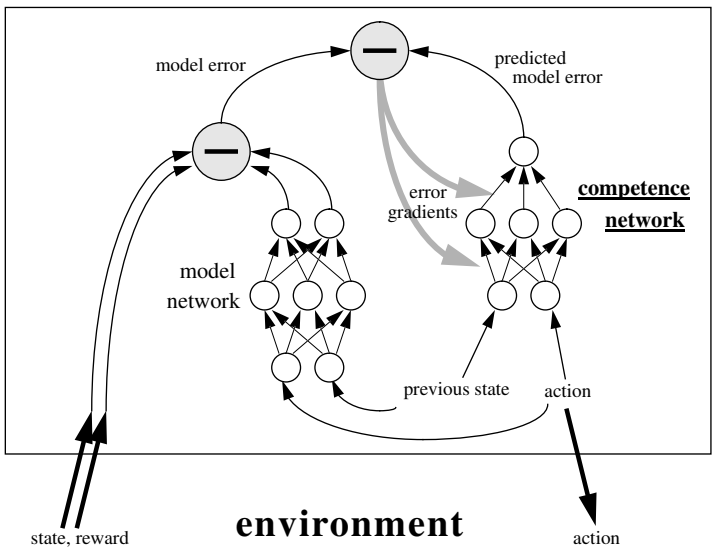


Figure 8: The competence map is trained to estimate the error of the model network. After applying one iteration of backpropagation to the model network (Figure 2a), its model error E_{pred} is used as target value for the competence network.

section we will generalize directed exploration to real-valued domains by using neural networks. A so-called *competence network* is trained to estimate the utility of exploration and is used for guiding exploration [55]. Exploitation is achieved with the control scheme described in Section 3, and selective attention is employed for combining exploration and exploitation (c.f. Section 5.2).

7.1 Directed Exploration: the Competence Map

A competence network $\Pi(s,a)$ is a mapping from states and actions to competence values. Its task is to estimate the accuracy of the model network: for each point in state-action space, the model network predicts the next state and reward, and the competence network estimates the accuracy of these predictions, thus assessing the *competence* of the model network. Since it is in principle impossible to exactly predict the accuracy of the model for regions in state-action space not yet visited, the training procedure of the competence map must rely on heuristics. In turn we will describe the training procedure as well as the underlying heuristics of the competence map.

Each time the weights and biases of the model network are re-estimated by one sweep of backpropagation, the competence map is subsequently trained in two steps, both times with the backpropagation algorithm:

1. First, the competence network is trained to predict the observed LMS-error of the model network (Figure 8). More specifically, the latest input to the model network – previous state and action – is also input to the competence network, but the target is the prediction error E_{pred} of the model network, normalized to the interval $[0, \varepsilon_{\max}]$ $(0 \leq \varepsilon_{\max} \leq 1)$. This normalization is necessary, since output values of backpropagation networks are always in $(0,1)$, and errors may not be.

2. The second training step establishes a heuristic, which realizes the loss of accuracy in unvisited regions in state-action space. Since the domain is real-valued, it is difficult to uniformly decrease the predicted competence for all non-visited state-action pairs, as it is done e.g. by applying decay to counters. Instead, for ν randomly generated state-action vectors, the competence map is trained with target 1,

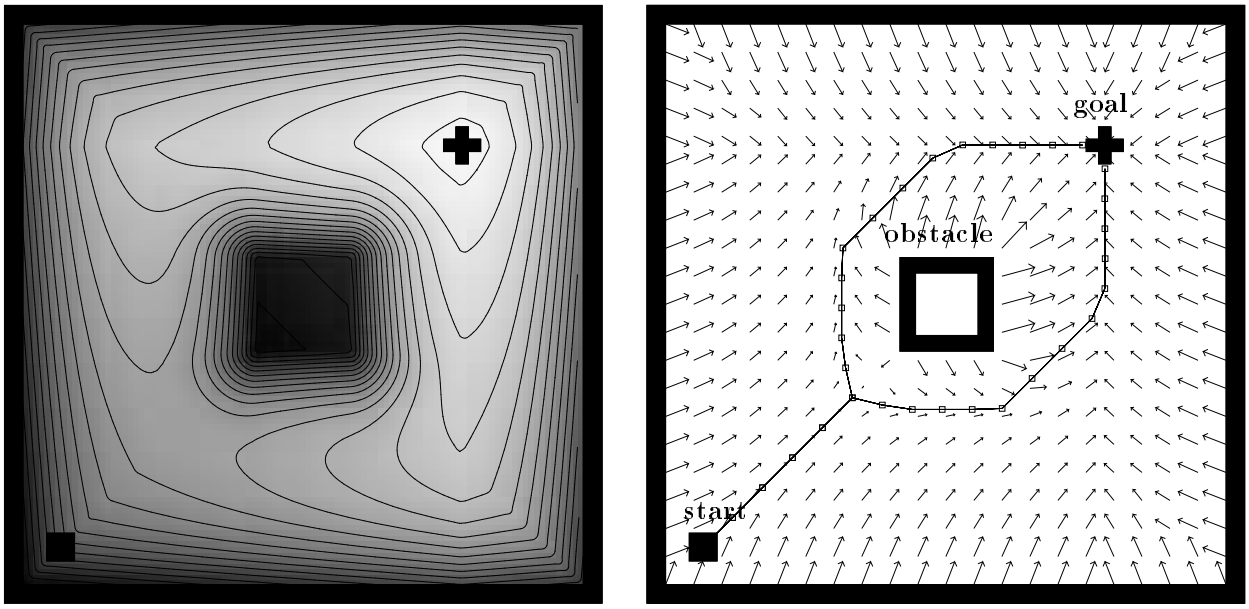


Figure 9: (a) Robot environment: The reward of the environment is a non-delayed, negative *potential field value* (the darker the color, the larger the potential field and the smaller the reward). Learning this potential field function is part of the system identification task. (b) Gradient descent in the potential field yields the optimal paths depicted – note that there are two equally good paths leading around the obstacle.

which is its largest possible output. Here ν is a small number, and the learning rate of backpropagation is divided by ν.

The resulting competence map is now employed for exploration by maximizing

$$E_{\text{explore}}^{\text{competence}} \quad = \quad \Pi(s^t, a) \ .$$

The larger $\Pi(s^t, a)$, the smaller is the competence value for an action, and the larger is its exploration utility. Exploration and exploitation are combined by maximizing Equation (4) with $E_{\text{explore}} = E_{\text{explore}}^{\text{competence}}$.

Note that the parameter $\varepsilon_{\max}$ determines the exploration behavior of the agent. If this parameter is set to 1, the competence map is trained to predict errors of the model network, and the exploration scheme is a generalization of error-based exploration. If $\varepsilon_{\max} = 0$, the competence map is trained with *occurrences* of states only, and most recent occurrences are weighted most. The resulting exploration rule is best described as a generalized mixture of counter-based and recency-based exploration. Therefore the competence map generalizes several directed exploration heuristics described in previous sections.

7.2 A Simple Real-valued Robot Navigation Problem

We will now demonstrate the benefits of directed exploration with a competence map on a simple real-valued robot navigation task. The environment is depicted in Figure 9. The task is to navigate the agent from start (marked by the square) to goal (cross) without colliding with the obstacle or a wall. Whenever the agent succeeds or fails, i.e. touchs either a wall or the obstacle, the trial is finished and the agent is reset to its initial position.

States are represented by $s = (x, y)$ -coordinates in $[0, 1]^2$. Accordingly, actions are represented by $a =$

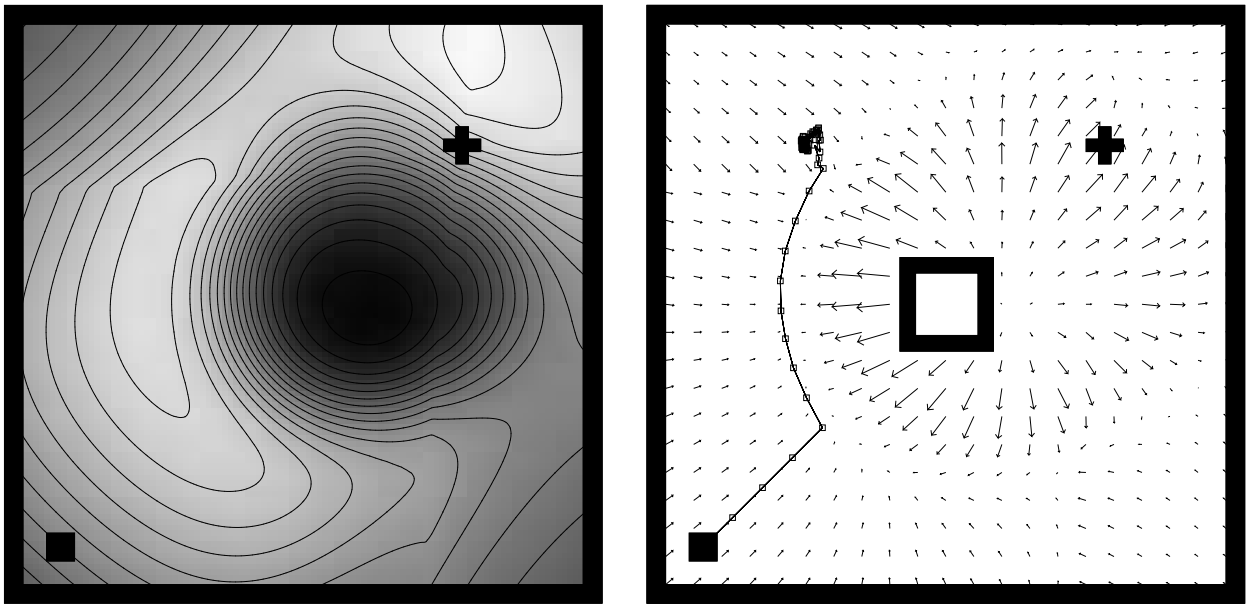


Figure 10: Random exploration: (a) Model of the potential field function. The dark color in the center indicates the large number of crashes against the obstacle. (b) None of the optimal paths is found by maximizing E_{exploit}.

$(\Delta x, \Delta y)$, and steps are performed simply by adding actions vectors to state vectors. Reward is given by a negative non-linear *potential field* function [21], shown in Fig 9b. The potential field at hand is a sum of a term measuring the distance of the agent to the nearest obstacle/wall, plus a term measuring the distance to the goal state. It is clearly seen that maximizing the negative potential field value carries the agent to the goal position from any point in the state space. However, both the state transition function and the potential field function are initially unknown and subject to learning.

Since the reward function is a function of states rather than state-action pairs, the state-transition function and the reward function were modeled separately by two networks in our experiments. State transition was predicted by a network with four input units for state and action vectors and two output units for the predicted next state. The reward function was modeled by a more complex network with two input units for the predicted state vector, one hidden layer with ten units, a second hidden layer with eight units, and one output unit for the predicted reward. The activation functions in the first hidden layer were gaussian functions [35]. The aim of this experiment was to compare three exploration techniques for real-valued domains: random exploration, exploitation, and directed exploration. Learning time was limited to 15,000 steps each. Table 2 summarizes the results.

Random exploration: This exploration technique performed extremely poorly, not at least because the task at hand was disadvantageous for random exploration. Since the agent was restarted after each collision, it was highly unlikely to find the goal by random walk. Thus, most of the training examples for the model network were drawn from positions close to the initial position, which serves as a good demonstration that uniform random exploration may imply a non-uniform distribution of training examples. Figure 10 shows the resulting model of the reward function after 15,000 steps. The dark color in the center indicates the large number of crashes against the obstacle. Using this model for control after learning did not yield any of the paths to the goal state, thus learning failed.

Exploitation: We then evaluated a conservative learning scheme combining exploitation and random exploration: whenever the maximization of the exploitation function E_{exploit} yielded some non-zero action (i.e.

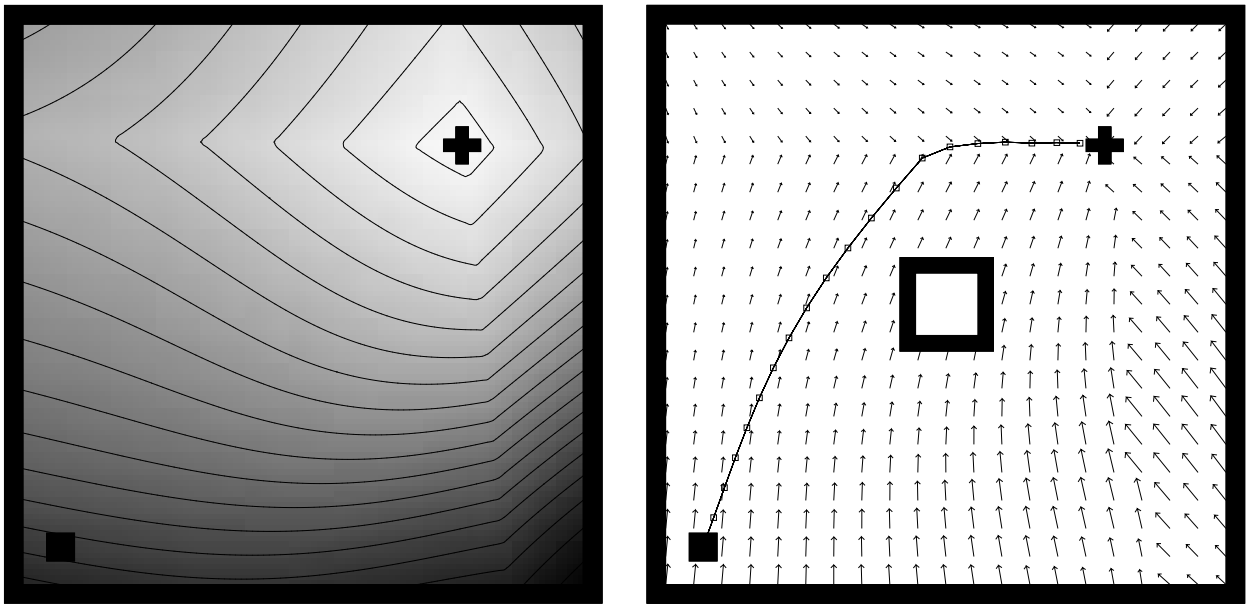


Figure 11: Pure Exploitation: (a) The resulting model is accurate along the path, but lacks the obstacle and is inaccurate elsewhere. (b) Only one of two paths is identified.

whenever the gradients of the model network were not 0), this action was selected without taking any exploration into account – otherwise an action was generated randomly. Since in the beginning of learning pure exploitation lacks the exploitation knowledge, it would always give zero-actions, and the agent would never learn anything. Therefore we could not do without random exploration in the beginning of learning. However, after some random actions, almost all actions were selected by exploiting E_{exploit}.

The resulting model of the reward function after 15,000 learning steps is depicted in Figure 11. Only one path can be found by using this model, and the obstacle is not modeled at all. This model is very accurate along the path, but clearly inaccurate elsewhere. It is worth noting that this approach did not minimize the number of crashes (costs, see Table 2), although it focused almost exclusively on exploitation.

Directed exploration using the competence map: In our experiments, competence was estimated by a neural network with two hidden layers with six hidden units each. The number of random "background" patterns (according to Step 2. of the competence training rule) was $\nu = 5$. Exploration was combined with exploitation using selective attention. Note that in our implementation, we used relative utilities for e_{exploit} and e_{explore} rather than absolute utilities, as given in Equation (5):

$$\begin{aligned} e_{\text{explore}}(a) &= \Delta E_{\text{explore}}(a) &&= \Pi(s^t, a) - \varepsilon_{\max} \cdot E^t_{\text{pred}} \\ e_{\text{exploit}}(a) &= -\Delta E_{\text{exploit}}(a) &&= -(\hat{r}(s^t, a) - r^t) \end{aligned} \tag{9}$$

The stability factor was $c = 100$, and $m_{\text{exploit}} = m_{\text{explore}} = 0.0$, and the function $f(\cdot)$ (c.f. Equation (6)) was $f(z) = e^z$.

Figure 13 shows the resulting competence map as well as some learning paths taken by the agent. It is evident how selective attention switched between exploration and exploitation, depending on expected costs and knowledge gain. In all experiments we performed, directed exploration outperformed both schemes described above and always found the most accurate model, allowing for generating both paths around the

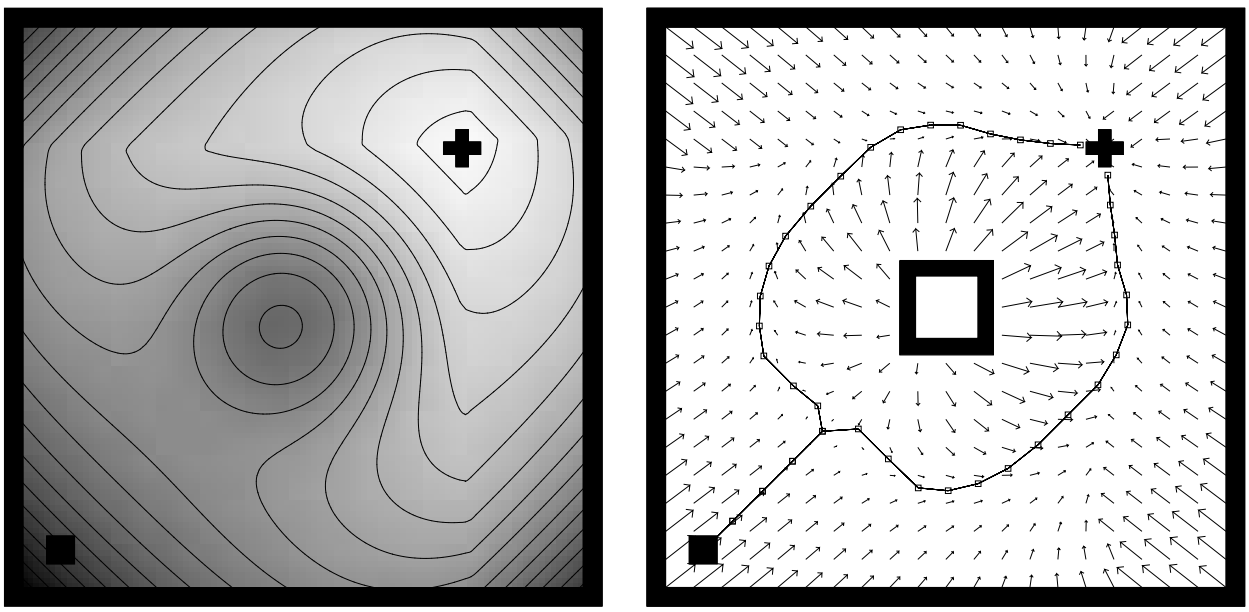


Figure 12: **Exploration with Competence map:** (a) Resulting model of the potential field function. This model is most accurate, and the number of crashes during training is the smallest. (b) Both paths are found about equally often.

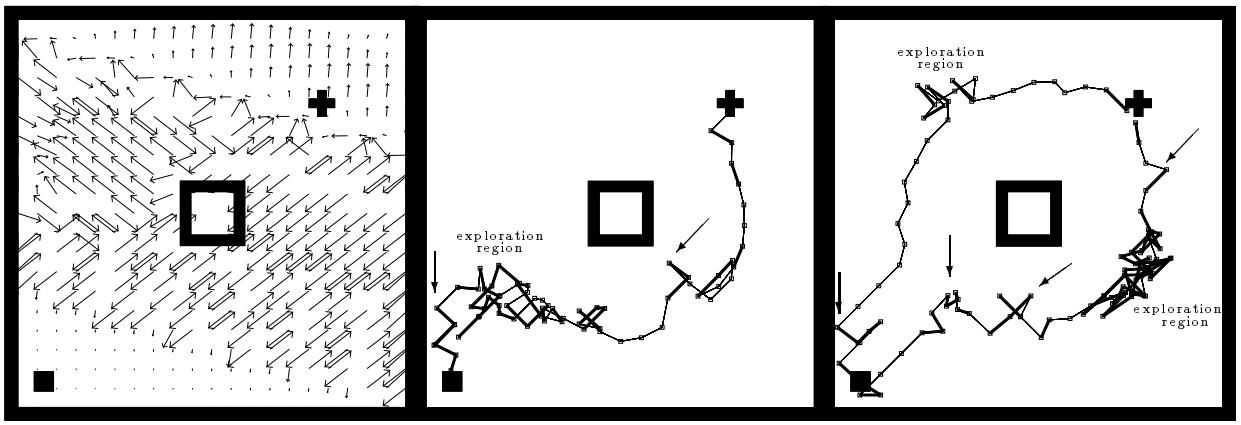


Figure 13: (a) A competence map during learning: The arrows indicate actions which maximize E_{exploit} (i.e. pure exploration). (b), (c) Three examples of trajectories during learning demonstrate the switching attention mechanism described in the chapter. Thick lines indicate exploration mode ($\Gamma < 0.2$), and thin lines indicate exploitation ($\Gamma > 0.8$). The arrows mark some points where exploration is switched off due to a predicted collision.

	# runs	# crashes	# paths found	L_2-model error
random exploration	15,000	14,991	0	2.5 %
pure exploitation	15,000	11,000	1	0.7 %
active exploration	15,000	4,000	2	0.4 %

Table 2: Results (averaged over 20 runs). The L_2-model error is measured in relation to its initial value (= 100%).

obstacle (Figure 12). These results demonstrate the appropriateness of the architecture for directed exploration as well as the appropriateness of directed exploration for learning control.

8 Discussion

The purpose of this chapter is to illustrate the impact of exploration on active learning and adaptive control, as well as to survey and evaluate basic heuristics used for efficient exploration. Reviewing some of the most common heuristics for exploration, we distinguished between two families of exploration schemes: *undirected* and *directed* exploration. We addressed the fundamental trade-off between exploration and exploitation and surveyed some techniques for combining these, including a technique for selectively switching attention between exploration and exploitation. After discussing some complexity results for directed and undirected exploration obtained for finite deterministic domains, we evaluated several exploration techniques on a finite reinforcement learning task. We generalized directed exploration to real-valued domains using neural networks, showing how the results presented in this chapter can be applied to real-valued adaptive control. Finally, the benefits of directed exploration were demonstrated on a real-valued robot navigation task.

There are several limitations to the approach taken, which have to be tackled in order to apply the ideas presented in this chapter to more complex domains.

Complex domains: All directed exploration schemes presented seek to explore the whole state-action space, assuming that it is feasible to explore this space or at least parts of this space exhaustively. This may be reasonable in the experiments presented, but many state spaces, including those typically studied in Artificial Intelligence, are too huge for exhaustive exploration. Instead of searching for the best solution possible, the goal is often to find a good suboptimal solution efficiently. In such cases, an intelligent agent must distinguish between relevant and irrelevant parts of the search space, and cut off exploration in irrelevant parts. To some extent, the combination of exploration and exploitation can restrict exploration search, since exploitation makes the agent focus on more relevant parts, and we have demonstrated how a combination of exploration with exploitation can solve learning tasks faster than pure exploration. In general, ruling out irrelevant parts can also be achieved by providing external knowledge for guiding exploration, which may help to evaluate the exploration utility of actions without actually having experienced them. Exploration techniques must seek to explore only regions where improvements are possible at all. This leads to the very important question of external knowledge.

External knowledge: In many approaches to learning control, some external knowledge is available. This knowledge may be represented by an initial controller, constraints on actions and/or states, domain knowledge in form of rules, pre-given exploration strategies, and so on. External knowledge might drastically reduce the complexity of learning [27, 59]. This chapter does not evaluate the role of exploration if external knowledge is provided. We do believe that directed techniques still do the better job, but clearly an efficient exploration rule should take this knowledge into account.

Exploration and generalization: Optimal exploration is a function of the generalization method used for learning. A generalization method transfers knowledge from a single example to a usually infinite set of related points. The impact of the generalization technique on optimal exploration is obvious. For example, if an agent has the choice between a completely novel action, and a second action similar to some actions selected earlier (similar with respect to the generalization technique), it should try the new action in order to maximize knowledge gain, although both actions might never have been tested before. Optimal exploration rules have to take into account how the generalization technique extrapolates knowledge from examples in order to evaluate the exploration utility. As long as one does not understand how a learning system generalizes, and we believe that this is the case for many learning algorithms like backpropagation, any exploration technique is heuristic in nature. On the other hand, we have seen how such heuristics may decrease learning time.

Acknowledgements

I would like to thank Tom Mitchell for his valuable advice, and Lonnie Chrisman, Sven Koenig, Long-Ji Lin, Tom Mitchell and Don Sofge for excellent comments on earlier drafts of this chapter. Sven Koenig pointed out many important details of exploration in finite domains. I also would like to thank Andy Barto, Ryusuke Masuoka, Knut Möller, Alex Waibel and the reinforcement learning group at CMU for various inspiring discussions. Finally, I am very grateful to Petra Dierkes for her valuable and patient assistance with a foreign language. This work was supported by a grant from Siemens Corp.

References

[1] Charles W. Anderson. Learning and problem solving with multilayer connectionist systems. Technical Report COINS TR 86-50, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, 1986.

[2] Christopher A. Atkeson. Using locally weighted regression for robot learning. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 958–962, Sacramento, CA, April 1991.

[3] Jonathan R. Bachrach and Michael C. Mozer. Connectionist modeling and control of finite state systems given partial state information. 1991.

[4] A. G. Barto. Connectionist learning for control: An overview. Technical Report COINS TR 89-89, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, September 1989.

[5] Andy G. Barto, Steven J. Bradtke, and Satinder P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report COINS 91-57, Department of Computer Science, University of Massachusetts, MA, August 1991.

[6] Andy G. Barto and Satinder P. Singh. On the computational economics of reinforcement learning. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models, Proceedings of the 1990 Summer School*, pages 35–44, San Mateo, CA, 1990. Morgan Kaufmann.

[7] Andy G. Barto, Richard S. Sutton, and Chris J. C. H. Watkins. Learning and sequential decision making. Technical Report COINS 89-95, Department of Computer Science, University of Massachusetts, MA, September 1989.

[8] Andy G. Barto, Richard S. Sutton, and Chris J. C. H. Watkins. Learning and sequential decision making. In M. Gabriel and J.W. Moore, editors, *Learning and Computational Neuroscience*, pages 539–602, Massachusetts, 1990. MIT Press.

[9] Lonnie Chrisman. Reinforcement learning with perceptual aliasing. Submitted for publication, 1992.

[10] Jeffrey L. Elman. Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[11] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[12] John H. Holland. Genetic algorithms and adaptation. In *Proceedings of Ill-Defined Systems*, England, 1984.

[13] John J. Hopfield. Neural networks and physical systems with emergent collectiv computational abilities. *Proceeding of the National Academy of Science (USA)*, 79:2554–2558, April 1982.

[14] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press and Wiley, 1960.

[15] Michael I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, 1986.

[16] Michael I. Jordan. Generic constraints on underspecified target trajectories. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE TAB Neural Network Committee.

[17] Michael I. Jordan and Robert A. Jacobs. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, San Mateo, 1990. Morgan Kaufmann Publishers.

[18] Leslie P. Kaelbling. *Learning in Embedded Systems*. PhD thesis, Department of Computer Science, Stanford University, 1990.

[19] Masazumi Katayama and Mitsuo Kawato. Learning trajectory and force control of an artificial muscle arm by parallel-hierarchical neural network model. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 436–442, San Mateo, 1991. Morgan Kaufmann.

[20] M. Kawato, Y. Maeda, Y. Uno, and R. Suzuki. Trajectory formation of arm movement by cascade neural network model based on minimum torque-change criterion. *Biological Cybernetics*, 62:275 – 288, 1990.

[21] Oussama Khatib. Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[22] Sven Koenig. Probabilistic and decisiontheoretic planning using markov decision theory. Master's thesis, U.C. Berkeley, Dept. of Electrical Engineering and Computer Sciences, 1991. UCB Engin T7.49.1991 K65.

[23] Richard E. Korf. Real-time heuristic search: New results. In *Proceedings of the sixth National Conference on Artificial Intelligence (AAAI-88)*, pages 139–143, Los Angeles, CA 90024, 1988. Computer Science Department, University of California, AAAI Press/MIT Press.

[24] L. Gordon III Kraft and David P. Campagna. A summary comparison of CMAC neural network and traditional adaptive control systems. In W. Thomas III Miller, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, pages 143–169. MIT Press, 1990.

[25] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. Technical report, Department of Computer Science, University of Texas at Austin, Austin, Texas 78712, January 1990.

[26] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of AAAI-91*, Menlo Park, CA, July 1991. AAAI Press / The MIT Press.

[27] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning Journal*, 1992. (to appear).

[28] Long-Ji Lin and Tom M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Submitted for publication, 1992.

[29] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. December 1990.

[30] Sridhar Mahadevan and Jonathan Connell. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 328–332, 1991.

[31] Bartlett W. Mel. Murphy: A neurally-inspired connectionist approach to learning and performance in vision-based robot motion planning. Technical Report CCSR-89-17A, Center for Complex Systems Research Beckman Institute, University of Illinois, 1989.

[32] W. Thomas Miller, Filson H. Glanz, and L. Gordon Kraft. CMAC: An associative neural network alternative to backpropagation. In *Proceedings of the IEEE, Vol. 78, No. 10*, pages 1561 – 1567. IEEE, October 1990.

[33] W. Thomas III Miller, Richard S. Sutton, and Paul J. Werbos. *Neural Networks for Control.* MIT Press, 1990.

[34] Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings of 1990 AAAI Conference*, Menlo Park, CA, August 1990. AAAI Press / The MIT Press.

[35] John Moody and Chris Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[36] Andrew W. Moore. *Efficient Memory-based Learning for Robot Control.* PhD thesis, Trinity Hall, University of Cambridge, England, 1990.

[37] Michael C. Mozer and Jonathan R. Bachrach. Discovering the structure of a reactive environment by exploration. Technical Report CU-CS-451-89, Dept. of Computer Science, University of Colorado, Boulder, November 1989.

[38] Paul Munro. A dual backpropagation scheme for scalar-reward learning. In *Ninth Annual Conference of the Cognitive Science Society*, pages 165–176, Hillsdale, NJ, 1987. Cognitive Science Society, Lawrence Erlbaum.

[39] Kumpati S. Narendra. Adaptive control using neural networks. In W. Thomas III Miller, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control.* MIT Press, 1990.

[40] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE TAB Neural Network Committee.

[41] Dean A. Pomerleau. Rapidly adapting neural networks for autonomous navigation. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 429–435, San Mateo, 1991. Morgan Kaufmann.

[42] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceedings of Foundations of Computer Science*, 1987.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II.* MIT Press, 1986.

[44] Charles Schley, Yves Chauvin, Van Henkle, and Richard Golden. Neural networks structured for control application to aircraft landing. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 415–421, San Mateo, 1991. Morgan Kaufmann.

[45] Jürgen H. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Technische Universität München, 1991.

[46] Jürgen H. Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506, San Mateo, 1991. Morgan Kaufmann.

[47] Satinder P. Singh. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning Journal*, 1992. (to appear).

[48] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, Department of Computer and Information Science, University of Massachusetts, 1984.

[49] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.

[50] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning, June 1990*, pages 216–224, 1990.

[51] Richard S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 471–478, San Mateo, 1991. Morgan Kaufmann.

[52] Ming Tan. Learning a cost-sensitive internal representation for reinforcement learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 358–362, 1991.

[53] Sebastian B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA 15213, January 1992.

[54] Sebastian B. Thrun and Knut Möller. On planning and exploration in non-discrete environments. Technical Report 528, GMD, Sankt Augustin, FRG, 1991.

[55] Sebastian B. Thrun and Knut Möller. Active exploration in dynamic environments. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1992. Morgan Kaufmann. (to appear).

[56] Sebastian B. Thrun, Knut Möller, and Alexander Linden. Planning with an adaptive world model. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 450–456, San Mateo, 1991. Morgan Kaufmann.

[57] Chris J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, England, 1989.

[58] Steven D. Whitehead. Complexity and cooperation in Q-learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 363–367, 1991.

[59] Steven D. Whitehead. A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, University of Rochester, Computer Science Department, Rochester, NY, March 1991.

[60] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991.

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