Reinforcement Learning
for Parameter Control
of Image-Based Applications

by

Graham William Taylor

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2004

©Graham W. Taylor 2004

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The significant amount of data contained in digital images present barriers to methods of learning from the information they hold. Noise and the subjectivity of image evaluation further complicate such automated processes. In this thesis, we examine a particular area in which these difficulties are experienced. We attempt to control the parameters of a multi-step algorithm that processes visual information. A framework for approaching the parameter selection problem using reinforcement learning agents is presented as the main contribution of this research. We focus on the generation of state and action space, as well as task-dependent reward. We first discuss the automatic determination of fuzzy membership functions as a specific case of the above problem. Entropy of a fuzzy event is used as a reinforcement signal. Membership functions representing brightness have been automatically generated for several images. The results show that the reinforcement learning approach is superior to an existing simulated annealing-based approach. The framework has also been evaluated by optimizing ten parameters of the text detection for semantic indexing algorithm proposed by Wolf et al. Image features are defined and extracted to construct the state space. Generalization to reduce the state space is performed with the fuzzy ARTMAP neural network, offering much faster learning than in the previous tabular implementation, despite a much larger state and action space. Difficulties in using a continuous action space are overcome by employing the DIRECT method for global optimization without derivatives. The chosen parameters are evaluated using metrics of recall and precision, and are shown to be superior to the parameters previously recommended. We further discuss the interplay between intermediate and terminal reinforcement.

Acknowledgements

This work was carried out over two continents, and I have many people to thank on both sides of the Atlantic. I offer my apologies to those I have missed.

First, I would like to thank both of my readers, Professors Otman Basir and Jean-Michel Jolion, for donating their precious summer time to read my thesis, as well as offer helpful suggestions towards its improvement.

At the University of Waterloo, Canada, I would like to thank my supervisor, Professor Hamid Tizhoosh, for introducing me to Reinforcement Learning. We have had many enriching discussions, inside and outside of the research world, over the past three years. Professor Ed Jernigan contributed much to the beginning of my graduate career, and has also been instrumental in moulding the Systems Design Engineering department into what it is today. I would like to extend my gratitude to Gord Lueck, for many years of peaceful but never boring student living, as well as my parents who have put up with many moves and have supported me from the beginning.

At INSA de Lyon, France, first and foremost, I hold deep respect for Dr. Christian Wolf, who provided guidance abroad and played a central role in ensuring that I would return to Canada with a thesis draft in hand. To the other graduate students within the Lyon Research Centre for Images and Intelligent Information Systems, I cannot name everyone here, but thank-you for accepting me and making my day-to-day activities enjoyable. Outside of the classroom, there have been many students, both French natives and fellow exchange students who have contributed to a wonderful experience abroad. I would like to give special mention to Wolfram Eberius, who not only saved the Christmas holidays, but is one of the most unique and sincere people I have ever met.

On both continents, Andria Natale has been extremely supportive and understanding. Neither my days at Waterloo nor in Lyon would have been the same without her nearby.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL), can be described as a computational approach to learning through interacting with the environment. This family of algorithms has been applied successfully in several fields, but within the domain of image-based tasks, the literature has not offered many such examples. The complexity of visual information has been one limiting factor in applying RL to image-based applications. Researchers have also not considered the treatment of single images to be the dynamic environment in which RL is typically applied. A temporal element is present in all environments modelled as reinforcement learning problems, and often, this is difficult to define. Recently, theoretical developments in RL as well as its growing popularity have resulted in research that has shown that RL can indeed be applied to such problems. The majority of results have been in vision tasks, processing low-resolution images, including raw sensory data [5, 20, 69, 68], but results are also beginning to emerge in the processing of higher-resolution images [70, 94].

This work aims to explore the use of reinforcement learning algorithms in image-based applications. It first offers a glimpse of some of the fundamental difficulties faced when combining the two areas, and then proposes some solutions to these problems. We then focus on the specific task of parameter control of multi-step vision or image processing algorithms, and develop a framework of applying reinforcement learning to this particular case. We examine specific questions, such as how to define state, actions and reward, and how generalization may be employed when facing problems of a highly-dimensional and continuous parameter space.

To evaluate the framework, we offer two specific applications. The first, and simpler of the two, is the automatic generation of "brightness" fuzzy membership functions for various images. Images are processed individually, and generalization is not used (algorithms are implemented in a tabular fashion). Next, we approach a significantly more complex problem, that is, the detection of text in images extracted from video sequences for semantic indexing. Here, we use the Fuzzy ARTMAP (FAM) artificial neural network [14] for generalization over states and actions. This work offers the first results of approaching a problem space of both continuous states and continuous actions using the FAM architecture. Selecting an optimal action for a given state becomes a bounded global optimization problem. This is resolved with the use of the DIRECT optimization algorithm, which does not require the gradient of the objective function to be known. The FAM can be then treated as an external entity, whose values are sampled, while keeping the number of samples as low as possible.

This thesis makes a number of contributions. First, it examines some of the wider aspects of RL in image-based tasks - the difficulties and ways to overcome them. Next, it provides a second look at the Fuzzy ARTMAP as a generalization method. It applies this neural network structure to a more complex problem (in both the state space and through continuous actions). Third, it proposes the DIRECT sampling method as a means to overcome the nonlinear optimization when the generalizer is computationally expensive to query and gradient information for the objective function is unavailable. Finally, it looks at two very different image-based applications, one simple enough to use tabular-based methods, and one requiring generalization. A number of novel image features for learning are proposed for the latter.

The outline of the thesis is as follows. Chapter 2 provides an overview of the field of reinforcement learning. Image-based tasks are then introduced, as well as a focus on their relationship with RL and challenges in agent design. We also review related work. Then Chapter 3 focuses on a specific problem involving visual information, parameter control of multi-step algorithms. Ideas from the previous chapter are used to develop a general framework for parameter control by RL. The FAM is introduced and defined as the method of generalization, as well as the DIRECT optimization architecture which is employed when we consider continuous rather than traditional discrete actions. Chapter 4 introduces and provides technical details of two specific applications which demonstrate the effectiveness of our framework. Results of the experiments

from both applications are presented and discussed in Chapter 5. Finally, Chapter 6 offers our reflections on the developments presented in this thesis and ideas for future progress.

Chapter 2

Background

2.1 Reinforcement learning

Here we will provide an overview of the field of reinforcement learning. A more thourough covering of the field can be found in [79, 31, 61]. The basis of RL is an intelligent agent that seeks some reward or special signal, and strives to receive this reward through exploring its environment. It also must exploit the knowledge it has previously obtained through past actions and past rewards (or punishments). This careful balance between exploration and exploitation is crucial to the success of RL. Reinforcement learning is very different to supervised learning, and thus can offer an advantage over the latter in many tasks. The agent does not need a set of training examples. Instead, it learns on-line, and can continually learn and adapt while performing the required task. This behaviour is useful for the many cases where precise learning data is difficult or impossible to obtain. In the following discussion, we will adopt the standard terminology used by Sutton and Barto [79].

The history of RL dates back to the early days of cybernetics. It draws from the fields of statistics, psychology, neuroscience and computer science. It is believed that RL is more animal-like than other learning methods. Like other biologically-inspired intelligent approaches, reinforcement learning underwent a period of hiatus, but as the number of publications in the field can attest, RL has emerged and has been steadily growing for the past 10-15 years. Already many learning algorithms have been proposed, and several so far have been proven to

converge to an optimal solution. Applications have been limited but wide-ranging, from a powerful backgammon-playing agent [82] to an elevator dispatcher for skyscrapers [18] to stability control in power systems [22]. It is worth noting that reinforcement learning models have begun to make significant contributions to fields such as biology, psychology and neuroscience [50], in a way "giving back" to the fields from where such ideas were borrowed many years ago. Section 2.3 treats the current focus of theoretical and applied RL research.

Figure 2.1 illustrates the core components of RL. The agent, which is the decision maker of the process, attempts an action that is recognized by the environment. It receives from its environment a reward or punishment depending on the action taken. The agent also receives information concerning the state of the environment. It acquires knowledge of the actions that generate rewards and punishments and eventually learns to perform the actions that are the most rewarding in order to meet a certain goal relating to the state of the environment. RL relies on prediction learning, that is, when an RL agent receives reinforcement, it must somehow propagate that reinforcement back in time so that all related states may be associated with that future consequence. The many RL algorithms dictate exactly how prediction learning is handled, relying on parameters to assign reinforcement temporally.

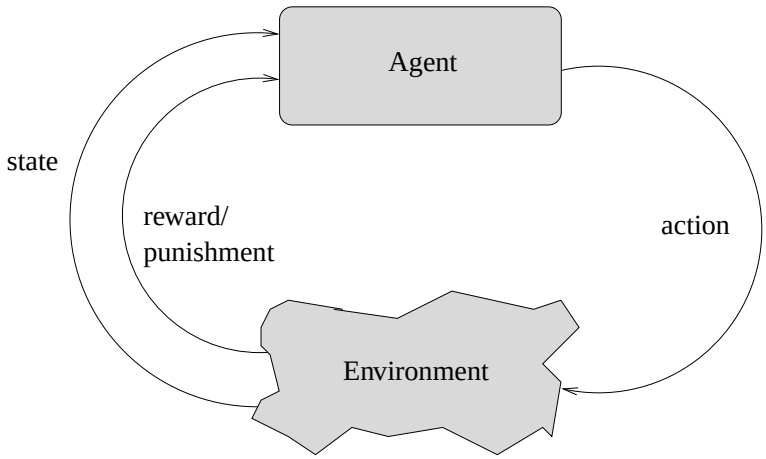


Figure 2.1: The components of a reinforcement learning agent

2.1.1 Markov decision processes

Understanding Markov decision processes (MDPs) greatly facilitates the understanding of reinforcement learning. In this thesis, we model problems as MDPs before applying RL algorithms. This is the most popular representation, but not the only one (see [89, 47, 32, 41] for examples of alternatives). MDPs are a convenient way to model a known stochastic domain, such that optimal behaviour can be calculated. An MDP is a 4-tuple, (**S**, A, r, P), where:

- **S** is a set of states;
- A is a set of actions;
- r is a reward function; and
- P is a state transition function, which specifies probabilistically the next state of the environment, given its present state and agent's chosen action.

In our models, we assume the presence of the Markov property on the environment. This requires that all state dynamics be independent of any previous environment states, rewards or agent actions. Simply stated, all the information or domain knowledge we need to predict the next state and expected reward is provided in a compact representation by the current state and action. This can be formulated as:

$$P(s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0) = P(s_{t+1} = s', r_{t+1} = r | s_t, a_t), \quad (2.1)$$

for all s',r and all possible values of past events,$s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

A stationary policy π for an MDP is a mapping $\pi : \mathbf{S} \mapsto A$, where $\pi(s)$ is the action the agent takes in state s. This concept of policy, which will be explored further, is central to RL. The agent is interested in maximizing the average reward received per time step:

$$\rho^{\pi} = \lim_{T \to \infty} \frac{1}{T} E \left[\sum_{t=1}^{T} r\left(s_t, \pi(s_t)\right) \right]. \quad (2.2)$$

The optimal policy π^* is characterized by a set of fixed-point equations:

$$\rho^* + V^*(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right], \tag{2.3}$$

for all s, where ρ^* is the maximum average reward and $V(\mathbf{s})$ is a value function. For any value function, V, there exists a special policy called the *greedy* policy, which is defined as:

$$Greedy(V)(s) = \arg\max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right]. \tag{2.4}$$

For MDPs, we formally define $V^\pi(s)$ as:

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\}, \tag{2.5}$$

where γ is some discount factor, employed to assign a greater value to more recent rewards. This is the *state-value function* for policy π. Sometimes it is more convenient to consider the value of a particular state-action pair rather than a state. For this case, we define:

$$Q^\pi(s,a) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right\}, \tag{2.6}$$

which is the *action-value function* for policy π. We will discuss the difference between the two approaches in more detail in Section 2.1.2. The greedy policy relative to the optimal value function, V^*, is the optimal policy, $\pi^* = Greedy(V^*)$. This can be applied to both the case of the state-value function and action-value function. The ability to calculate, or estimate the optimal value function is the central problem to reinforcement learning.

2.1.2 Policy estimation

As we have stated in the previous section, our aim is to estimate the optimal value function for a given Markov decision process. Many methods have been proposed for solving this problem. In this section, we briefly review two existing methods, Dynamic Programming and Monte Carlo

methods, and then introduce a third family of algorithms, Temporal-difference methods, which neatly form a bridge between the first two.

Dynamic programming

Dynamic programming (DP) methods require a complete model of the environment, formulated as an MDP. The worst case time DP methods take to find an optimal policy is polynomial in the number of states and actions [12]. Therefore, for certain problems, limited in state and action space, DP methods are efficient. Often, these methods are thought to have limited applicability, because of the *curse of dimensionality*, which will be revisited later on in Section 2.2. As the number of states grows exponentially with the number of state variables, computation time blows up in problems with large state spaces.

Because DP methods require operations over the entire set of MDP states, we must *sweep* the state space multiple times. This is impractical for large state spaces. Fortunately, learning takes place at every step. We do not have to wait until reward is received to update the value estimates of states or state-action pairs. DP methods use a technique called *bootstrapping*, which means that value estimates are updated on the basis of other value estimates. This is an extremely attractive property of DP approaches.

Monte Carlo methods

Monte Carlo (MC) methods differ from DP methods in that they do not require a complete model, or knowledge of the environment. They require only raw experience. MC methods are sampling algorithms, in which returns are averaged. The simplest example of an MC approach would be to record and then average all returns following all visits to a particular state, s. The value estimate of state s would then be updated with this average. Because estimates for each state are independent, the computational expense of estimating the value of a state is independent of the number of states. This is in contrast to the complexity of DP. Therefore, these techniques are able to perform in very highly demensional spaces. This makes them attractive even beyond policy estimation [42]. However, maintaining *sufficient exploration*, that is, enough coverage of the state space, is an issue with such sampling techniques.

Temporal-difference methods

The ability of MC methods to work without a model of the environment and learn directly from experience is attractive. Unfortunately, they do not bootstrap like DP methods, and therefore must always wait for a final outcome before experience can be recorded and learning can occur. Ideally, we desire methods that can learn directly from experience, in a Monte Carlo fashion, but also bootstrap like Dynamic Programming [79]. In this work, we focus on a specific family of algorithms called *temporal-difference* (TD) learning which do just that.

TD learning algorithms strike a balance between the raw experience and non-model nature of Monte Carlo methods and model-based dynamic programming. Their name refers to the appropriate distribution of rewards by the agent over successive steps, or discrete time increments. The idea is that it may take several steps and several more actions before an action taken by an agent results in a reward (or punishment). TD algorithms simply define an approach to distributing this delayed reward.

As we have discussed in Section 2.1.1, reinforcement learning algorithms can be approached through the the policy estimation or prediction problem, in which we estimate a value function, $V^{\pi}(s)$, for a given policy π or the control problem, in which we describe the value of state-action pairs through a function which estimates the optimal policy, $Q^{\pi}(s, a)$. Both approaches are useful. For example, if we were designing an RL agent to play a particular board game, we may wish it to learn the value of certain features extracted from the current board layout, which would define its state. It could then evaluate some number of potential board actions, and resulting states, using a method such as minimax. In this case, we would only need to estimate a state-value function, and thus use the policy estimation approach. On the other hand, the common gridworld problem of an agent navigating some maze with a limited number of actions, such as up, down, left, and right, is usually approached using the control framework. In this thesis, we will focus on the latter, though our framework could be revisited with policy estimation algorithms such as $TD(\lambda)$.

The most widely used and well-known TD control algorithm is Q-learning [87, 88], introduced by Christopher Watkins in his Ph.D thesis. Q-learning is characterized by the following

update equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{2.7}$$

where r_t is the reward received from the environment at time t, α is the step-size parameter, and γ is the discount-rate parameter. It is a form of off-policy TD control. This means that the learned action-value function, Q, directly approximates the optimal action-value function, Q^* independently of the policy being followed. This is seen in the "max" term where the maximum Q value over all actions at the next state is always taken, regardless of the policy.

A similar TD control algorithm, but instead on-policy, is the Sarsa algorithm [63], introduced by Rummery and Niranjin. The update equation for Sarsa (Eq. 2.8) is nearly identical to that of Q-learning, but the next state action pair must be selected (using the policy) before the update can be made:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]. \tag{2.8}$$

Both equations describe an update to the estimated optimal action-value function for the previous state, following an action which causes the agent to transition to a new state. The update algorithm is the core of a reinforcement learning agent.

One criticism of these two algorithms, is that the value function backup is based on only immediate reward. The value of the state one step later is used as an estimate for expected future rewards. However, when we update the value of a state, we may wish to update using not only the next reward, but the reward after that, and other rewards into the future. This is because the outcomes of a particular decision may not be immediate. Of course, future rewards would be discounted, but they should still be considered. A concept called *eligibility traces* provides the mechanism for distributing rewards temporarily. Essentially they are a history of past states and actions which decay as they become more and more distant in the past. The Sarsa-(λ) algorithm works the same way as Sarsa, except that at each update, past visited state and action pairs are also updated with the current reward by using eligibility traces. The strength of the update is proportional to how recently they have been visited. Generally, after a certain number of iterations, we discard the traces associated with old states and actions, as they decayed to

nearly zero. The parameter λ, where $0 \leq \lambda \geq 1$, controls a balance between one-step returns (at $\lambda = 0$), and infinite-step returns (at $\lambda = 1$). The added memory requirements for eligibility traces are often well worth the efficiency gain in learning. While this is enough background to understand the experiments in Chapter 5, we have provided only a brief introduction to eligibility traces. They certainly play an important part in reinforcement learning, and many more details and algorithms may be found in [79].

2.1.3 Action selection policies: the tradeoff between exploration and exploitation

Already we have discussed a great deal about policies. First we established that the agent is trying to estimate an optimal policy. Then we introduced on-policy and off-policy versions of TD learning. But we have said nothing yet about what policy the agent actually follows while it learns. This is one of the most fundamental questions concerning RL, and has been a popular subject of discussion in the literature [56].

An agent changes its current state by executing an action. Given that it knows of Q-values relating each possible action to expected reward, we may think it should always take the action associated with the highest Q-value, the greedy action. This is the naïve answer, as the agent must make a trade-off between immediate return and long-term value. The agent that did not explore unseen states and only attempted to maximize its return by choosing what it already knew would be a poor learner [79]. There needs to be a balance between *exploration* of unseen states and *exploitation* of familiar states.

There are many proposed solutions for this exploration vs. exploitation dilemma [56, 94], but so far no solution has emerged as the clear winner. The action selection policy generally depends on the task at hand. In the experiments that follow, we have used simple but often-used ϵ-greedy policy. This dictates that given a state, we will select the greedy action (that which has highest Q-value) with probability ϵ, and select a random action with probability $1 - \epsilon$. We have experimented with both fixed values of ϵ as well as values that decrease according to a specified formula.

One major drawback of ε-Greedy action selection is that when the agent explores, it chooses

equally among non-ideal actions. This is undesirable when certain actions are much lower valued than others. A policy called *Softmax* action selection solves this problem by varying the action probabilities as a graded function of estimated value. The greedy action is given the highest selection probability, but the other actions are ranked and weighted according to their value estimates. The most common Softmax method uses a Gibbs or Boltzmann distribution [79]. The number of state changes accumulated must be known to the agent. The agent will then choose action a on iteration t with probability

$$P(a|t) = \frac{e^{\frac{Q_{t-1}(a)}{\tau}}}{\sum_b e^{\frac{Q_{t-1}(b)}{\tau}}}, \tag{2.9}$$

where τ is a positive parameter called the *computational temperature*. Higher temperatures will result in all actions being equally probable. As τ approaches 0, Softmax error selection becomes the same as greedy action selection.

A more involved policy, such as Softmax action selection could be employed, but so far there is no proof of it performing better, and often it is difficult to select a value of τ [84]. Another problem with the Softmax action policy is that it requires all Q-values to be ordered and considered. This is computationally impossible in the case of continuous actions (see Section 3.5). To date, the ϵ-greedy policy is still the most popular policy in use.

2.2 Generalization

Ideally, one could assume that the estimates of value functions could always be represented as a look-up table with one entry for each state or state-action pair, depending on whether the policy estimation or control methodology is being used. Unfortunately, this is often not the case, and can be treated as so in only very limited problems. Not only does this approach present unrealistic memory requirements, but for large state spaces, an agent is not able to visit all states or state-action pairs, and thus the time needed to fill these tables becomes increasingly problematic. In the case of a large, continuous state-space, the problem becomes intractable. This is known as the *curse of dimensionality* and requires some form of generalization. Generalization has been extensively studied before the popularity of RL [79], so many of the existing generalization

methods are commonly combined with RL.

The most popular method of function approximation in the reinforcement learning literature has been the local linear function approximator. An example of this is CMAC introduced by Albus [2], also known as tile-coding. Sutton later fortified this method by providing strong empirical results [77], even after it had already been widely used in RL. Tile-coding is convenient, being easily understandable and implementable, but can only handle a relatively low-dimensional state space. More recent research has shown that connectionist systems [86, 49] can handle a much higher-dimensional state space [17]. The well-known multi-layer perceptron (MLP) has been combined with RL in many examples [63, 17]. Unfortunately feedforward MLP networks tend to forget previously learned patterns as more and more significantly different patterns are presented to them. In RL, non-stationary environments are a perfect catalyst of this problem, called *catastrophic interference* [25]. If an example (in the case of RL, a state or state-action pair) is introduced which is very different than what the network has previously experienced, it will adjust its weights in such a way that the old knowledge is permanently damaged, or forgotten. Let us consider the example in which an agent is trained to find the shortest route from point A to point B through a series of streets. After a certain amount of iterations, the agent has learned a near-optimal path. Now, assume that one of the streets along this path is suddenly blocked. The agent must suddenly adapt to this new obstacle, but using the MLP for function approximation, its knowledge of the previous state space may be permanently damaged.

Patrascu and Stacey [56] provide emperical results for a dynamic grid-world RL agent problem that demonstrates that the fuzzy ARTMAP [14] (FAM) vastly outperforms the MLP when the two are compared using a gradient-descent Sarsa algorithm. The FAM is also much less sensitive to algorithm parameters. Like these researchers, we are concerned with non-stationary environments. Our framework must be able to handle many different images, containing various content and acquired from multiple sources. For its past performance, and also for its ease of implementation, we have chosen to adopt this method of generalization for one of the applications introduced in this thesis. The FAM is outlined in Section 3.4.

A common criticism of connectionist systems used as generalization methods in RL is their black-box nature. They function well, and implementation can be done in a modular fashion, but if any problems occur, it becomes very difficult . Besides troubleshooting, we also may want

details regarding the nature of the approximated Q-function, and often this is not convenient when using such systems. Recently, researchers have turned to more statistical-based methods, which not only generalize well, but explain how the state and action space has been generalized. Sallans and Hinton [65] approximate Q-values by the free energy of a product of experts network. This network implicitly represents a joint probability distribution over state-action pairs, and thus is readily queryable and understandable.

A final class of generalization methods will be mentioned here, and that is a class of algorithms that store past training data and process the data at the time of a query. They are known as memory-based methods, instance-based methods, and also lazy learning. When using such methods in RL, value-function estimates are generated at query-time. Simply stated, real world experiences are saved and used to create a value function approximation. Smart and Kaelbling [74] propose an algorithm called HEDGER based on one form of lazy learning, *locally weighted regression* (LWR) techniques, which allow training points closer to the query point to have more influence over the approximation than those further away. This technique can avoid the common extrapolation error of other function approximation techniques by constructing a convex hull around training points, and providing output only for queries within this surface. More details on locally-weighted learning can be found in [6, 7].

2.3 Current areas of research in RL

Reinforcement learning is a wide field, and in-depth coverage of the range of current research interests within it would exceed the scope of this thesis. In this section, however, we aim to highlight some of the major areas in the field. The author wishes to acknowledge that many of these ideas have been proposed by Sutton [78].

2.3.1 Theoretical research

While it has been proven that many of the fundamental RL algorithms converge to an optimal policy (under specific conditions), we are still unaware of the convergence properties of many families. This is especially true for algorithms involving function approximation. It is expected

that newer and more effective generalization methods, such as the one described in [65] will be developed in coming years, and modifications will be made of existing methods. One promising field is that of Neuroevolution [75], where neural network weights as well as topologies are gradually optimized by evolutionary computation methods. Theoretical analysis of new algorithms, notably actor-critic methods [76, 79] and policy-gradient methods [81], as well as function approximation techniques, will follow.

As Equations 2.7 and 2.8 show, the agent aims to maximize expected discounted return. The majority of RL algorithms employ discounting and employ a parameter that determines the current value of future returns, such that rewards received closer to the immediate future are assigned higher value compared to those received far in the future. Alternatively, certain algorithms do not discount, but consider the average reward [44]. Empirical results have shown that these algorithms perform better than discounting algorithms in cyclical tasks. However, this area still remains unexplored relative to the numerous discounted reward algorithms.

Knowledge representation also poses a major challenge to RL. Specifically, we must search for methods to represent temporally abstract knowledge. This domain covers the hierarchical nature of real-world tasks. Any RL problem will inevitably involve sub-problems. Consider the task of making breakfast. This not only includes the high-level decision making of what to make, but also smaller steps like finding the right dishes, or at even a lower-level, moving one's arms. A framework for treating such tasks in a hierarchical nature must be established before we can expect RL to treat a variety of sufficiently complex real-world tasks. Several methods for representing temporally abstract knowledge have been proposed, such as hierarchical reinforcement learning [19, 3, 64], Q-concept learning [62] (which is also a generalization approach), and the method of options [80], but they need to be further evaluated and refined. This area is also closely related to the active research field of multi-agent learning [11]. Researchers in RL will continue to examine how multiple agents may work together to solve a task, in particular, by subdividing the task and tackling these sub-problems individually.

Finally, we have mentioned in Section 2.1.1 that underlying reinforcement learning is the theory of Markov decision processes. We have assumed that the state of the environment is fully available to the agent, though in reality it is often not. Allowing the state representation to be non-Markov [89] opens the field up to a plethora of untreated applications, but also many questions.

This phenomenon is commonly observed in computer vision problems, where objects are often occluded or unrecognizable. If the agent had previously seen an object, and could remember its location, then obviously this would aid in learning. This is common practice for humans: we remember the location of buildings, times of appointments and temperature outside, even if this information is not readily available from sensory input. Current methods of adressing perceptual limitation do include history-based approaches [47] such as n-th order Markov models, but also the framework of *partially observable* Markov decision processes [73, 32] and the more recently developed predictive state representations [41, 72]. These, and other methods will continue to evolve.

2.3.2 Applied research

RL has frequently been applied in some areas such as robotics and game-playing, fields which concern high-level decision making. Its adoption in other environments, has been hampered by some of the theoretical issues discussed in the previous section. As these concerns are better explored and alleviated, RL will continue to be applied to wider-reaching areas. Many applications, including those in image processing and vision, require the treatment of large and continuous state spaces (in which RL has had promising results) but also large and continuous action spaces (which have received less attention in the literature). Of course, there is much room to apply the novel techniques developed in theoretical RL to areas that have previously received less attention.

This thesis presents application-driven research. Specifically, it focuses on image-based tasks, a wide-ranging field. Because so many of the ideas in RL are new, there is no "toolbox" that allows us to immediately start generating results. The problem must, of course, be formulated in a way that it can be treated by RL algorithms. Then ideas from other domains, such as connectionist theory and optimization can be employed so that the problem is manageable. A similar approach would be necessary in applying RL to any new field.

2.4 RL and visual information

As in the previous section, the examples are numerous, and we may only highlight some of the more interesting related research. We focus on addressing the nature of image-related problems and the difficulties they present to RL.

2.4.1 What are image-based tasks?

First, we begin with some definitions. Digital images are visual representations of real-world scenes. Image-based tasks are algorithms and procedures that process this visual information. Visual information may be the information contained in digital images (or videos), the image itself (a digital representation of a real-life scene), information presented as an image, or information perceived by a biological vision system. This data can be analyzed and manipulated in many ways.

This set of algorithms encompasses both image processing and computer vision. Just as high-level vision tasks include low-level image processing operations, higher level decision-making algorithms (as in robotics) may well include vision operations. Common processes involving visual information include:

- Image enhancement;
- Image restoration;
- Object recognition; and
- Image storage and retrieval (in multimedia databases).

Intelligent agents can play an important role in all of these operations. Learning about the environment can be aided by images representing the real world.

2.4.2 Why are image-based tasks difficult?

As discussed in Section 2.1, the state of the environment is a major component of reinforcement learning. We introduced the concept of policy as a mapping from states to actions. In fact,

the goal of RL is to estimate an optimal value function, from which the optimal policy can be built. State construction, therefore, is of utmost interest. It seems natural that agents working with images will incorporate image information into states. But this is a difficult task. Directly generating states from image information results in huge states (often between 0.2-3MB of data per image), that cannot be handled even with value function approximation. Images also often contain noise and irrelevant data, which further complicates state construction. One method of achieving more compact states is through feature extraction, already widely used in pattern recognition and learning. These methods are concerned with extracting the maximum amount of information from images, while keeping the amount of data manageable.

Another challenge lies in the definition of reward. The RL agent learns only through this special signal from its environment, and therefore a reward that accurately represents the goal of the agent is crucial to its success. The reward also must be a single value that represents the completion of a goal or sub-goal. In image-based tasks, reward usually is extracted directly from an image or set of images. In the enhancement task, it is the goodness of the image. In the image retrieval task, it is the similarity of the returned images to the query images. In the segmentation task, it is the accuracy of boundaries. In all of these cases, a human observer can perform a better evaluation than a quantitative metric. Simply stated, humans are much better than machines at both evaluating images and in recognizing patterns. This is related to the fact that images are polysemous: they may possess many meanings, and only one of these may be relevant to the task at hand, defining what is desirable. Even if humans are involved in the evaluation process, their evaluation is subjective. What one person considers to be a “good” image, the next may think to be just “satisfactory”. How to quantify this subjectivity and how to create a system that is observer-dependent are both important questions in image-based learning. Regardless of the evaluation method itself, be it objective or subjective, the translation of such a complex evaluation into a signal that can be processed by the agent may also result in simplifications that can adversely affect learning.

2.4.3 Previous work

The majority of published results concerning image-based tasks and reinforcement learning have been within the field of robotics. Asada et al. [5] published one of the first applications of vision-

based RL. They successfully employed RL to teach a robot to shoot a ball into a goal. However, the visual information had to be significantly processed before incorporated into states. For example, information was extracted from the image, such as size, position and orientation of the ball. Since the state space was kept so small, Q-learning without function approximation was employed. Shibata et. al have been actively pursuing in direct-vision-based RL [69], where robots are fitted with CCD cameras. Raw visual signals are inputted directly to neural networks, without preprocessing, and actor-critic RL is used to train the network. The authors have demonstrated success in applying the framework to real robots seeking targets [30] as well as reaching and then pushing a box [68]. In these examples, a 320×240 pixel display has been cut and averaged to provide an input of $64 \times 24 = 1536$ visual signals. While the approach is effective for the specific task, operations involving higher-resolution images, where information contained in the image is rich and complex could not undergo such a "lossy" processing. Aitkenhead and Mcdonald [1] have looked at the obstacle-navigation task by an autonomous vision-based RL agent. Again, a very low-resolution vision system was used, employing biologically-plausable neural network structures. The researchers attempted to mimic a biological vision system with this network, creating a layered perception system first employing low-level operations such as local feature extraction and segmentation, followed by high-level vision such as object recognition and scene analysis. The raw visual data was translated into overlapping grids which contained gradient information. The experiments also examined supervised training, unsupervised training, and a mix between the two which was called *partially supervised training*. Supervised training provided superior results, followed by partially supervised and then unsupervised training, as expected. However, the latter is more biologically plausable. Intermediate methods such as partially supervised training, where a reward signal can be provided both directly from the environment and by an external teacher informing whether choices are "good" or "bad" are attractive for these complex problems. We explore these ideas further in Section 5.2.5.

Outside the field of robotics, and especially among what are traditionally classified as image-processing applications, the examples are less numerous. Similar to some of the above examples in the types of images used, Ouslim and Curtis [52] have trained a neural network by RL to classify machine parts in low-resolution images. They propose an extension to higher resolution images by increasing the network size. However, images used in these experiments, do not

exceed a resolution of 128×64 pixels. Yin [94] introduces an RL-based approach to multilevel image thresholding, where entropy is used as reinforcement. The results are compared with an existing genetic algorithm-based method, and shown to be superior in quality and computational expense. Hossain et al. [28] use RL to modify the parameters of the segmentation, feature extraction and model matching steps of a pattern-recognition system designed to detect tropical cyclones in satellite images. The authors, however, are unclear on their state space formulation, as well as whether generalization methods are employed.

Our first experiments with image-based reinforcement learning was with the contrast adaptation task [84]. The aim was to examine whether or not RL agents could learn to choose one of a discrete set of simple grey level linear point transformation functions given a degraded input image. State information was constructed from histogram statistics, and reward was normalized Mean Opinion Score (MOS) provided by subjective observers. There were two shortcomings of this work. The first, and more major one dealt with the handling of the temporal aspect in RL. For each image out of a set of training images, the image was selected, its state was computed and then an action was taken, based on the RL policy which resulted in an output image. The state of this output image was computed, reward received and the Q-learning equation was applied. Then we moved onwards to the next image in the training set, whose state was entirely different than the output image of the previous step. As a result, the state of the improved image was neglected and this presented an episodic task with only one step per episode. It leaves us questioning the problem formulation. The second shortcoming was in applying rewards. Using only human observers to evaluate the output images and then provide reward and punishment was very time consuming. This approach is suitable for specialized tasks, in which learning could be performed quickly enough to permit human intervention. On the other hand, using humans to provide a simple feedback signal could be a breakthrough in environments where imaging expertise was not available. Despite these challenges, the experiments provided us with a first glimpse at RL involving images, and provided positive results which encouraged further and more sophisticated investigations.

Shokri and Tizhoosh [70] then looked at the application of Q-learning to global, bi-level thresholding. Again, the implementation was one-step episodic, but the researchers this time considered both subjective and objective reward. The approach was very different from Yin's

RL thresholding approach [94], but empirical results were comparable. Current work focuses on a locally adaptive implementation of the objective reward approach, as well as the application of more complicated learning algorithms.

The majority of successful examples in applying RL deal with decision making at a high-level, where actions are discrete and of low cardinality. The work in this thesis, however, focuses on low-level decisions, particularly in the selection of continuous and highly dimensional parameters. Ideally, agents should be able to make both low-level and high-level decisions, similar to the learning processes of humans. Flexibility over decision resolution will be facilitated by future developments in temporal abstraction.

Chapter 3

A Framework for Reinforced Parameter Control

So far, we have discussed image-based tasks at large. We have already highlighted some of the challenges unique to this field of applications. In this chapter, we focus on a particular problem: selecting an optimal set of parameters for a multi-step computer vision or image processing algorithm by using an RL agent. Here we define the problem itself, and summarize the major difficulties it presents. Then, we develop a general framework with which to approach it. Finally we discuss some of the technical details of the methods employed in our proposed solution.

3.1 Parameter control for an image-based algorithm

The class of algorithms for which we wish to control parameters is wide. While some processes may be classified as image-processing and others as vision (the definition is fuzzy), we do not differentiate between the two. All that we require is that these algorithms manipulate digital images in some way to arrive at some desirable, measurable result. Algorithms may range from something simple (in its number of steps) such as contrast enhancement, or something requiring more operations of varying complexity, such as segmentation. Each of these algorithms contain a series of N steps, and each step, n, is important to the outcome. As well, each of these steps require some parameters, which we can represent in vector form.

Common to all of these algorithms is that digital images will form part, or all of the input. The input image, or set of images also may transformed during the various steps of the process, providing intermediate images.

Output may be in the form of an image, or set of images. It may be some more "desirable" form of the input (as in the case of noise reduction), or it may be some information represented in image form (as in the case of segmented regions backprojected onto an image). Output may even be entirely in a non-image form. For example, it may be a set of co-ordinates marking some points of interest in the image, or another representation of the image, such as a graph. We assume nothing about the output format, just that it can be evaluated, subjectively or objectively. The aim is to to choose a final set of parameters, such that the defined evaluation metric is maximal when the algorithm is employed. We may wish to select parameters in one of two ways, these are:

- Generate a global set of good parameters. that is, for all images; or
- Select parameters with respect to particular images, or sets of images.

Both options are possible with RL, though the latter requires considerably more effort.

Crucial to RL is the temporal aspect. We do not think of the parameter selection problem as choosing all of the parameters at once, progressing through all steps of the algorithm and then evaluating the result. If this were the case, we may be more inclined to employ some other optimization method. Instead, we treat the parameter control as a decision process, where parameters are chosen gradually, whether this be one at a time, or grouped relative to some step. Decisions will always be based on what has been experienced from previous decisions. This considers not only past full executions of the algorithm, but also the previous steps of the current execution. The advantage of a sequential approach is not only the use of past parameter information, but also images that may be generated at intermediate steps. This is further explained in Section 3.3.

3.2 Implementation issues

The nature of the above problem provides several obstacles to the intelligent agent architecture. These include, but are not limited to the following:

- The parameter vector at any step may be highly dimensional, and continuous;
- The parameters may exhibit local maxima;
- Images generally contain a large amount of data, thus complicating the determination of state; and
- Image evaluation is often subjective.

Reinforcement learning, however, is well-suited to the problem. It has the ability to overcome the above obstacles, namely,

- Generalization in RL can be employed to handle problems of high dimensionality and a continuous nature;
- The stochastic nature of RL (through an exploratory policy) can avoid the pitfalls of local maxima;
- We can formulate the problem so as to not include raw image data. This mainly affects state definition; and
- An RL agent interacts directly with its environment, and thus can incorporate subjective evaluation in terms of reward and punishment. An example of this is through the use of Mean Opinion Score [84, 85].

Reinforcement learning also has advantages in terms of its well-founded theory. The convergence to an optimal solution has been proven for many of the fundamental RL algorithms, and thus we are assured of asymptotic optimality [88, 79].

3.3 Fitting the reinforcement learning framework

If we intend to use a reinforcement learning agent to solve the parameter selection problem, we must integrate the general problem into the RL framework. This means modelling the parameter selection problem as a Markov decision process. Limiting ourselves to the control architecture, as described in Section 2.1.2, we must define the states, s, actions, a and reward (or punishment), r.

3.3.1 States and actions

We discuss states and actions together, as one generally follows the other in terms of definition. They are so closely related, that the construction of one may interfere with the other. In order to construct the state space, we must know how to take actions. On the other hand, the state space must already be formed, recognizing the current state, in order to construct the action space [37]. However, this problem concerns aggregation, and adaptive methods such as artificial neural networks can allow state and action space to evolve together. Here we are concerned with the basic formulation of states, before aggregation and generalization are even considered.

A limited number of methods of defining state are seen in the literature with respect to image-related problems. One can use raw image data [52, 20, 69, 68], gradient information [1] or histogram statistics [84] but with high-resolution images, this approach can generally lead to extremely large state spaces, which are undesirable. Images can be pre-processed so that information such as size, location and orientation of objects is extracted and used as state [5]. This may work for specific applications, but it requires the use of sophisticated vision algorithms and significant computational expense. Other unexplored methods may include local statistics, or even subjective evaluation of the image. Though again computationally intense, in the case of having a corrupted image and its original, various filters could be applied to the image, and then their error output could be used to determine state. In fact, any combination of the above methods can be used to determine state. In general, if image-specific results are required, feature extraction should be used to build the state space, where the nature of the features is related to the goal at hand.

As stated in Section 3.1, the temporal dimension cannot be separated from RL. In developing his thresholding agent, Yin [94] recognized that the selection of thresholds represented a decision process, in which higher thresholds, selected later, were dependent on lower thresholds, selected earlier. By treating each image individually, past decisions regarding the thresholds formed state information for the next step. This idea can be extended beyond thresholding, to a multi-step parameter-selection problem in general, even where the parameters are continuous and can be considered at each step as vectors rather than scalar values. We have proposed such an extension for our state/action layout, in which the state is based on the value of previous parameters and the number of the current step of the algorithm. This is represented visually in Figure 3.1.

While Figure 3.1 represents states as discrete nodes, they can be continuous in parameter space. As a consequence, both states and actions will be referenced in vector form. We will use the notation $\boldsymbol{s}_{n,\boldsymbol{p}}$ to imply that any given state corresponds to the current step of the algorithm, n, and the parameter vector, $\boldsymbol{p}$ chosen at the previous step. We will also subscript parameter vectors as $\boldsymbol{p}_{n,i}$ since the dimensionality of the parameter vector is determined by the step of the algorithm to which it is associated. The subscript i is one of an infinite number of parameter vectors chosen at step n of the algorithm. The initial state, $\boldsymbol{s}_{0,0}$ is common, reflecting that we have yet to choose any parameters, and we are at the start of the algorithm. The RL agent then dictates the parameter vector, $\boldsymbol{p}_{1,i}$ for the first step of the algorithm. This corresponds to action $\boldsymbol{a}_{1,i}$. This step of the algorithm is applied using this parameter vector, any intermediate reward is received, and learning takes place. This brings us to $\boldsymbol{s}_{1,\boldsymbol{p}_{1,i}}$, stating that the 1st step of the algorithm has concluded, using parameter vector i. From here, the RL agent then selects the parameter vector for the second step of the algorithm, $\boldsymbol{p}_{2,i}$. Once again, this corresponds to taking action $\boldsymbol{a}_{2,i}$. Any intermediate reward is received, learning takes place and we proceed accordingly until we are finally at state $\boldsymbol{s}_{N,\boldsymbol{p}_{N,i}}$. All parameters have now been chosen, and the algorithm has completed. Final output has been obtained, and terminal reinforcement calculated. Thus, the agent transitions to a common terminal state, $\boldsymbol{s}_{N+1,0}$, and one interation of the algorithm is complete.

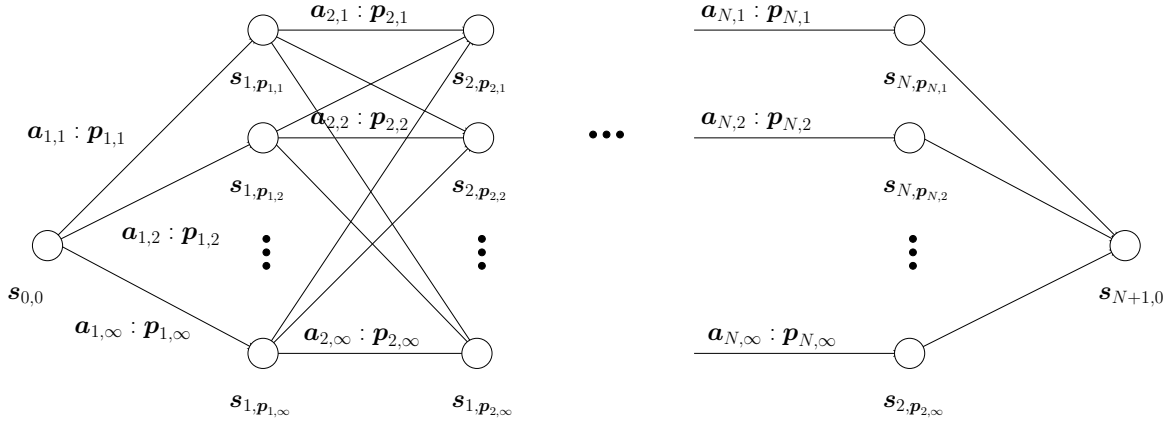


Figure 3.1: Constructing states

In addition, we incorporate the optional consideration of image information (through contin-

uous features). This is practical only when using some method of generalization, as it greatly complicates the state space. This is demonstrated in Figure 3.2. Here, the state space remains the same as in Figure 3.1 (as indicated by the greyed nodes), with the addition of another dimension. States now depend on the algorithm step and past parameters as before, as well as a set of features extracted from the image, or set of images available at that step of the algorithm. As a result, there is no longer a common starting node, if features are extracted from the input image. This information is used to formulate the initial state. The terminal state, however, remains common.

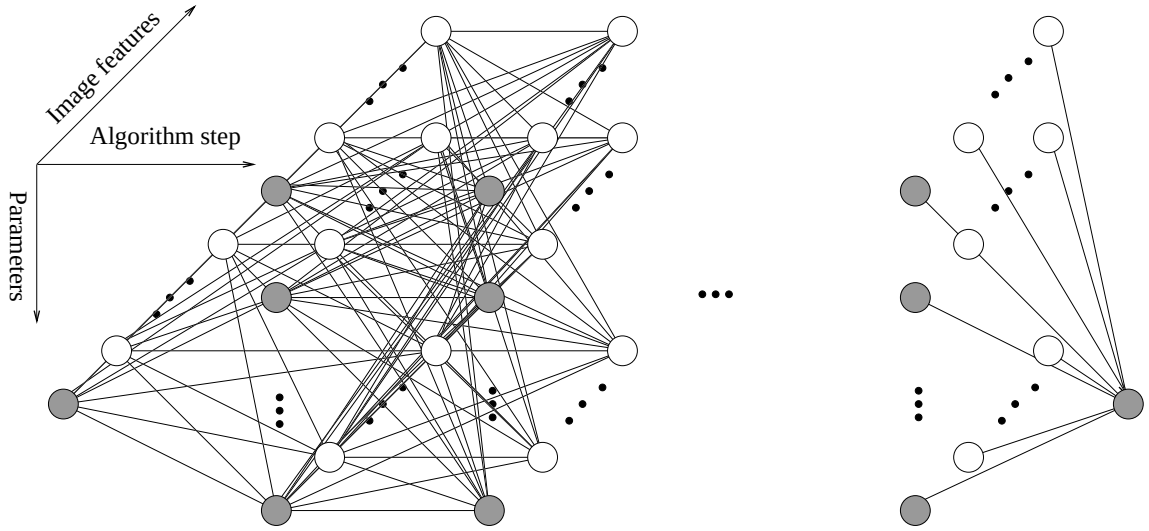


Figure 3.2: Constructing states, considering image features

We note that if there are intermediate steps at which no parameters are required, the algorithm simply proceeds and for the next parameter decision, uses the last chosen parameter set to form state input. If image features are included, then they are extracted from the most recent image or set of images available.

For our definition of states and actions, we also considered another way to state the temporal aspect of the problem. This was the idea of the state being simply the vector of parameters, and actions being modifications of one of these parameters. The agent then does not interact with the computer vision algorithm at each step. It simply modifies the parameter set and waits for the algorithm to complete, and subsequently provide a reward. Unfortunately, this approach fails in two respects. First, it does not scale well to continuous parameters, as it is difficult to define

the actions which modify the parameter set. How many, and which parameters should change at each step? By how much should any given parameter change, and should this amount change throughout learning? Second, we lose the ability of the agent to interact with the image-based task at each step. This interaction allows the agent to receive intermediate rewards which may further guide learning. It also may consider information from intermediate images, generated at different steps of the task. We should not neglect the intuitive notion that parameter decisions in sequential algorithms may be dependent on the choice of prior parameters.

3.3.2 Rewards

On one hand, since the reward is provided by the environment and thus external to the agent, its definition does not affect the learning framework to the same degree as state and action. On the other hand, reward definition is as important as that of state and action, as it must correctly indicate the goal (and sub-goals, if any) of the agent. Due to its external definition, we have the ability to experiment with different reward functions without redefining the framework.

The reward function conveniently allows for the introduction of image information that has been kept out of the definition of state space. When the agent has chosen a set of parameters for some step of the computer vision algorithm, that step can then be applied to receive some intermediate, or output image. Then we can use our evaluation metric to produce a reward. The evaluation can either be subjective or objective. The former, however, as performed in [84, 70] requires the participation of human operators which can make experimentation lengthy and expensive. Regardless, human input is desirable in certain algorithms, for example, in interpreting medical or industrial images. While the operator may have little knowledge about the image adaptation or enhancement process, their experience in the field allows them to judge the output far better than any machine. Observer-dependent adaptation would allow several operators to generate enhanced images fit to their individual liking.

While the broad definition of state and actions in the framework will be applicable to many tasks, the reward itself is highly specific to the goals of the particular application. Thus, we will describe the details of reward calculation in Chapter 4, following the introduction of the specific applications presented in this thesis. We therefore do not place any restrictions upon the reward, other than it must be applied as a single numerical signal, and must accurately reflect the aim of

the task.

3.4 The fuzzy ARTMAP

Given that the state space is derived from parameters that are often continuous and highly dimensional, generalization should play an important role in state space formulation. The sole case in which tabular methods may be applied, is when the parameters are relatively few and can be discretized (see Section 4.1).

After having considered many of the methods presented in Section 2.2, we prepose the fuzzy ARTMAP (FAM) architecture for generalization of the continuous state and action space in one of the applications discussed later in this thesis (Section 4.2). The FAM is adopted to map state-action pairs to their Q-values, $Q(s, a)$. In other words, it represents the current estimate of the action-value function (Section 2.1.1). It has been selected to address the problem of catastrophic interference faced by neural networks in dynamic environments. It is specifically designed to balance plasticity and stability and we therefore expect it to be able to adapt to continuously changing image sources. Patrascu's previous empirical results using the FAM with RL have been positive [56], but to the best of our knowledge, this is the only published work combining FAM and RL. The relationship is worth investigating further, especially with more complex applications such as the one presented later in this thesis. The rest of this section provides an overview of the FAM at a level necessary to understand how and why it has been integrated into the proposed RL framework. Full details of its structure and algorithm are given in [14].

The FAM is a supervised learning system which emerged from the field of adaptive resonance theory (ART) [21]. It is capable of growing its recognition categories in response to both binary and analogue input patterns [14]. This is in contrast to its predecessor, ARTMAP, which classifies inputs by a set of binary input features. FAM is therefore a more general system, accepting a fuzzy set of features as a pattern of fuzzy membership values between 0 and 1.

At the core of the FAM are two fuzzy ART modules. Figure 3.3 pictures a single fuzzy ART module. This structure performs unsupervised clustering on its input vectors. It consists of three layers: the input layer, F_0, the comparison layer, F_1 and the recognition layer, F_2. The first two layers contain M nodes, where M is equal to the length of the input patterns, and the

recognition layer contains N nodes. N will grow to meet accuracy criteria, which is controlled by a parameter, ρ, called *vigilance*. Each node in the F_2 layer represents a particular input category. Associated with each F_2 node, j, is a weight vector, $\boldsymbol{w}_j = (w_{j1}, \ldots, w_{jM})$. This weight vector is essentially a prototype vector for the category it represents.

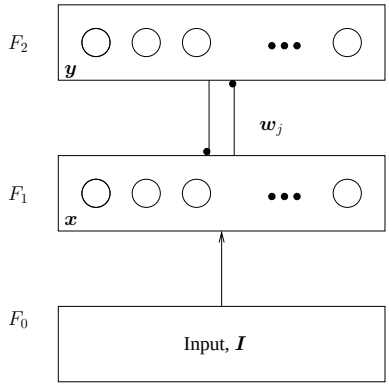


Figure 3.3: The fuzzy ART unsupervised learning architecture

When an input vector, $\boldsymbol{I} = (I_1, \ldots, I_M)$, is presented to the system, the network attempts to classify it into one of the existing categories, based on its similarity to that category's prototype, $\boldsymbol{w}_j$. To do this, we must define the *choice function*:

$$T_j(\boldsymbol{I}) = \frac{|\boldsymbol{I} \wedge \boldsymbol{w}_j|}{\alpha + |\boldsymbol{w}_j|}, \tag{3.1}$$

where $\alpha > 0$ is a choice parameter, and the fuzzy AND operator is defined by:

$$(\boldsymbol{p} \wedge \boldsymbol{q})_i \equiv \min(p_i, q_i), \tag{3.2}$$

and where the norm, $|\cdot|$, is defined by:

$$|\boldsymbol{p}| \equiv \sum_{i=1}^{M} |p_i|, \tag{3.3}$$

for any M-dimensional vectors, $\boldsymbol{p}$ and $\boldsymbol{q}$. The network will calculate the choice function for

every node, j, and then make choice of category by selecting the F_2 node J such that:

$$T_J = \max \{T_j : j = 1, 2, \ldots, N\}. \tag{3.4}$$

The F_1 layer activity vector, $\boldsymbol{x} = (x_1, \ldots, x_M)$, is then set to:

$$\boldsymbol{x} = \boldsymbol{I} \wedge \boldsymbol{w}_j, \tag{3.5}$$

and we compare it to the current input vector, $\boldsymbol{I}$, by the *match strength*:

$$\frac{|\boldsymbol{I} \wedge \boldsymbol{w}_J|}{|\boldsymbol{I}|}. \tag{3.6}$$

The F_2 layer activity vector, $\boldsymbol{y} = (y_1, \ldots, y_N)$, will always reflect the current chosen category. That is, $y_J = 1$ and $y_j = 0$ for $j \neq J$. The vigilance parameter, ρ is now used to evaluate the match strength. If

$$\frac{|\boldsymbol{I} \wedge \boldsymbol{w}_J|}{|\boldsymbol{I}|} \geq \rho, \tag{3.7}$$

then we say that the input sufficiently matches the category prototype for the chosen category, J. The network then learns the current input pattern by adjusting the weight vector for that category so that it is closer to the input pattern. The following learning rule is employed:

$$\boldsymbol{w}_J^{\text{new}} = \beta(\boldsymbol{I} \wedge \boldsymbol{w}_J^{\text{old}}) + (1 - \beta)\boldsymbol{w}_J^{\text{old}}, \tag{3.8}$$

where $0 \geq \beta \leq 1$ is the learning rate parameter (different, of course, to the RL learning rate). If the category weight, $\boldsymbol{w}_J$ does not sufficiently match the input (i.e. the match strength is less than the vigilance), a *match reset* occurs, which means that the F_2 node is not considered for the rest of the presentation of the current input. Therefore, another category (with next highest T_j value) will be chosen and then its prototype will be tested for match strength against the input. This process of elimination will continue until either a prototype with sufficient match strength is found (Eq. 3.7) or, alternatively, a new F_2 node is created. In the latter case, the new category

weight will match the current input vector:

$$\boldsymbol{w}_{N+1} = \boldsymbol{I}. \tag{3.9}$$

The fuzzy ARTMAP architecture (Figure 3.4) is a supervised learning structure built from two fuzzy ART modules, ART$_a$ and ART$_b$, linked by an associative memory called a *map field*, F^{ab}. The ART$_a$ module learns to categorise input patterns, $\boldsymbol{a}$, presented at its input layer, F_0^a, while the ART$_b$ module learns to categorize target patterns, $\boldsymbol{b}$, presented at layer F_0^b. Another weight vector, $w_j^{ab} = (w_{j1}^{ab}, \ldots, w_{jN_b}^{ab})$ connects F_2^a and F^{ab}. A series of 1-to-1 pathways link F_2^b and F^{ab}.

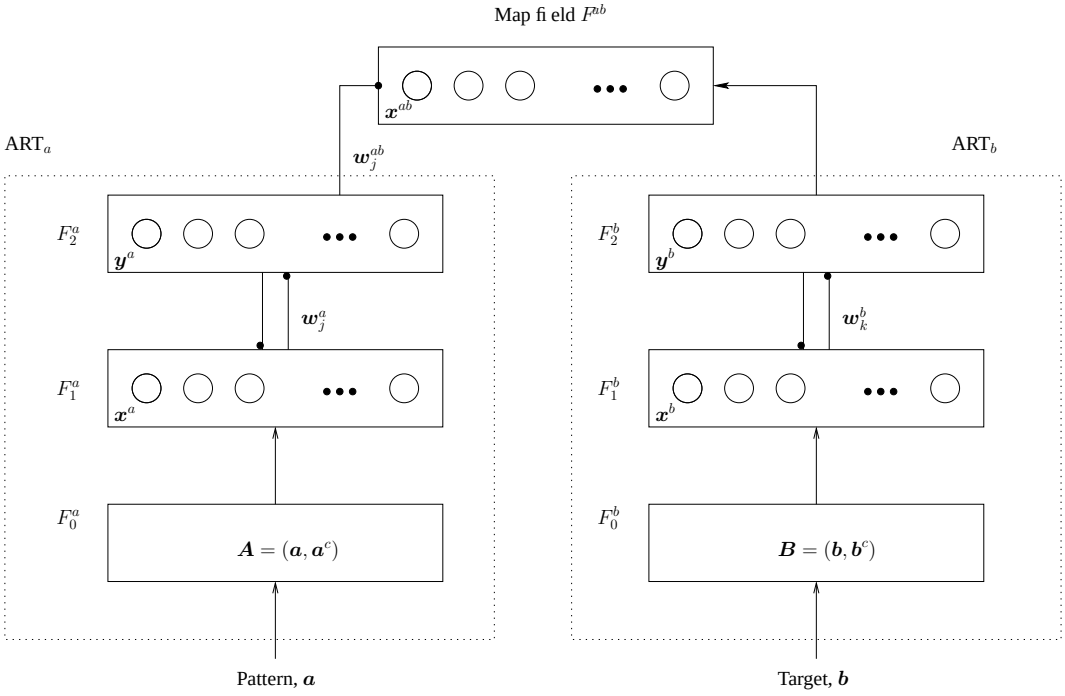


Figure 3.4: The fuzzy ARTMAP supervised learning architecture

To learn a new association between ART$_a$ category J and an ART$_b$ category K, the corresponding map field connection weight ($F_2^a \rightarrow F^{ab}$ link) is set to one, and all other links from the same ART$_a$ module are set to zero. The connection weights, $\boldsymbol{w}_j^{ab}$, therefore hold the mapping

between input and output categories. We can write the update as:

$$w_{jk}^{ab} = \begin{cases} 1 & \text{if } j = J \text{ and } k = K, \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

Before the input and target patterns, $\boldsymbol{a}$ and $\boldsymbol{b}$ are presented to the network, they are first *compliment-coded*, such that for ART$_a$, $I^a = A = (\boldsymbol{a}, \boldsymbol{a}^c)$ and for ART$_b$, $I^b = B = (\boldsymbol{b}, \boldsymbol{b}^c)$, where $a_i^c = 1 - a_i$. Compliment-coding is a normalization rule which preserves amplitude information and prevents category proliferation [14]. The output fields, F_2^a and F_2^b, respond to the patterns at the inputs, F_0^a and F_0^b, and the F^{ab} layer will receive input from both the ART$_a$ module and the ART$_b$ module. The input from ART$_a$ will be from the connection weights $\boldsymbol{w}_J^{ab}$ (through a previously learned $J \rightarrow K$ associative link), and the input from ART$_b$ module will be by the 1-to-1 link to the output vector y^b, where the winning F_2^b node, K, will be set to one.

If the two inputs at F^{ab} match, we say that the network has correctly predicted the target. The FAM then learns by modifying the weight vectors of both the chosen ART$_a$ and ART$_b$ categories (Eq. 3.8) with respect to the patterns present at F_0^a and F_0^b. If the two inputs at the F^{ab} layer do not match, a map field *reset signal* is generated, and we begin a process called *match tracking* which searches for a better category match in ART$_a$. The vigilance of ART$_a$ is always set to a *baseline vigilance*, $\overline{\rho_a} \in [0, 1]$, at the presentation of a new input. At each step of *match tracking*, the vigilance of ART$_a$ is raised by the minimal amount needed to cause a mismatch:

$$\rho_a = \frac{|\boldsymbol{A} \wedge \boldsymbol{w}_J^a|}{|\boldsymbol{A}|} + \iota, \tag{3.11}$$

where ι is defined to be a very small real number. This triggers a search for a new ART$_a$ category, which is then tested again with the current ART$_b$ category at the F^{ab} field for a predictive match. This process continues until we either find an ART$_a$ category that predicts the category of the target through the $\boldsymbol{w}_J^{ab}$ link, or a new F_2^a node is created. In this case, we also create a corresponding link in the map field (Eq. 3.10) to associate the new input category, J, with the target category, K.

Patrascu [57] proposes two changes to the fuzzy ARTMAP to ensure its compatibility with a reinforcement learning framework, and thus we use the same implementation. The first change occurs when learning a pattern. If no mapping has been found and ART$_a$'s vigilance has been

raised above one, the pattern is considered unmappable. This is not compatible with the RL framework, as we must be able to map all input/output pairs. In this special case, the category found just before the vigilance exceeded one is mapped with the currently selected ART_b selected category, instead of considering the pair unmappable. The second change occurs when querying the network. The FAM is not a function approximator; it is a nearest neighbour associator. Thus, when it is queried with a particular input that it has never seen before, it will return an output for the most similar, previously learned input. The incompatibility with RL is that during the retrieval stage of the FAM, if the input is considerably different than any previously seen input, the FAM algorithm will not return an output. In the case of RL, this means that given a state and action input sufficiently different than what has been seen in the training data so far, a corresponding Q-value may not be returned. One option may be to return a default value when this situation occurs. Instead, Patrascu proposes that the vigilance starts high, but if there is no output, then it should be slowly relaxed until an output is given. The FAM will always provide an output, provided the vigilance is sufficiently low. Such a technique is more accurate than returning a default Q-value.

It still remains to be said how the FAM is integrated with the RL algorithm for value-function approximation. This is actually quite simple. Unlike a backpropagation neural network, its training algorithm is not based on a gradient-descent method, and therefore we do not need to change our RL algorithms as in [63]. Storing state-action pairs and their corresponding value estimates becomes nearly as simple as using a look-up table. When we perform the update rule (Consider 2.7 or 2.8) the state-action pair to be updated are provided as input to the FAM, and the new value-function estimate (right hand side of the equation) is provided as output. In the case of the Sarsa(λ) algorithm, where eligibility traces are used, we simply add another output unit to the network, and this is used to generalize over eligibility traces. So a given state-action pair is then associated with both an approximate value and approximate eligibility. The number of input units will depend on the dimensionality of states and actions.

3.5 Continuous actions

One major difference from the previous fuzzy ARTMAP implementation [56] is our use of continuous actions. The neural network replaces the traditional Q-matrix lookup table, but functions in a similar way. Traditional connectionist approaches using discrete actions employ one network for each action [63, 56]. The state is fed as an input to each network and a Q-value is returned corresponding to each action. It is then simple to calculate the action corresponding to the maximum Q-value for any given state. This approach cannot be used for a continuous action space. We have chosen to implement one network per algorithm step (this meaning the image-based algorithm and not the reinforcement learning algorithm!), and thus apply the state and currently considered action to the network, which then returns the corresponding Q-value. Unfortunately, to find the maximum Q-value, we must query the network once for every action, which means an infinite amount of queries. Searching through this infinite set is computationally impossible.

Several possible approaches to this problem have been presented in the literature. It is essentially global, bounded optimization without derivatives, where function calls should be minimized, as they are expensive. Smart and Kaelbling [74] employ an approach similar to Brent's root-finding method [13], which is a simple iterative algorithm. Baird and Klopf [8] have introduced a complex method called "Wire fitting", in which control points (control wires in a high-dimensional space) determining the shape of a complex function are shifted to respond to training data. For their product of experts network, Sallans and Hinton [65] employ Gibbs sampling [42] to choose actions according to their associated expected reward, to be used directly in a Softmax policy. We have chosen to use the DIRECT optimization method [60], which has not yet been used in combination with RL.

A whole different class of RL algorithms, known as actor-critic methods [76, 79] can avoid the need for optimization altogether. They employ a separate memory structure to represent the policy independent of the value function. Beyond the computational speed improvements gained by not having to optimize over the value function, stochastic policies can also be explicitly learned. This is useful in scenarios such as competitive game-playing [11] and in situations where the problem to be solved is non-Markov [89, 32]. Recent experimental results have shown through analysis of functional magnetic resonance images of humans engaged in choosing ac-

tions that the actor-critic model may reasonably represent neural activity [50].

3.5.1 The DIRECT method of optimization

Any RL method which generates a policy based on on a value-function estimate through generalization must be able to provide an action with highest value with respect to a given state. In the case of an infinite number of actions, as for continuous action spaces, we have seen that this is a difficult optimization problem. In the case of the fuzzy ARTMAP, the objective function gradient is not readily available. Estimating it would be extremely difficult and error-prone. Thus we need to turn to a global sampling algorithm. The DIRECT method of optimization, first introduced by Perttunen et al. [60] is one such algorithm, based on Lipschitzian optimization. The author would like to acknowledge that MATLAB code used for the implementation of the algorithm has been provided by Finkel [23].

The Lipschitz condition is defined as follows:

Definition 3.5.1 (Lipschitz Condition) *A function $f(x)$ satisfies the Lipschitz condition of order α at $x = 0$ if*

$$|f(h) - f(0)| \leq B|h|^{\beta},$$

for all $|h| < \epsilon$, where B and β are independent of h, $\beta > 0$, and α is an upper bound for all β for which a finite B exists. The value α is called the Lipschitz constant.

Knowing a particular function is Lipschitz continuous, that is, it satisfies the Lipschitz condition, can aid us greatly in determining the minimum of that function. If we are currently searching at some point, the Lipschitz condition places some bound on the function value of surrounding points, depending on how far we deviate from that point. This aids in guiding the search. Consider the opposite case, for example, a plane in three-dimensional space which has value zero, except for at one finite point, where there exists the minimum. This point would be impossible to find without sampling all of the (infinite) points, as no information exists to guide us toward the minimum. Lipschitz optimization methods are iterative algorithms which rely on the condition above, as well as the constant α to find a minimum. Unfortunately, there are two major shortcomings of this family of algorithms:

- The algorithms are not designed for high dimensions; and
- The Lipschitz constant can be difficult, if not impossible to determine. Estimates can lead to poor performance.

The second problem is particularly relevant for the many surfaces that aren't Lipshitz continuous throughout their domains. Conversely, the DIRECT algorithm addresses both of these issues. It performs well in high dimensions, and requires no knowledge of the Lipschitz constant, nor requires the objective function to be Lipschitz continuous [23].

The DIRECT algorithm begins by transforming the domain of the optimization problem to be solved into the unit hypercube. That is:

$$\overline{\Omega} = \left\{ x \in \mathbb{R}^N : 0 \leq x_i \leq 1 \right\}. \tag{3.12}$$

The centre of this normalized space is c_1, and the first step is to find $f(c_1)$. Next, the hypercube is divided into "hyper-rectangles". This is how the DIRECT algorithm has been named, from *DIviding RECTangles*. This step is performed by evaluating the points $c_1 \pm \delta \boldsymbol{e}_i, i = 1, \ldots, N$, where δ is one-third the length of the sides of the hypercube, and e_i is the ith unit vector. The division is determined by:

$$w_i = \min\left(f(c_1 + \delta e_i), f(c_1 - \delta e_i)\right), 1 \leq i \leq N, \tag{3.13}$$

where we consider each dimension, i, such that $1 \leq i \leq N$, individually. We then split the dimension with smallest w_i into thirds, so that $c_1 \pm \delta e_i$ become the centres of the new hyper-rectangles. The dimension with second-smallest w_i is then split the same way, and we continue accordingly, until all dimensions have been considered.

Figure 3.5 demonstrates this process in two dimensions. First we find $f(c_1) = 7$. Then we find for the horizontal dimension, $f(c_1 + \delta e_1) = 9$ and $f(c_1 - \delta e_1) = 3$. Then for the vertical dimension, $f(c_1 + \delta e_1) = 8$ and $f(c_1 - \delta e_1) = 2$. Therefore, $w_1 = \min(9, 3) = 3$ and $w_2 = \min(8, 2) = 2$. So we divide the vertical dimension first. This creates three regions, with 8,7 and 2 the function values at the respective centres. We again consider just the rectangle which contains $f(c_1) = 7$. We have only one more dimension (horizontal) to consider, so it is split. We are left with 5 regions.

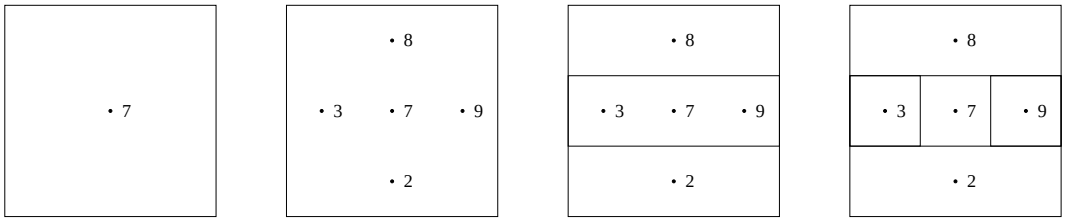


Figure 3.5: Initial division of rectangles in DIRECT

Now we begin the main loop of the DIRECT algorithm, that is, the identification of "potentially optimal" hyper-rectangles which are then divided. DIRECT uses the following lemma to identify potentially optimal rectangles:

Lemma 3.5.1 (Potentially Optimal Rectangles) *Let $\epsilon > 0$ be a positive constant, and let f_{min} be the current best known function value. Let c_j be the centre of hyper-rectangle j and let d_j be some measure for this hyperrectangle (for example, the distance from c_j to its vertices). Let I be the set of indices of all existing rectangles. For a given rectangle, j, let*

$$\begin{aligned} I_1 &= \{i \in I : d_i < d_j\}, \\ I_2 &= \{i \in I : d_i > d_j\}, \\ I_3 &= \{i \in I : d_i = d_j\}. \end{aligned}$$

Rectangle $j \in I$ is potentially optimal if

$$f(c_j) \leq f(c_i), \forall i \in I_3, \tag{3.14}$$

there exists some $\hat{K} > 0$ such that

$$\max_{i \in I_1} \frac{f(c_j) - f(c_i)}{d_j - d_i} \leq \hat{K} \leq \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{d_i - d_j}, \tag{3.15}$$

and

$$\epsilon \leq \frac{f_{min} - f(c_j)}{|f_{min}|} + \frac{d_j}{|f_{min}|} \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{d_i - d_j}, f_{min} \neq 0, \tag{3.16}$$

or

$$f(c_j) \leq d_j \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{d_i - d_j}, f_{min} = 0. \tag{3.17}$$

We note that more than one potentially optimal hyper-rectangle may be found. Once one or more potentially optimal hyper-rectangles have been identified, they are then divided, similar to the initialization process previously discussed. However, this time, we consider only dimensions of maximum length. In the case of hyper-cubes, we divide along all dimensions. We define:

$$w_j = \min \left(f(c_i + \delta_i e_j), f(c_i - \delta_i e_j) \right), j \in K, \tag{3.18}$$

where K is the set of all dimensions of maximal length for hyper-rectangle i. Again, divisions are done in order of increasing w_j, where the dimension with smallest w_j is divided first. Let $\hat{j} = \arg\min_{j \in I} w_j$. The hyper-rectangles are split into three hyper-rectangles along this dimension, so that $c_i \pm \delta i e_{\hat{j}}$ and c_i are the centres of these new rectangles. We then proceed to the dimension with second smallest w_j and so on until we have divided along all dimensions in K. Algorithm 1 summarizes the DIRECT algorithm.

Figure 3.6 shows two iterations of DIRECT, starting from an initialized hyper-cube (or a square in two dimensions, as in Figure 3.5). In the first iteration, the bottommost rectangle is found to be potentially optimal (note that it is greyed). It has only one dimension of greatest length (horizontal), so it is split along this dimension. In the second iteration, two rectangles are found to be potentially optimal. They are both divided, but the square is divided along both dimensions, as its sides are of equal length.

Algorithm 1 Summary of the DIRECT algorithm

Inputs: f is the function to be optimized; a represents the lower-bound vector; b represents the upper-bound vector
Parameters: κ represents the maximum number of iterations; φ represents the maximum number of calls to f, ϵ is some small constant, $1 \times 10^{-7} \leq \epsilon \leq 1 \times 10^{-2}$
Normalize the domain to be the unit hypercube with centre c_1
Find $f(c_1)$, $f_{min} \leftarrow f(c_1)$,$i \leftarrow 0$,$m \leftarrow 1$
Divide hyper-cube (as in initilization step), considering all dimensions
while $i \leq \kappa$ and $m \leq \varphi$ **do**
  Identify the set, S of all potentially optimal hyper-rectangles
  for all $j \in S$ **do**
    Identify the set, K of all dimensions of longest length of rectangle j
    Order K by increasing w_k
    for all $k \in K$ **do**
      Divide j into smaller rectangles along dimension k
      Find new centres, $c_1^{new} = c_j + \delta_j e_k$,$c_2^{new} = c_j - \delta_j e_k$
      Find value of f at the new centres, $f(c_1^{new}), f(c_2^{new})$ and update $m \leftarrow m + 2$
      Update $f_{min} \leftarrow \min(f_{min}, f(c_1^{new}), f(c_2^{new})$
    end for
    $i \leftarrow i + 1$
  end for
end while

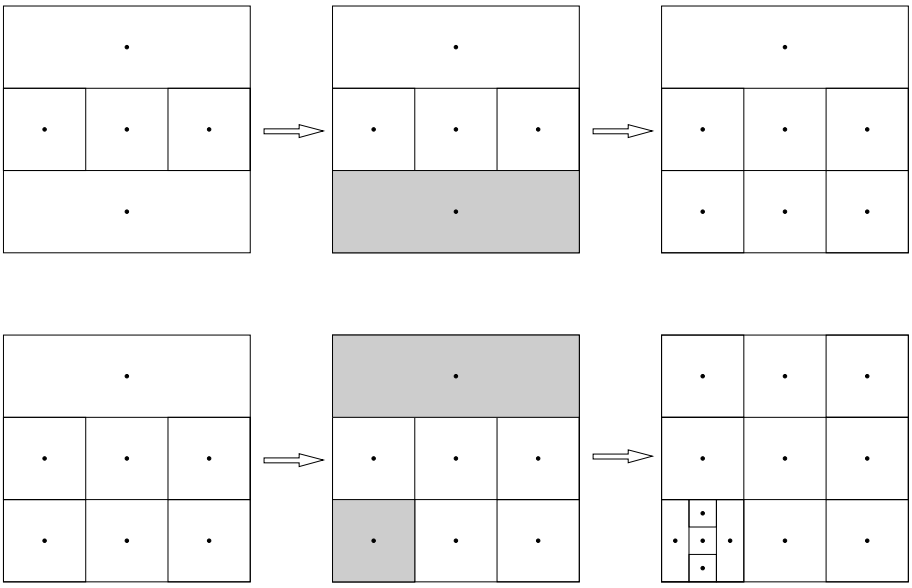


Figure 3.6: Two iterations of DIRECT. The identified potentially optimal rectangles are marked in grey.

3.6 Summary: A connectionist-based reinforcement learning agent for parameter control

Figure 3.7 visually represents the main contribution of this thesis: a framework of parameter control by a reinforcement learning agent for an image-based task. The dotted lines represent the output, or action taken by the agent. The solid lines represent input to the agent. Optional components, including feature extraction and subjective evaluation have also been included. The following paragraphs provide a description of complete process of parameter control through learning. Algorithm 2 summarizes this text.

The algorithm begins with a single input image, or group of images. Optionally, features are extracted and used for state information. Otherwise, a common initial state is assumed. The first iteration begins and the parameter vector for the first step is selected. The parameter vector is stored at each step, as it contributes to the state at the following step. To select the parameter vector for any given step, the agent employs its policy, which communicates with the fuzzy ARTMAP via the DIRECT algorithm to find the actions with highest Q-value. Occasionally random actions (parameters) are chosen by the agent, according to its stochastic policy, to encourage exploration of the parameter space.

After the first step of the algorithm has executed (with the chosen parameters), an intermediate image (or set of images) will be generated. A reward may be extracted from these intermediate images, thus providing the agent with additional guidance. The current state is then calculated using the first parameter vector, and optionally, features extracted from the intermediate image(s). After the next state has been calculated, learning occurs, regardless of whether or not intermediate reward is used. The RL agent employs bootstrapping to estimate expected reward based on the Q-values at the new state. As previously stated, rewards may only be provided at the end of the algorithm (in the terminal state). In this case, bootstrapping ensures that the terminal reward is "backed-up" to previous state-action pairs. Learning is slightly different, depending on whether or not we are using an on-policy algorithm (such as Sarsa) or an off-policy algorithm (such as Q-learning):

On-policy: the next action is selected (via the policy) in order to determine the Q-value used in the update before the updating takes place. When the policy selects a non-random action, it

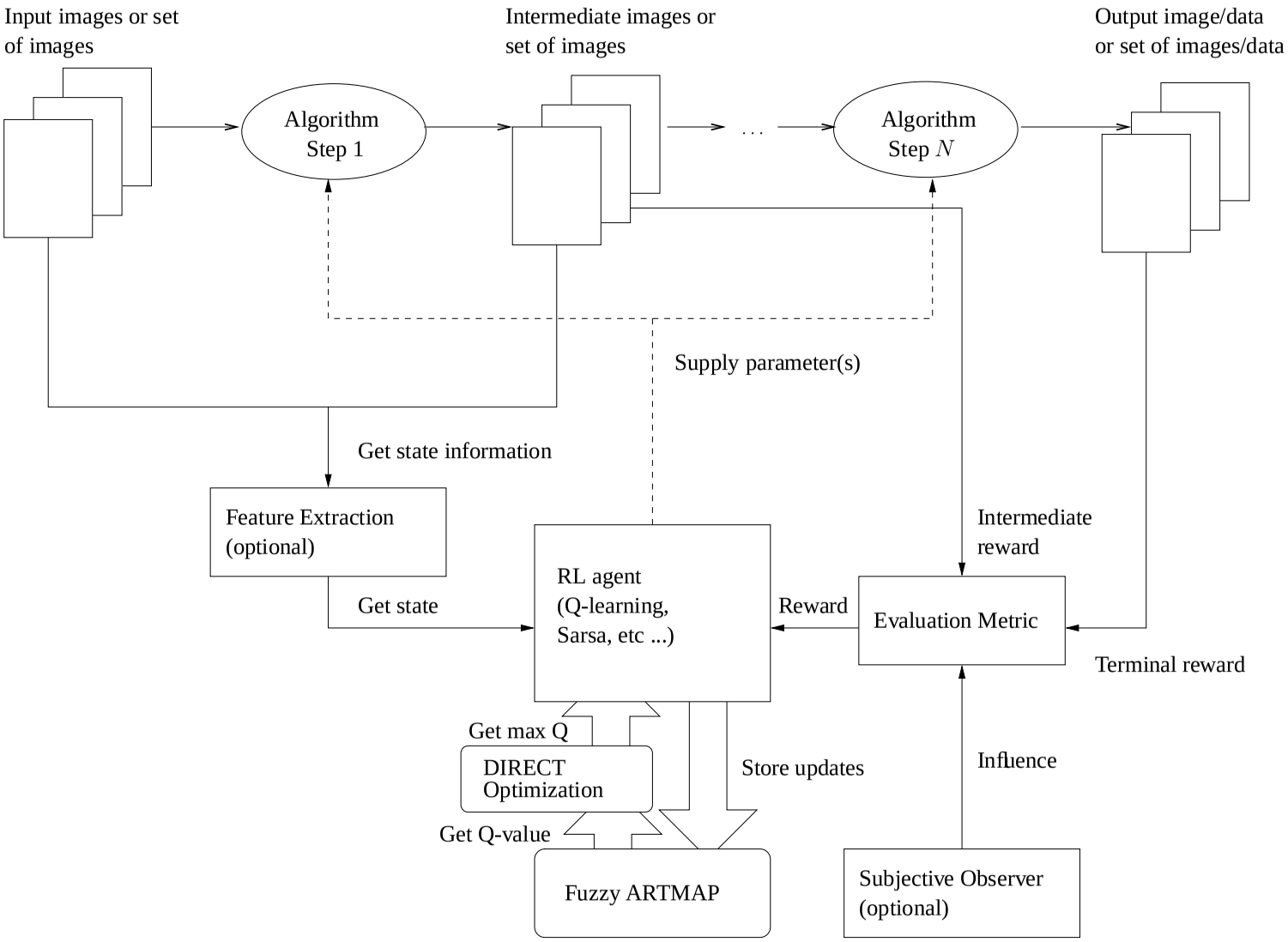


Figure 3.7: A framework for connectionist-based reinforced learning for controlling the parameters of an N-step image-based application

always calls upon the DIRECT algorithm, which, in turn, calls upon the FAM to provide the current action with the greatest expected reward. The new Q-value for the initial state is stored in the FAM.

Off-policy: the update is performed immediately after the next state has been calculated, and then the next action is selected via the policy. This is because we do not need to know the next policy-selected action to perform the update. In fact, in the the off-line tabular case, this order (update vs. action selection) should not matter, as only the Q-value for the previous state is updated. But when generalization is employed, if the two successive states are sufficiently similar, then the update to the old state may affect the Q-value of the new state.

Now that the second parameter vector has been chosen, the next step of the algorithm may execute and we can calculate the new state. Again, learning takes place. This process continues until all parameters have been chosen, and the last step of the algorithm has executed. The result of the algorithm is evaluated, and this determines terminal reward. The final state is common, and signifies that the current iteration of the algorithm has ended. The Q-value for the state reached before the terminal state with respect to the final parameter choice is updated using the terminal reward, and stored in the FAM.

The framework has now been presented, but in a way removed from any particular application. Application-specific design issues, such as reward definition and feature selection still need to be explored. In addition, the framework so far remains untested. The remainder of this thesis will focus on its application within two different image-based domains and address all of these issues in turn.

Algorithm 2 Image-based task parameter optimization using reinforcement learning

```
Initialize one FAM network for each step in algorithm requiring parameter optimization
for all input images, or sets of images do
    Read input image(s)
    Calculate s using input image features, or alternatively, set state to "initial"
    Find parameter vector with maximum Q-value using DIRECT
    Choose parameter(s) for step 1 using policy
    repeat
        Execute step using chosen parameter(s) and obtain output
        Calculate next state, s′ from previous parameters and optionally, features of output
        Employ evaluation metric, observe reward, r; if no intermediate reward, r = 0
        if RL algorithm is on-line then
            Find parameter vector with maximum Q-value using DIRECT
            Choose parameter(s) for next step using policy
            Update value function (stored in FAM) through RL update equation
        else
            Update value function (stored in FAM) through RL update equation
            Find parameter vector with maximum Q-value using DIRECT
            Choose parameter(s) for next step using policy
        end if
    until s′ is terminal (final step of task)
end for
```

Chapter 4

Applying the Framework

In the previous chapter, we defined a framework in which a large amount of problems using visual information can be parameter-controlled by reinforcement learning agents. Now we focus on two separate areas where the framework discussed in the previous chapter may be applied. We first examine the problem of choosing a small number of discrete parameters to form a fuzzy membership function to describe the "brightness" of grey levels of an image. Next, we move onto a problem where generalization is necessary: selecting several continuous parameters for an algorithm that detects text in images extracted from video.

4.1 Automatic determination of membership functions for fuzzy image processing filters

Fuzzy filters are increasingly being developed and used for the field of image processing. A number of state-of-the-art filters have been recently published in the literature [59, 43, 16, 9, 33, 85] which can aid in the reduction of noise in images, evaluation of objects in images, edge detection and segmentation, image enhancement, and colour image processing, to name only a few applications. The majority of filters rely on the definition of fuzzy membership functions (MF) which describe certain characteristics of the image. For example, a membership function related to the histogram of the image may identify certain grey levels of the image as bright or

dark. What is bright, however, in one image (e.g. an image with a histogram encompassing only high grey levels), may not be so bright in another image (e.g. an image with a histogram encompassing only dark grey levels). Furthermore, the evaluation of these membership functions is subjective and will differ between observers. The problem of automatically determining a membership function based on data (i.e. a digital image) has kindled research interest since the advent of fuzzy techniques [67, 58]. Several methods from various points of view have been proposed, from data-driven approaches, such as clustering [38] and genetic algorithms [35, 4] through to ways of building MF directly from subjective interpretation [34]. Newer methods are striking more of a balance between user-interaction and data validation [58].

4.1.1 Evaluation of membership function optimality

To automatically determine any membership function, we must have a method of measuring the optimality of the MF, given the parameters which determine its shape and position. Shannon's entropy measure [21], often considered the fundamental definition in information theory, can assess the information content described by a particular membership function. This is based on the histogram of the image's grey levels to which this membership function is attributed. We note that this is different than measuring the entropy of a particular image, as the relevance of the entropy measure to human visual perception is not undisputed. Pal and Pal [54, 55, 53] have presented other objective measurements such as fuzzy entropy, indices of fuzziness, compactness of a fuzzy subset, and the index of coverage of a fuzzy subset which are all valid alternatives in evaluating a membership function. The process could also be semi-automated in that it could involve the participation of humans who would provide subjective evaluation of generated MF. Again, we would experience the same problems involving humans as in image evaluation. One may also employ combinations of the above measures, even going as far as to fuse metrics using an intelligent agent approach.

4.1.2 A previous experiment using simulated annealing

Cheng and Chen [15] have shown that a membership function for "brightness" can automatically be determined through maximum entropy. They have restricted themselves to a standard "S-type"

membership function, determined by three parameters, and evaluate its optimality based on the maximum entropy principle. However, for even an 8-bit greyscale image, this leaves hundreds of thousands of possibilities to evaluate, in terms of membership function parameter configurations. Instead of performing an exhaustive search, they have employed simulated annealing to tackle what is essentially a combinatorial optimization problem. They demonstrate that this approach can arrive at a near-optimal solution in approximately 3,000 iterations of the simulated annealing algorithm.

4.1.3 Theory

To maintain consistency, we will adopt the notation used by Cheng and Chen [15] as we discuss the problem of automatically determining a brightness membership function.

First, we must examine the concept of a fuzzy event, defined by Zadeh [95] as follows.

Definition 1 *Let (R^n, F, P) be a probability space in which F is the σ-field of Borel sets in R^n and P is a probability measure over R^n. Then, a* fuzzy event *in R^n is a fuzzy set A in R^n, whose membership function, $\mu_A(\mu_A : R^n \rightarrow [0,1])$, is Borel-measurable. The* probability *of a fuzzy event A is defined by the Lebesgue-Stieltjes integral:*

$$P(A) = \int_{R^n} \mu_A(x) dP. \tag{4.1}$$

We will consider a greyscale image of L grey levels, ranging from the darkest, g_0 to the lightest, g_{L-1} which has the histogram of $h(g_k)$, where $k = 0, ..., L-1$. It can be modeled by a triplet (Ω, F, P). Ω is the set of greylevels, $\Omega = \{g_0, g_1, ...g_{L-1}\}$, P is the probability measure of the occurrence of greylevels, $P(g_k) = \frac{h(g_k)}{N}$ (where N is the total number of pixels in the image), and F is a σ-field of fuzzy subsets of Ω.

In the context of our problem, the brightness of the grey levels in an image is a "fuzzy" concept. Instead of grouping some grey levels into the category "bright" and others into the category "not-bright" (the classical approach), we allow each grey level to possess membership in the fuzzy set "bright".

Fuzzy set theory allows us to express the degree of belongingness of an element to the sample space of a particular fuzzy set through a membership function. In this case, we describe the

"brightness" of a particular greylevel, r_k with the membership function $\mu_{bright}(g_k)$. Thus the "brightness of greylevels" is a fuzzy set of Ω. By Zadeh's definition of a fuzzy event, a fuzzy event is a fuzzy set in Ω. The fuzzy set, "brightness of greylevels" is then a fuzzy event in Ω. We can write this event in fuzzy set notation as

$$\text{bright} = \mu_{\text{bright}}(g_0)/g_0 + \mu_{\text{bright}}(g_1)/g_1 + \ldots + \mu_{\text{bright}}(g_{L-1})/g_{L-1}, \tag{4.2}$$

or simply

$$\text{bright} = \sum_{g_k \in \Omega} \mu_{\text{bright}}(g_k)/g_k. \tag{4.3}$$

The probability of this fuzzy event can be calculated using Eq. 4.1. In the discrete case, it is

$$P(\text{bright}) = \sum_{k=0}^{L-1} \mu_{\text{bright}}(g_k) P(g_k). \tag{4.4}$$

4.1.4 Entropy

Entropy is the fundamental concept of information theory. It describes, mathematically, the amount of uncertainty and/or the amount of information in a data set. We would like our fuzzy event to possess a high entropy, and thus high amount of information. We measure this through the *entropy for occurrence of a fuzzy event A*, which is defined as [55]:

$$H(A) = -P(A)\log(P(A)) - (1 - P(A))\log(1 - P(A)), \tag{4.5}$$

and therefore the entropy of the fuzzy event "bright" follows as:

$$H(\text{bright}) = -P(\text{bright})\log(P(\text{bright})) - (1 - P(\text{bright}))\log(1 - P(\text{bright})). \tag{4.6}$$

To summarize, we can calculate the entropy of a fuzzy event, "brightness of greylevels" through a series of steps:

- Construct the membership function. This provides each grey level with an associated fuzzy membership in the fuzzy event "brightness of greylevels".
- Calculate the probability of the occurrence of greylevels, $P(g_k)$.
- Calculate the probability of the fuzzy event, $P(\text{bright})$ (Eq. 4.4).
- Calculate the corresponding entropy of the fuzzy event, $H(\text{bright})$ (Eq. 4.5).

This measure is one way of evaluating the goodness of a particular membership function.

4.1.5 Choosing a membership function

The types of membership functions that can be used to describe image brightness are numerous. However, choosing a membership function with relatively few parameters will be easier to tune, since it reduces the state space (as described in Section 2.2). While the approach described here can be adapted to the tuning of any regular-shaped membership function described by a sequence of discrete parameters, we have chosen to use the simple S-shaped membership function, so that we may compare our results to the previous work [15]. The parameters control both the positioning and shape of the MF. Research has shown, however, that the shape does not have as much as an effect on the fuzzy system as does the positioning [83].

The S-function is defined as follows:

$$S(x;a,b,c) = \begin{cases} 0, & x \leq a, \\ \frac{(x-a)^2}{(b-a)(c-a)}, & a \leq x \leq b, \\ 1 - \frac{(x-c)^2}{(c-b)(c-a)}, & b \leq x \leq c, \\ 1, & x \geq c, \end{cases} \tag{4.7}$$

where x is the independent parameter (in our case, a grey level in Ω), and parameters a,b, and c determine the shape of the membership function. Parameter b can take any value in the range $a \leq b \leq c$.

4.1.6 An optimization problem

For a given membership function, $S(x; a, b, c)$ and image, I, we can define a corresponding entropy measure, $H(\text{bright}; a, b, c)$ for a fuzzy event "bright". This value is dependent on both the image and the parameters which determine the membership function. The entropy of the fuzzy set is then used to automatically determine the best set of parameters defining the brightness membership function for a given image. Essentially, we want to find an optimal set of parameters, $(a_{opt}, b_{opt}, c_{opt})$ such that

$$H_{max}(\text{bright}; a_{opt}, b_{opt}, c_{opt}) = \max\{H(\text{bright}; a, b, c | r_0 \leq a < b < c \leq r_{L-1})\}. \quad (4.8)$$

This is not an easy task, considering all of the possible combinations of parameters a,b, and c. An exhaustive search of the parameter space is not practical, so we must use a more intelligent approach. The problem was originally solved using the simulated annealing algorithm [36]. Despite the success obtained in determining near-optimal membership functions, this approach may be hampered by the local minimum problem [15]. Of course, this is dependent on the initial temperature and cooling function applied. This issue becomes increasingly more important in the more general parameter selection case, when we move away from the smooth, regular nature of membership functions. Many alternative methods of solving the combinatorial optimization problem do exist, for example dynamic programming, genetic algorithms and reinforcement learning. If we adopt RL, then the problem fits neatly into our framework described in Chapter 3. We have a three-step task where the general image-based task parameters become the parameters a, b and c, chosen one at a time, and our reward function becomes the entropy of a fuzzy event. State and action formation are already defined by the framework.

4.2 Parameter control for a text-detection algorithm

Wolf et al. have developed an algorithm to detect text in images extracted from video sequences [92]. Video is processed on a frame by frame basis, tracking detected rectangles of text across multiple frames. Our learning focuses only on the detection of text in still images. This text de-

tection algorithm is multi-step, involving gray level, morphological and geometrical constraints. This results in a continuous and high-dimensional set of parameters to optimize.

4.2.1 Motivation

Text detection in documents has been an active research area for many years [91]. Already, there exist many commercial software packages which can accomplish the task effectively and efficiently. More recently, researchers have been concerned with the detection of text in multimedia, such as video sequences. The nature of the media does not allow us to just simply apply the methods developed by the document analysis community. It is necessary to recognize the differences between printed data and video, and from this standpoint, develop techniques for multimedia text detection by exploiting the properties of text.

Large databases of video have been and are currently being developed for archival purposes. In order to access the individual documents, navigation must be straightforward for the end user. It is expected that the user will browse by low-level and/or semantic features. Detecting text is only one way of making this browsing possible. Other means include speech recognition and face detection. Text, however, possesses “immediate” meaning, thus providing information without a sophisticated interpretation process. Although its interpretation is still subjective, it is less ambiguous than, say, low-level image features. Therefore, we rely on text to not only annotate particular videos, but scenes and segments within the videos themselves.

Historically, the field has relied on individuals to prepare “documentary notes”, those being the subjective interpretation of multimedia content; and most importantly among these, a summary of the document. As video possesses a time-element, the construction of documentary notes is time and resource intensive. If the two are compared, one can see similarities between text contained on the screen and text within the notes [91]. This indicates that we may exploit screen text for the description process. An example of this would be the case of television news broadcasts, where often on-screen text indicates the beginning of a story or interview. From this example, we can see that the detection of text in video can be useful not only for description, but also for segmentation purposes. For videos containing a well-defined structure, such as news broadcasts, individual segments may be both opened and closed by on-screen text (the latter often being the names of the journalists or production team involved in that particular sequence).

Types of text in video

In the computer-vision literature, generally we define two classes of text in videos [91]:

Artificial text is superimposed on the signal after the video has been captured by camera. As it is designed to be read, often it is highly contrasted against the background. An example of this may be the score in a sports game.

Scene text is part of the scene captured by camera. As it is not designed to be read, often it may be of low contrast, very small, partly occluded, or distorted. An example of this may be the writing on a building in the background of a scene.

In addition to the above two classes, we may also separate a third:

Targeted scene text is a sub-class of scene text, as it is part of the scene. However, this text is specifically targeted in the video to attract the viewer's attention (i.e. by zooming into it). Usually it is visible, but as in classical scene text, distortions may be present.

While Wolf et al.'s system is not limited to any particular type, nor orientation of text, we have restricted our experiments to images containing only horizontal, artificial text.

Difficulties in detecting text in multimedia documents

As previously mentioned, classical document analysis techniques for detecting text in printed documents cannot be applied directly to multimedia documents. Table 4.1 lists some of the differences between document images and video sequences. It also indicates that text detection in video is much more difficult than text detection in printed documents. There are two conclusions to be drawn from the comparison of document images and video sequences:

1. Text detection in video sequences is problematic; and

2. Document recognition software cannot be applied to video without significant adaptation.

Table 4.1: Comparison of document images and video sequences

Document images	Video sequences
Sole purpose of document images is their usage as input to document recognition software	Main goal is to be displayed with sufficient quality
Stored as uncompressed or compressed without loss	Often massively compressed using lossy techniques (e.g. MPEG,DIVX)
High resolution	Low resolution
Contents simple and well-separated	Complex contents, text superimposed or even part of the image
Simple background	Complex background
Mostly binary	Contain rich grey-scale or colour information
Text is main information of image	Text is not main information of image

4.2.2 The algorithm

Wolf et al.'s algorithm has been developed to overcome the previously discussed difficulties in recognizing text within video sequences. In order to discriminate between many types of text and non-text, an algorithm must be based on the recognition of characters. Unfortunately, text must be detected before it is recognized. This is a limitation of current OCR systems. If we are interested then in detecting regions which contain text, we must ask ourselves, exactly what constitutes text? Table 4.2 summarizes some of the important components of text, which are discussed more thoroughly in [91]. These lead us to the features that are exploited for text detection.

Text detection in video sequences and text detection in still images go hand in hand. In fact, the detection of text in still images (extracted from video sequences) is an important step in the video text extraction process. The output of still-image text detection is a set of rectangular regions of the image, representing areas believed to contain text. We note that the decision made is binary- the rectangular regions do not differ in confidence. After the detection process is applied to each frame of a sequence, the temporal aspect of video is then considered. The detected text is tracked through frames, identifying corresponding rectangles representing the

Table 4.2: What is text?

Spatial grouping	Text consists of letters or symbols which can be segmented. Often the poor quality of videos complicates this process.
Texture	This is easier to exploit in noisy and low resolution documents. Characters and symbols are composed of strokes. In western text, they are for the most part vertical. We can identify the pseudo-regular patterns of text using texture classification methods such as second order statistics or Gabor filtering.
Contrast and geometry	Artificial text is often high contrast. Artificial text strings, i.e. words or sentences, form rectangles.
Colour	For the detection of artificial text, we do not need colour information. For scene text, often its colour is very different from the background, so we may use colour gradients. Usually this is reserved for rare cases.

same text appearance in successive frames. These frames are then integrated to form a single enhanced image, which is then binarized. This output is then passed to standard commercial OCR software.

In his PhD thesis [91], Wolf proposes two methods for text detection in still images. He describes this as the "heart" of the extraction system. The first method assumes the presence of text in the image, and attempts to find a threshold between text and non-text pixels by statistical modelling of the two distributions. Post-processing (morphological and geometrical constraints) then correct the errors made by the original erroneous hypothesis: not all video frames actually do contain text. The second method employs Support Vector Machines [48] (SVM) to learn text features from training data. Here, geometrical features are included directly in the detection phase, rather than in post-processing. We focus on parameter control for the first and less adaptable method, that of detection of text in still images by local contrast, with the intent of improving performance.

While it has been necessary to choose a complex algorithm to demonstrate the robustness of our framework, it is easy to become engaged in the details of the algorithm and lose track of our focus: reinforcement learning for parameter control. This is why we have provided a summary

in Algorithm 3 but discuss its details in Appendix A. The algorithm is also represented visually in Figure 4.1. Knowledge of the algorithm is necessary to understand components such as the reward (Section 4.2.4) and image features (Section 4.2.5) specific to our task. Therefore we hope the reader will consult the details provided in the appendix.

Algorithm 3 Summary of Wolf et al.'s text detection algorithm. Parameters are given in braces.

Greylevel constraints:
Apply horizontal version of the Sobel operator as a gradient measure of the input
Detect the text by accumulating gradients $\{S$, size of the gradient accumulation filter$\}$
Apply a two-threshold version of Otsu's global thresholding algorithm $\{\alpha$, second threshold for the binarization of the accumulated gradients$\}$
Morphological constraints:
Apply morphological closing to close small holes in the components and connect components separated by small gaps
Remove small horizontal bridges between connected components by a thresholding operation $\{t_1$, threshold on column height$\}$
Perform a conditional dilation to re-connect noisy text regions separated by the previous step $\{t_2$, threshold on height difference; t_3, threshold on position difference$\}$
Perform a conditional erosion to restore the size of components
Perform horizontal erosion to eliminate components that do not satisfy the hypothesis that text has a minimum length
Perform horizontal dilation to restore components to nearly their original size
Extract remaining components by connected components analysis
Grow bounding boxes to further account for reduced size
Geometrical constraints:
Impose geometrical constraints to eliminate false rectangles $\{t_4$, threshold on ratio width/height; t_5, threshold on ratio pixels/area$\}$
Consider special cases where heads and tails of characters have been eliminated in the morphological operations
Combine partly overlapping rectangles according to their geometry $\{t_6$, t_7, t_8, thresholds for combining rectangles$\}$

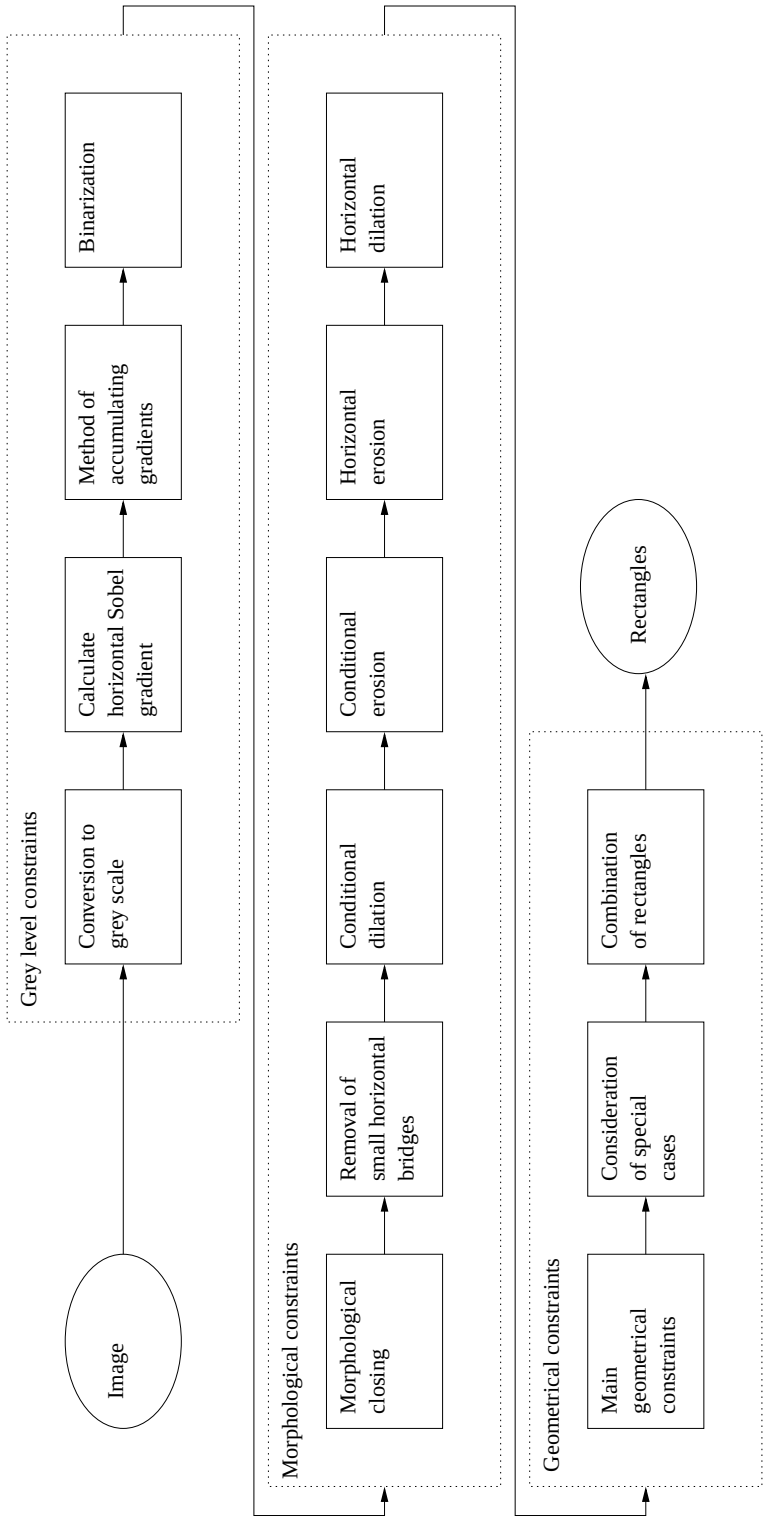


Figure 4.1: The algorithm for detection of text in images from video sequences for semantic indexing

Table 4.3: Parameters for the text detection algorithm in [92]

Par.	Range	Description
S	$5-35$	Size of the gradient accumulation filter
α	$50-150$	Determination of the second threshold for the binarization of the accumulated gradients
t_1	$1-5$	Threshold on column height
t_2	$20-200$	Threshold on height difference
t_3	$20-200$	Threshold on position difference
t_4	$1-6$	Threshold on ratio width/height
t_5	$0.2-0.9$	Threshold on ratio pixels/area
t_6	$0.1-0.9$	Threshold for combining rectangles
t_7	$0.1-0.9$	Threshold for combining rectangles
t_8	$0.1-0.9$	Threshold for combining rectangles

4.2.3 Another optimization problem

From the summary of the text detection algorithm, it is clear that there are a number of parameters than can significantly affect the algorithm's performance. Although Wolf et al. have proposed a set of good parameter values determined empirically, we would like to investigate whether a better set of parameters can be generated using our RL framework. The ten parameters that we aim to optimize and their suggested ranges are shown in Table 4.3.

At each iteration, the agent randomly selects three images containing text, and three images containing no text from an image base. The agent then proceeds, employing parameters chosen by an RL algorithm at each step of the text-detection process. Each of the six test images are processed in parallel, using identical parameters. The rewards are averaged over the images. Our reasoning for the use of multiple images and averaging of results is to assist the agent in

generalizing across many different images. This lowers the sensitivity of the reward to any particular image. The calculation of this reward is discussed in the next section.

4.2.4 Evaluation and reward

As indicated in Section 4.2.2 Wolf et al.'s algorithm incorrectly assumes the presence of text in an image, and then identifies pixels as text or non-text by thresholding. The correcting steps of morphological and geometrical constraints accommodate for the original, faulty hypothesis. It would be simple to evaluate the system in a way that matches the hypothesis (i.e. using only images which contain text), but such an analysis is not sound. We must also evaluate the performance of the algorithm, and in our case, the agent controlling its parameters on images containing both text and no text. The aim of our RL agent is to choose the best parameters for text-detection in still images. In the case where text is present, we wish to reward the agent when text regions are properly detected. In the case of non-text images, we wish to reward the agent when it detects no text regions. We first consider the case of images that we know *a priori* to contain text. Here, we employ the classical method of measuring performance in information systems. That is, the measures of recall and precision [29]:

$$\begin{aligned} \text{Recall}_{IR} &= \frac{\text{number of correctly retrieved items}}{\text{number of relevant items in the database}}, \\ \text{Precision}_{IR} &= \frac{\text{number of correctly retrieved items}}{\text{total number of retrieved items}}. \end{aligned} \tag{4.9}$$

In order to have a single performance value for the ranking of results, we have used the harmonic mean of precision and recall:

$$\text{HM}_{IR} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4.10}$$

In our specific case, we do not consider the recognition of characters, and thus concern ourselves only with the position of text on the screen. Our ground truth images contain marked rectangles, representing the areas which contain text. So we state that the "number of correctly retrieved items" corresponds to the "number of correctly detected rectangles", the "number of relevant items in the database" corresponds to the "number of ground truth rectangles", and the

"total number of retrieved items" corresponds to the "total number of detected rectangles". This still leads us to the question: what does it mean for an object to be "detected"? Of course, some objects may partially be detected, and it is unlikely that the rectangles detected by the algorithm have exactly the same dimensions and position as the ground truth rectangles, even if we do consider the detection to be "good". Fortunately Wolf et al. have already devised several methods which take into account the geometrical information, that is, the overlapping between ground truth and detected rectangles. We have chosen to use one of the methods, CRISP [91], to deal with the ambiguity of "detection" when calculating the reward signal provided to the agent.

The CRISP method

The CRISP method takes into account one-to-one as well as many-to-one and one-to-many matches between detected and ground-truth rectangles. We first create two lists, G and D, representing the ground truth and detected rectangles, respectively, in a particular image. From these lists, two overlap matrices are created, σ and τ. The rows $i = 1..|G|$ of the matrices correspond to the ground truth rectangles, and the columns, $j = 1..|D|$ correspond to the detected rectangles. The matrix elements are calculated as:

$$\sigma_{ij} = \frac{\text{Area}(G_i \cap D_j)}{\text{Area}(G_i)}, \text{and} \tag{4.11}$$

$$\tau_{ij} = \frac{\text{Area}(G_i \cap D_j)}{\text{Area}(D_j)}. \tag{4.12}$$

The values σ_{ij} and τ_{ij} can be thought of as representing measures of recall and precision, respectively, if we consider a detection of ground truth rectangle i by rectangle j. We then analyze the matrix with respect to three types of matches:

- One-to-one matches: G_i matches against D_j if row i of both matrices contains only one non-zero element at column j and column j of both matrices contains only one non-zero element at row i. Two additional conditions are imposed on the match. Those are, $\sigma_{ij} \geq e_1 = 0.8$ and $\tau_{ij} \geq e_2 = 0.4$. This states that the overlap area must have a certain size compared to both the ground truth and detected rectangles to be considered as detected.

- One-to-many matches (one ground truth rectangle): G_i matches against several detected rectangles if row i of both matrices contains only one non-zero element at column j. Again, additional conditions are imposed. We require that $\sum_j \sigma_{ij} \geq e_3 = 0.8$ and $\forall j : \tau_{ij} \geq e_4 = 0.4$. The first threshold, e_3 ensures that the ground truth rectangle is sufficiently detected by the multiple rectangles (i.e. with high recall), and the second threshold, e_4 ensures that each of the detections is sufficiently precise.

- One-to-many matches (one detected rectangle): D_j matches against several ground truth rectangles if column j of both matrices contains only one non-zero element at row i. Similar constraints are enforced, $\sum_i \tau_{ij} \geq e_5 = 0.4$ and $\forall i : \sigma_{ij} \geq e_6 = 0.8$, where threshold e_5 ensures the precision of the detection and threshold e_6 ensures that each of the ground truth rectangles is detected sufficiently (i.e. with high recall).

Finally, one more geometrical constraint is imposed on the one-to-one matches. We require that the differences of the left coordinates of the rectangles must be smaller than an adaptive threshold, that depends on the width of the ground truth rectangle. This accounts for certain situations where overlap information indicates a good match, but the detected rectangle is significantly larger than the ground truth rectangle (i.e. we wouldn't want the detection of the whole screen considered to be a match).

Thresholds were chosen empirically in [91], where one may also find details on performance with respect to the various thresholds. It was found that the thresholds related to the recall, e_1,e_3 and e_6 had little effect on performance, though the thresholds related to precision, e_2,e_4 and e_5 significantly affected the performance negatively as they approached 1. An alternative is to calculate the average recall and precision over the whole range of thresholds [93].

After considering the three types of matches, the CRISP recall and precision values are then calculated as follows:

$$\begin{aligned} \text{Recall}_{CRISP} &= \frac{\sum_i \text{MatchCrisp}_G(G_i)}{|G|}, \\ \text{Precision}_{CRISP} &= \frac{\sum_j \text{MatchCrisp}_D(D_j)}{|D|}, \end{aligned} \tag{4.13}$$

where MatchCrisp$_G$ and MatchCrisp$_D$ are functions which consider the various types of matches previously defined, and evaluate to finite values based on the quality of match:

$$\text{MatchCrisp}_G(G_i) = \begin{cases} 1 & \text{if } G_i \text{ matches against a single detected rectangle,} \\ 0 & \text{if } G_i \text{ does not match against any detected rectangle,} \\ 0.8 & \text{if } G_i \text{ matches against several detected rectangles,} \end{cases}$$

$$\text{MatchCrisp}_D(D_j) = \begin{cases} 1 & \text{if } D_j \text{ matches against a single ground truth rectangle,} \\ 0 & \text{if } D_j \text{ does not match against any ground truth rectangle,} \\ 0.8 & \text{if } D_j \text{ matches against several ground truth rectangles.} \end{cases}$$

As we can see, one-to-many matches are taken into account, but not given as much credit as one-to-one matches. As a result, the value of "number of correctly detected rectangles" does not need to be an integer. In our experiments, we use a set of three text-containing input images for each iteration. The calculated values of recall and precision, using the CRISP method, allow us to then determine, over the set of three images, the total number of ground truth rectangles, the total number of detected rectangles, and the total number of correctly detected rectangles (using Equation 4.9). This allows us to generate precision and recall values between 0 and 1 for the set. Simply using the arithmetic mean to average the individual recall and precision values would not lead to the same result. These overall precision and recall values are combined into a single value using the harmonic mean (Equation 4.10). This represents the first half of the reward, which is with respect to text-containing images.

Evaluation for images not containing text

Evaluation is handled differently for the images containing text and images containing no text. For the former, we have a set of ground truth images to which we compare the agent's selection of rectangular text regions. Unfortunately, the measures of recall and precision are not applicable to the latter. For these experiments, we consider an equal number of images containing text and images containing no text. For non-text, we wish to generate a measure on the same scale, [0,1], which has maximum value when we detect no text regions, and gradually decreases as we detect

more rectangles. To achieve this result, we use the following simple formula for reward:

$$r_{nontext} = e^{-\delta N}, \tag{4.14}$$

where N is the number of detected rectangles (over all test images, in our case 3), and δ is an empirically determined factor. We found that $\delta = 0.5$ offered good performance. As the number of detected rectangles grows, the evaluation is reduced to zero. As in the case of the previous evaluation, the maximum value (at $N = 0$) is 1. Now that we have one performance measure for each of images containing text and images containing no text, we simply combine them by taking their arithmetic mean. If we wanted to train the agent to consider less the non-text images, or vice-versa, this mean could be weighted differently.

Generality

Clearly the text detection performance depends on the database used for testing. This is not only with respect to the image content itself (artificial text vs. scene text, size and font of text, noise, etc ...) but also depends much on the structure of the test database itself. Generally, with any information retrieval task where measures of precision and recall are used, we are concerned with the ratio of relevant data to irrelevant data in the database. Huijsmans and Sebe formalize this notion as the measure of generality [29]:

$$\text{Generality}_{IR} = \frac{\text{number of relevant items in the database}}{\text{number of items in the database}}. \tag{4.15}$$

Low generality will tend to produce results with lower precision. In traditional information retrieval tasks, where the result set is of fixed cardinality, low generality will also affect recall. In text detection, however, generality does not affect recall, as all detected items are returned. This means that increasing the number of non-text containing images will cause the precision to fall, but recall will remain constant, as this will not affect the amount of ground truth rectangles detected, if the number of text containing images is unchanged. For our task, generality is defined as:

$$\text{Generality} = \frac{\text{number of text rectangles in the database}}{\text{number of images in the database}}. \tag{4.16}$$

This leads us to the dilemma on how to weight text and non-text images in the database. We have decided to include 50% images with text and 50% images without text in the database, and on each iteration consider an equal amount of each. This enables us maintain a reasonable level of generality as well as compare our results to the previous work in which this ratio was used. We note that this ratio is not a realistic one if we consider video streams in general, but the addition of too many non-text images into the database negatively affects the detection system [91].

Intermediate reward

So far, we have only discussed a single reward at the end of each episode (terminal reward). In reinforcement learning tasks, often such a reward is sufficient, as the agent aims to maximize cumulative reward and not immediate reward. Intermediate reward, that is, reward following some step other than the final step in an episode may also aid in learning - but we must be careful. The reward serves to formalize the goal of the agent, or in other words, *what* it should achieve. Reward is not the mechanism to communicate to the agent *how* we want it to be achieved [79]. If we confuse the purpose of the reward, the agent may, in fact, learn to achieve high returns yet not meet its final goal. Sutton and Barto [79] note the example of a chess-playing agent that, if rewarded when it captures pieces, may learn to capture a high number of pieces without ever attaining checkmate. Such a case emphasises the need for careful design of the reward signal.

In the text-detection application, we initially evaluate at the agent's performance while using only terminal reinforcement, derived from the information retrieval metrics previously discussed. We also propose two methods of intermediate reward which are simple to calculate and effective measures in theory, but also may fall on the fine line between "what" and "how". It should be noted that these intermediate rewards are made possible by the existence of ground truth images. In a setting where such images were not available (i.e. truly on-line) they could not be applied. These intermediate rewards are extracted at two specific points within the text-detection algorithm (Figure 4.2).

Bhattacharyya distance

This reward is extracted from the images following the accumulated gradients step. Each pixel contains a probability measure that the same pixel in the original image is text.

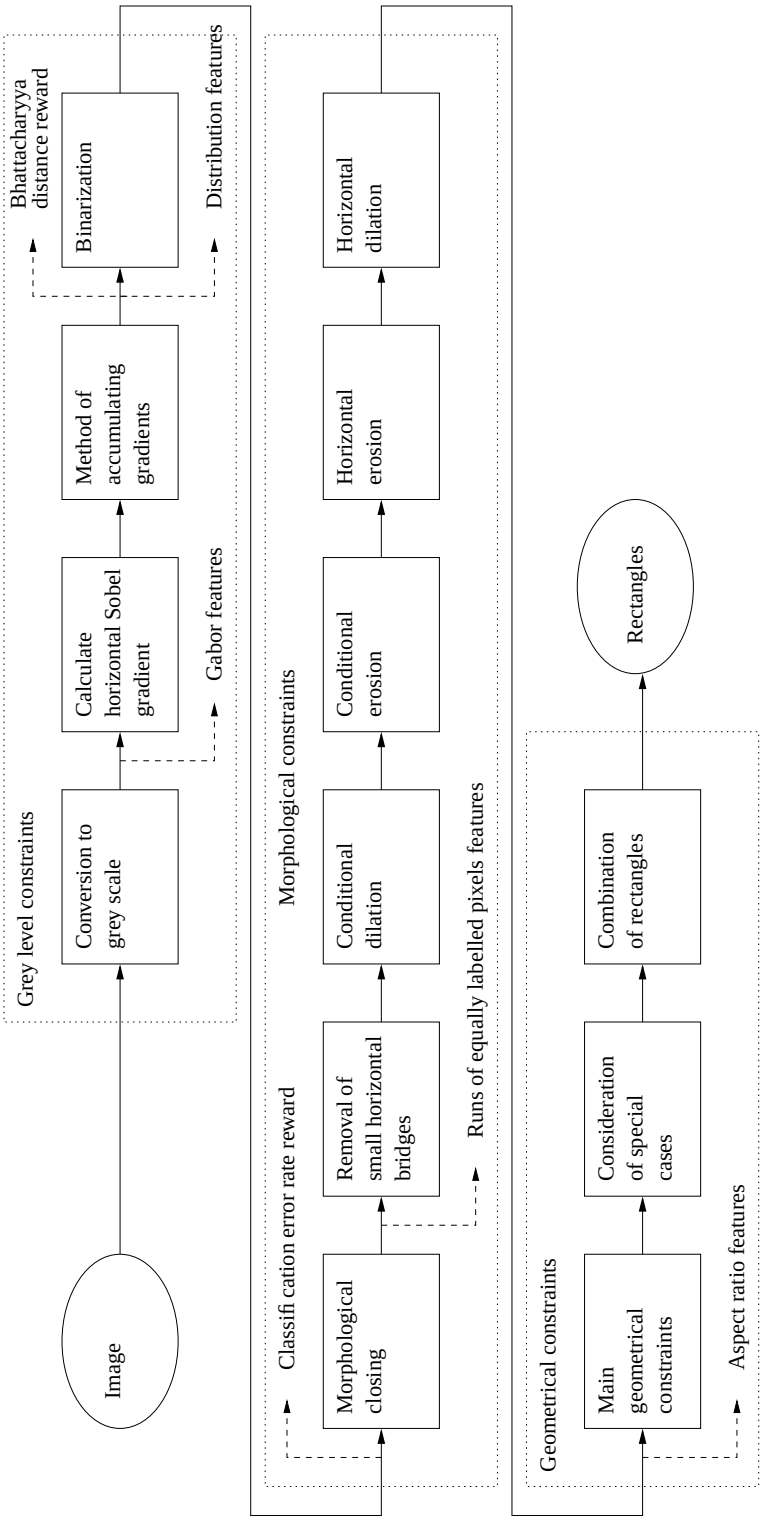


Figure 4.2: The algorithm for detection of text in images from video sequences for semantic indexing. Extraction of intermediate reward (above the steps) and image features (below the steps) are shown in dashed lines.

A popular measure of similarity between two distributions is the Bhattacharyya distance [26]:

$$B = \frac{1}{8}(M_2 - M_1)^T \left(\frac{\Sigma_1 + \Sigma_2}{2}\right)^{-1} (M_2 - M_1) + \frac{1}{2}\ln \frac{|\frac{\Sigma_1+\Sigma_2}{2}|}{\sqrt{|\Sigma_1|}\sqrt{|\Sigma_2|}}, \tag{4.17}$$

where M_i is the mean vector of class i, and Σ_i is the covariance matrix of class i. The first term measures class separability due to the distance between class means, while the second term measures class separability due to the difference between class covariance matrices. Consider Figure 4.3, which is similar to an example given in [66]. We can see that although the distribution pairs marked A and C have the same mean distance, the second term of the Bhattacharyya distance expresses the overlapping of the distributions in C. Therefore, the Bhattacharyya distance for pairs A is greater than for pairs C. Alternatively, while the mean distance is very different for distribution pairs A and B, their Bhattacharyya distance is similar. Again, variance is considered.

If the two distributions are Gaussian, the Bhattacharyya distance gives an upper bound on the Bayes error:

$$\epsilon^* \leq \sqrt{P_1 P_2} e^{-B}, \tag{4.18}$$

where P_i is the *a priori* probability of class i. The Bhattacharyya distance is computationally simple to calculate and because it is derived from an error bound rather than an exact solution, it provides a "smoothed" distance between the two classes [45]. This is useful in our case, where we do not believe that our data follows a true normal distribution.

The Bhattacharyya distance is implemented as an intermediate reward following the accumulating gradients step in the text detection algorithm, but only for images containing text. The pixels of the input image are assigned to one of two classes: "text" and "non-text", based on the matching ground truth image. We have one single variate for each class, and that is the probability of a pixel being text (assigned by the accumulating gradients step, Equation A.1). As the following step is binarization by thresholding, we want these two distributions to be well-separated, and thus have a large Bhattacharyya distance. The arithmetic mean of the Bhattacharyya distance with respect to the three input images is applied directly as intermediate reward following the binarization step.

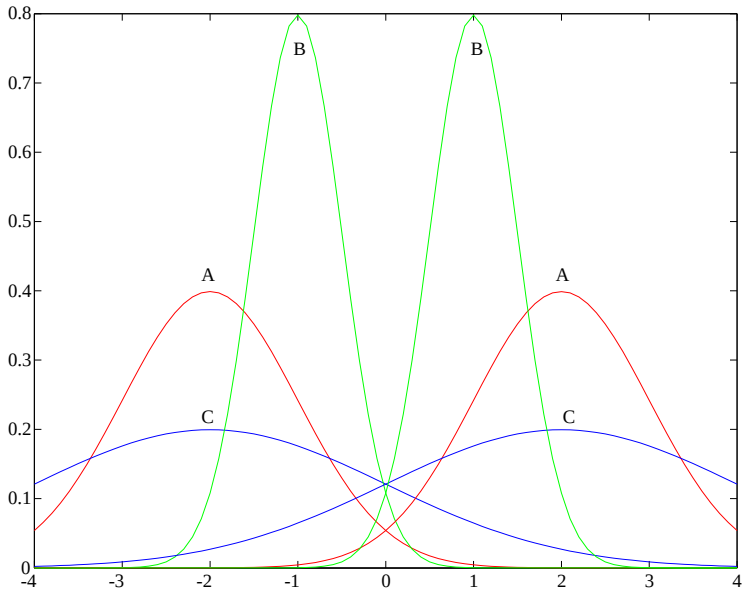


Figure 4.3: Example of Bhattacharyya distance

Classification error rate

This reward is extracted from the images following the binarization step. Each pixel is now marked either "text" or "non-text".

The calculation of classification error rate is even simpler than the Bhattacharyya distance. The aim is to evaluate the performance of the agent before the "clean-up" steps of mathematical morphology and geometrical constraints. Following the binarization step, we can compare the marked pixels to their respective ground truth images with the same labels, "text" and "non-text". The classification error rate can be defined as:

$$\text{Classification error rate} = \frac{\text{total number of correctly classified pixels}}{\text{total number of pixels}}. \tag{4.19}$$

This value is already scaled between 0 and 1 is can be applied directly as reward.

4.2.5 Image features

As already stated, incorporating image features will allow us to specify parameters on a per-image basis, rather than responding to the general set of images on which it has been trained. Several image features have been chosen with respect to the various steps of the text-detection algorithm. These features are extracted from the images produced at these steps (Figure 4.2)). Our aim is toward a system that can choose parameters for the following step, based on properties of the current intermediate image. While associating complete image data with state at each step is far too complex for the current state of learning algorithms, we may extract features that we deem *a priori* to be relevant to the parameter choice at the following step. It is then left to the RL agent to discover the relationship between these features and expected reward.

In each case, we aim to keep the number of features of as small as possible, to maintain the compactness of the neural networks. The features are always scaled to match the magnitude of the normalized parameters which form part of the state information. As these features are application-dependent, their addition requires considerable design and implementation effort, as well as moderate computational expense. Therefore, they have been incorporated only after the system for global parametre selection has been developed and tested.

Gabor filters for texture features

These features are extracted from an input image immediately after it has been converted to grey scale.

During our description of what constitutes text (Table 4.2), we noted that it has a specific texture. Therefore, using features that are descriptive of the texture should aid the agent in selecting the size of the gradient accumulation filter. Gabor filters are a popular method of extracting texture features from images [90]. They are frequently used in the domain of content-based image retrieval. When we describe of texture, we think of image regions rather than pixels or scenes. Therefore, we adopt an approach which calculates texture-features for several windows of fixed-size centered around points of interest in the image. For the Gabor filters themselves, we follow the approach and parameters given by Manjunath and Ma [46].

A two-dimensional Gabor function suitable for images can be written as:

$$g(x,y) = \left(\frac{1}{2\pi\sigma_x\sigma y}\right)\exp\left[-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)+2\pi jWx\right], \tag{4.20}$$

and its Fourier transform can be written as:

$$G(u,v) = \exp\left\{-\frac{1}{2}\left[\frac{(u-W)^2}{\sigma_u^2}+\frac{v^2}{\sigma_v^2}\right]\right\}, \tag{4.21}$$

where $\sigma_u = 1/2\pi\sigma_x$, $\sigma_v = 1/2\pi\sigma_y$ and W is the frequency of interest. A family of related functions, called *G*abor wavelets can also be defined. We create a filter bank based on appropriate dilations and rotations of the *mother function*, $g(x,y)$:

$$\begin{gathered} g_{mn}(x,y) = a^{-m}G(x',y'), a>1; m,n\in\mathbb{Z}, \\ x' = a^{-m}\left(x\cos\theta + y\sin\theta\right), y' = a^{-m}\left(-x\sin\theta + y\cos\theta\right), \end{gathered} \tag{4.22}$$

where $\theta = n\pi/K$ and K is the total number of orientations. We also define S to be the total number of scales in the multiresolution decomposition. Details on selecting the filter parameters a, σ_u, and σ_v to reduce redundancy within the filter bank are given in [46].

Given an image, I, we can define its Gabor wavelet transform to be

$$W_{mn}(x,y) = \int I(x_1,y_1)g_{mn} * (x-x_1, y-y_1)dx_1dy_1, \tag{4.23}$$

where * indicates the complex conjugate. Manjunath and Ma take the mean, u_{mn}, and the standard deviation, σ_{mn}, of the magnitude of the transform coefficients as feature components, assuming that the local texture regions are spatially homogeneous. We, however, use only the maximum of the magnitude of the transform coefficients, as proposed by Wolf [90]. We choose the maximum due to its discriminant properties as the mean tends to smooth the filter response. Though it is less robust than the mean, its performance is illustrated in [90].

This maximum value is calculated for 50 windows of size 32×32, centered around points of interest within the image (accommodating for borders). Using points of interest rather than equally spaced windows (as done in [46]) allows us to obtain Gabor features around points in the image more likely to contain text (if it is present). To calculate points of interest, we use

the popular corner detector proposed by Harris and Stephens [27]. This is repeated for $S = 3$ scales and $K = 2$ orientations, as we are only interested in the horizontal and vertical texture features for text. Aiming to keep the number of features small, and make a decision based upon textures at a global level, we take the mean of the maximum values over all of the windows. This provides us with 6 image features, which compose the initial state of the agent.

Distribution features for accumulated gradients

These features are extracted from the intermediate image immediately after the step of accumulating gradients. For each image pixel, we have a measure representing the probability of that pixel being text.

Let us envision the accumulated gradient values being a sample taken from an underlying distribution. The shape of this distribution is then of interest when choosing the thresholding parameter. If the distribution is bimodal, then we expect these modes to represent non-text and text pixels and the threshold should be chosen appropriately between modes. If it is unimodal, then the image is likely a non-text image, and therefore the threshold should be chosen to minimize the number of pixels binarized to be text.

When describing this distribution, the mean and standard deviation of the accumulated gradients come to mind as natural features. But what can we say about the shape of the distribution? The technique proposed by Silverman [71] is perhaps the most commonly applied method to assess whether a given data set is driven from a $k-$modal distribution. It is based on kernel density estimates, and it is a relatively simple, automated approach. We implemented this method, but found that it was computationally slow, and also required some manual calibration for each new dataset. It is appropriate for the analysis of a finite number of datasets, but not for a stream of images, each containing a high magnitude of data points, as in our application. Therefore we turned to a simple measure of unimodality, where a Gaussian distribution is fitted on the accumulated gradients data, using the mean and standard deviation previously extracted. A histogram with a set number of bins is calculated for the data as well as the Gaussian, and this is used to compare the overlap between the distribution of accumulated gradients and the Gaussian. Specifically, for each bin, we are interested in the difference between the area underneath the accumulated gradients distribution and the area under the Gaussian distribution. We also account for the area under

the Gaussian from $-\infty$ to the lower value of the first bin as well as from the upper value of the final bin to $+\infty$. This technique has the disadvantage that a unimodal distribution is not necessarily shaped like a Gaussian, but it is a simple and computationally fast measure of unimodality suitable for our application, and is complimented by the other distribution-based features.

The distribution can be further described by several features from discriminant analysis. Let us return to Otsu's global thresholding method [51] briefly described in Section 4.2.2. Using the histogram of the accumulated gradients (one bin for each possible value), this provides us with an optimally chosen threshold for the accumulated gradients. Then we calculate the means of the "non-text" and "text" classes, μ_0, μ_1 as well as their variances, σ_0^2, σ_1^2. The intra-class variance is defined as:

$$\sigma_W^2 = \omega_0 \sigma_0^2 + \omega_1 \sigma_1^2, \tag{4.24}$$

where $\omega_0 = Pr(C_0) = \sum_{i=1}^{k^*} p_i$ and $\omega_1 = Pr(C_1) = \sum_{i=k^*}^{L} p_i = 1 - \omega_0$ are the probabilities of class occurrence. Here, $p_i = n_i/N$ is the probability of having accumulated gradient value i, where $i = 1, 2, \ldots, L$; n_i is the number of pixels having accumulated gradient value of i; and N the total number of pixels, $N = n_1 + n_2 + \ldots + n_L$. The threshold k^* is the optimally chosen threshold. The interclass variance is also defined as:

$$\sigma_B^2 = \omega_0 \omega_1 \left(\mu_1 - \mu_0\right)^2. \tag{4.25}$$

These variance measures are calculated to give more information about the shape of the underlying distribution. To summarize, we calculate eight distribution-based features following the accumulating gradients step, and prior to thresholding. They are: μ and σ, the mean and standard deviation of the accumulated gradient values, u, the unimodality measure calculated by comparing the histogram to the histogram of a Gaussian distribution, and the linear discriminant features: $k^*, \mu_0, \mu_1, \sigma_W$, and σ_B. Note that we use standard deviation values rather than variance for the latter two measures. Adding the parameter value chosen in the first step gives us a state vector with nine elements.

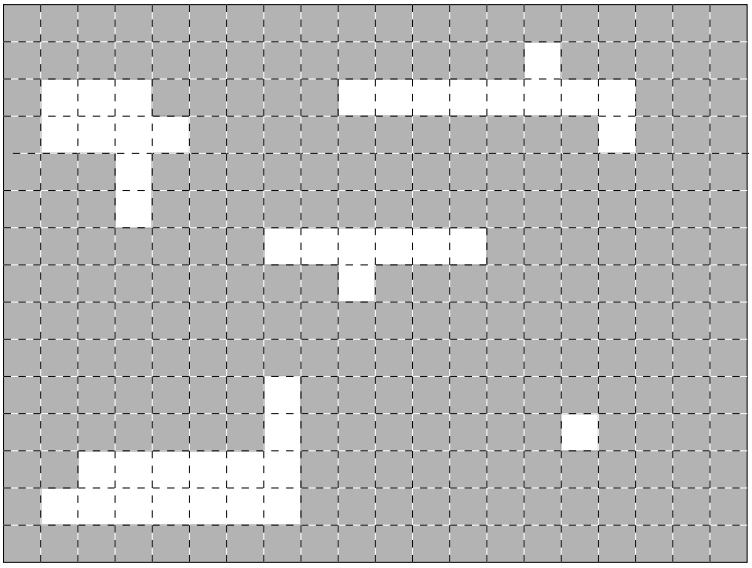
Runs of equally labelled pixels

These features are extracted from the intermediate binary image, just after the morphological closing has been performed. This image indicates pixels believed to be "text" or "non-text".

These particular features are used to construct a state for the decision of the parameter for the erosion operation to remove tiny bridges connecting components. The same features are then extracted again from the eroded image, and another state is constructed for the decision of the "dilation for connecting characters" operation. Statistical measures related to pairs of neighbouring runs should assist in choosing the parameters for the two morphological operations.

This set of features is based on the vertical run lengths of equally labelled pixels. We progress through each column of the binary image, calculating the run lengths of each series of equally labelled pixels (Figure 4.4). We may progress from a "text" run, to a "non-text" run then to another "text" run and so on. Because the image is binary, the runs will always be alternating. We create a 2-D histogram to store the counts with respect to the lengths of all pairs of neighbouring runs. The histogram then has two indices: one with respect to the "text" run length, and one with respect to the "non-text" run length. Immediately after the lengths of each pair are calculated, the element corresponding to that particular length of "text" run and length of "non-text" run is incremented. After we have progressed through all of the columns of the image, we are left with the complete histogram, which gives the frequency of all neighbouring runs of various lengths. We also store the lengths of each encountered "text" run and "non-text" run in another variable, so that the means of the lengths of all "text" and "non-text" runs can be computed.

The final step is to compute the means, as well as the 2×2 covariance matrix using the frequency matrix and the means. Noting that the covariance matrix is symmetrical, we may discard one of the covariance elements to eliminate redundant data, and arrive at a feature vector of five elements. Variances are converted to standard deviation, and the covariance element is scaled by its magnitude. Adding the value of the parameter chosen at the previous step, we form a state vector with six elements.



(a) Calculating the vertical run lengths. Consider the binary images where pixels marked as "text" are white, and the rest are marked "non-text". For each column, we calculate the vertical run lengths of equally labelled pixels. We do not record anything in the first column, because there are no changes. In the second column, there are four changes. We count the pairs (2,2), (2,9), (1,9) and (1,1) where the first index refers to the "text" pixel of the pair and the second index refers to the "non-text" pixel. In the third column, we count (2,2), (2,8), (2,8) and (2,1). We proceed like this through every column.

	1	2	3	4	5	6	7	8	9	10	11	12
1	1	6	10	0	0	2	0	7	1	0	0	1
2	6	3	0	0	0	1	1	3	1	0	1	3
3	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	0	0	1	0	0	0	0	0	0

(b) 2-D histogram of run lengths after all image columns have been processed. Note that the maximum vertical run length for "text" pixels (y-axis) is 4, while the maximum vertical run length for "non-text" pixels (x-axis) is 12.

Figure 4.4: Vertical runs of equally labelled pixels

Aspect ratio features

These features are extracted from the intermediate binary image, following all morphological operations. This image is similar to the one used in the previous set of features, but it is expected to be much cleaner.

Finally, after the morphological processing has finished, we wish to derive features from the aspect ratio of the marked text regions to set the thresholds of the geometrical constraints. At this step, we have a binary image which may contain one or more text regions. For each pixel marked as "text", we consider its neighbouring "text" pixels. Features are derived from the aspect ratios of the horizontal and vertical run lengths of each "text" pixel. This is performed knowing that these aspect ratios may aid in discriminating text from non-text regions. The geometrical constraints exploit this property, but rely on several thresholds to be set. Information regarding the aspect ratios at a global image level should assist in setting these thresholds.

We first record the length of each pixel's vertical and horizontal run. We then disregard the pixels with horizontal or vertical run lengths less than a threshold of five pixels (a value that we have determined empirically). A matrix is initialized with the same size of the image, so that with each "text" pixel meeting the threshold criterion, we associate with it the ratio of width to height (horizontal to vertical lengths, respectively) of its associated runs (Figure 4.5). We note that the value of each element in the matrix corresponding to a "non-text" pixel, or pixel with small associated run lengths will be zero. Non-zero elements are extracted and transformed to the $\log$ domain. A histogram of constant bin-size (in our experiments, we used 50 bins) is built from these ratio values. Then, we use this histogram to calculate the exact same linear discriminant-based features (from Otsu) that we calculated from the accumulated gradients information. The features k^*, μ_0, and μ_1 are calculated in the $\log$ domain and then transformed back to the original domain, while σ_W, and σ_B are calculated in the original domain. These linear discriminant-based features, as well as the overall mean and standard deviation of the $\log$ aspect ratios form the final feature set. The previous two-element parameter for the "dilation for connecting characters" step is added to create a nine-element state vector.

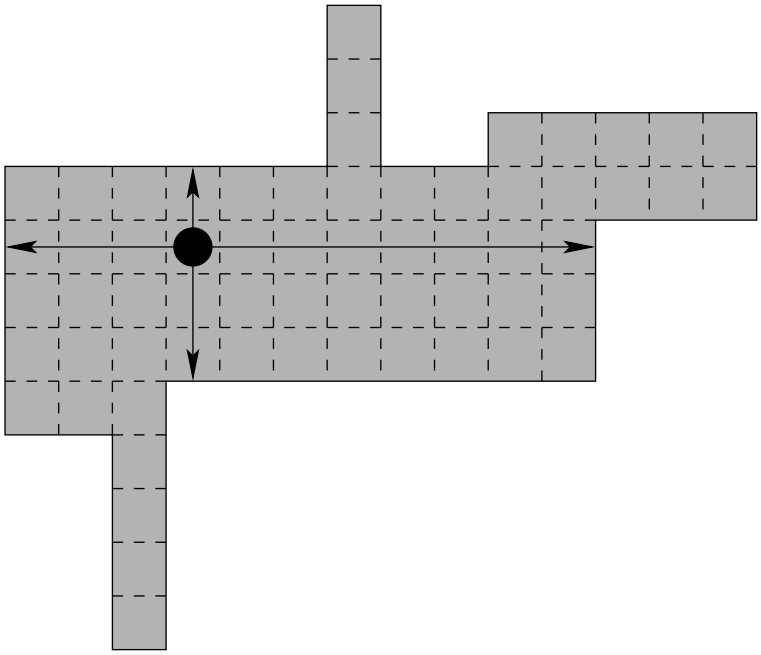


Figure 4.5: Calculating the aspect ratio. Consider a region of pixels marked as “text” in a binary image. With each such pixel, we associate an aspect ratio of the horizontal to vertical run length. In this example, the horizontal run length is 14 and the vertical run length is 4 for the currently selected pixel.

Chapter 5

Results

Two separate tasks, to which the proposed framework can be applied, have been introduced in the previous chapter. This chapter describes the experimental results generated through the implementation of each. We begin with the simpler of the two, and then move to the more involved application.

5.1 Automatic determination of membership functions for fuzzy image processing filters

Now we must return to the problem described in Section 4.1, that is, to solve the combinatorial optimization of choosing a brightness membership function evaluated by fuzzy event entropy. This example fits our framework, as it is a parameter selection problem for image processing, but it does not push the limits of the framework: the agent selects parameters one at a time, and there are no intermediate or final images returned. This is not necessary, as the evaluation metric determining the reward assigned to the agent is solely based on the parameters (determining the S-function) and the histogram data of the input image. Fortunately, the entropy calculation consistently returns a value between 0 and 1, allowing us to directly use this result as reward, without having to manipulate it in any way.

Though function approximation or generalization is necessary for a highly-dimensional and/or

continuous parameter space, this particular example is discrete and small enough to be implemented in a more straightforward tabular form. In the case of the S-function, we are limited to three discrete parameters, which can take values between $[0, 255]$ but are limited to $0 \leq a < b < c \leq 255$. This produces a large state space, but one that can still be handled without function approximation. It also serves as a comparison, in terms of the learning rate, to the next application (Section 5.2), in which generalization is employed.

5.1.1 Experiment setup

We have implemented the algorithm with three table-based RL algorithms. Q-learning and Sarsa, which are very similar, and Sarsa(λ) which uses eligibility traces. Parameters were determined empirically, and in each case, were optimal (out of a discrete number of options). These are shown in Table 5.1. A trace threshold of 0.01 was also used in the case of Sarsa(λ). This means that traces with values less than this threshold were not stored for efficiency. The methods were then compared by each determining a brightness membership function for the Lena image. The methods were compared in terms of average reward received. This means that at each iteration, we have record the mean of the previous entropy values received up to and including that iteration. These values are plotted in Figure 5.1. Each test was run five times with each algorithm. Results shown are the averages of these trials. We have also compared the running times for 5000 iterations of each algorithm (Table 5.2). In all cases, we see that the algorithm converges far before the maximum number of iterations is reached.

It is of interest that Q-learning considerably outperforms Sarsa for this task. There are documented cases [57, 79] (although for different tasks) where the opposite is seen. As expected, Sarsa's performance improves with the additional consideration of eligibility traces. The running time, however, is significantly increased due to the consideration of eligibility traces. Instead of just considering one-step returns, the algorithm considers an entire history of returns. For the remainder of the MF experiments, we will use Q-learning to obtain results.

Table 5.1: Parameters used in the reinforcement learning algorithms, Application 1

Algorithm	ϵ	α	γ	λ
Q-learning	0.1	0.9	0.9	N/A
Sarsa	0.1	0.9	0.9	N/A
Sarsa-(λ)	0.1	0.9	0.9	0.75

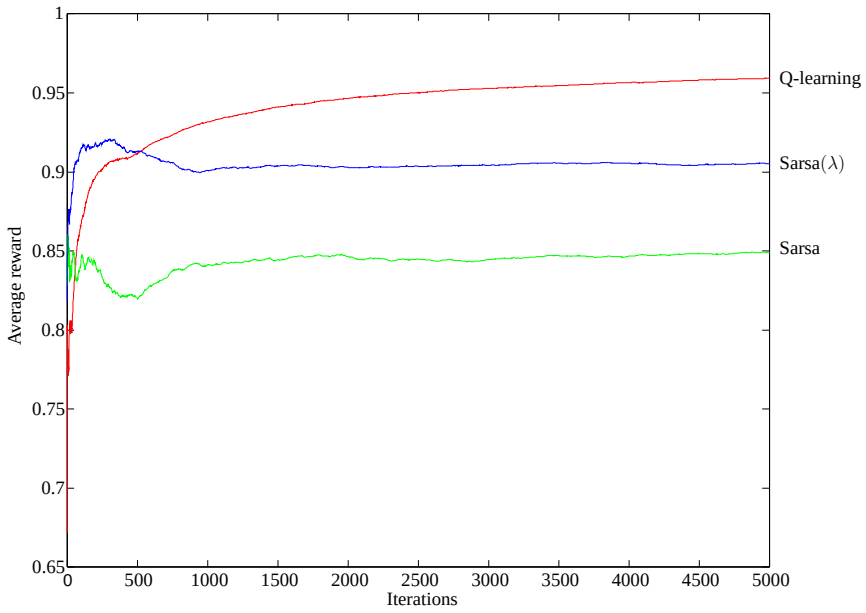


Figure 5.1: Comparing the reinforcement learning algorithms, Application 1. Average reward vs. number of iterations.

Table 5.2: Running times of reinforcement learning algorithms

Algorithm	Average running time (s)
Q-learning	6.5180
Sarsa	6.9380
Sarsa-(λ)	96.5046

5.1.2 Determining an optimal membership function

Next, we would like to compare our method with the simulated annealing (SA) method for determining the optimal membership function for several different images. We have implemented the simulated annealing method in [15] for comparison purposes (Algorithm 4). Five different images with varying histograms have been selected for testing. Due to the randomness involved in both approaches, we have run three trials for each image. The results for SA are presented in Table 5.3. The results for Q-learning are presented in Table 5.4. Simulated annealing requires a stopping criterion, and this was defined to be the point at which the state did not change for 100 iterations. The process of determining such a criterion for RL was somewhat more involved. The entire process was initially run for 5000 iterations to determine a maximum average reward. The tests were then run to a point at which the average reward was within 1% of the maximum reward.

The SA method tends to find very steep S-functions, all with the maximum entropy, 1.0. The results between trials are all very similar for the SA method. On the other hand, the Q-learning approach tends to find more varied curves, at high entropy values, but usually less than 1.0. The curves generated by the RL agent seem more natural, that is, curves that would be more likely to be generated by an expert. This indicates that there may be a fundamental problem in using only the fuzzy event entropy to evaluate S-functions. There are many oddly-shaped, narrow S-functions which evaluate to entropy 1.0. Interestingly enough, in their results, Cheng and Chen [15] report much more natural membership functions using the same method. There is no apparent reason why the membership functions we have generated here should be any different. Perhaps the authors have restricted the parameters to remain separated by some unreported constraints.

When considering the number of iterations, with the exception of the Dark and Bright Lena images, Q-learning converges at much fewer iterations than simulated annealing. SA does not exhibit as much of a sensitivity to the image as does Q-learning in terms of computational time.

We present the images, their histograms and respective membership functions (determined by both SA and Q-learning) in Figures 5.2-5.6. For the case of Q-learning, where several varied membership functions can be generated, we have chosen to display the one with highest fuzzy

Algorithm 4 Simulated annealing for determination of membership function

Initialize the cooling rate, $\alpha = 0.996$
Input the image and compute the histogram
Randomly generate an initial state, $\mathbf{X_{init}}$ using the following technique:
1. randomly select a from the interval $[0, 127]$
2. randomly select c from the interval $[128, 255]$
3. randomly select b from the interval $[a + 1, c - 1]$

Let $\mathbf{X_{curr}} = \mathbf{X_{init}}$ and set the initial temperature, T
repeat
  Randomly select a move from the move set $M_1, M_2, M_3, M_4, M_5, M_6$. Apply this move to the current state $\mathbf{X_{curr}}$, which produces a new state $\mathbf{X_{new}}$. If the produced state is an illegal state, try another move.
  Compute the cost change: $\Delta E = Cost(\mathbf{X_{new}}) - Cost(\mathbf{X_{curr}})$
  if $\Delta E \leq 0$ **then**
    $\mathbf{X_{new}}$ is a better state. Set $\mathbf{X_{curr}} = \mathbf{X_{new}}$
  else if $\Delta E > 0$ **then**
    Set $p = e^{-\Delta E/T}$
    Select random number r between $[0, 1]$
    if $r \leq p$ **then**
      Set $\mathbf{X_{curr}} = \mathbf{X_{new}}$
    else
      Do not change the current state
    end if
  end if
until Termination conditions are met
Termination conditions are the following: if an effective move cannot be found in 100 consecutive move attempts, it is assumed that the point reached is a stationary point.
There are six possible moves, defined as:
$M_1 : (a, b, c) \leftarrow (a - 1, b, c), M_2 : (a, b, c) \leftarrow (a + 1, b, c)$,
$M_3 : (a, b, c) \leftarrow (a, b - 1, c), M_4 : (a, b, c) \leftarrow (a, b + 1, c)$,
$M_5 : (a, b, c) \leftarrow (a, b, c - 1), M_6 : (a, b, c) \leftarrow (a, b, c + 1)$.
The function $Cost(X)$ is defined as

$$Cost(X) = 1 - H(\text{bright}; X) = 1 - H(\text{bright}; a, b, c) \tag{5.1}$$

where H is the entropy of a fuzzy event, in our case "bright".

Table 5.3: Generation of membership functions using simulated annealing

Image	$(a_{opt}, b_{opt}, c_{opt})$	Entropy	Iter.
	Trial 1		
Lena	(127,128,130)	1.0000	3469
Dark Lena	(26,29,30)	1.0000	3491
Bright Lena	(226,229,230)	1.0000	3826
Cameraman	(142,143,146)	1.0000	3243
Blood	(144,151,166)	1.0000	2758
	Trial 2		
Lena	(126,129,130)	1.0000	3520
Dark Lena	(27,28,30)	1.0000	3470
Bright Lena	(226,228,231)	1.0000	3493
Cameraman	(140,144,147)	1.0000	3373
Blood	(153,154,158)	1.0000	2462
	Trial 3		
Lena	(127,128,130)	1.0000	3391
Dark Lena	(27,28,30)	1.0000	3387
Bright Lena	(227,228,230)	1.0000	3367
Cameraman	(142,144,145)	1.0000	3252
Blood	(151,155,158)	1.0000	2594

Table 5.4: Generation of membership functions using Q-learning

Image	$(a_{opt}, b_{opt}, c_{opt})$	Entropy	Iter.
	Trial 1		
Lena	(119,125,143)	0.9999	2727
Dark Lena	(3,9,70)	0.9351	4031
Bright Lena	(98,247,255)	1.0000	4189
Cameraman	(48,108,244)	0.9998	3153
Blood	(54,110,247)	1.0000	1096
	Trial 2		
Lena	(53,127,195)	1.0000	1150
Dark Lena	(14,37,38)	0.9995	4353
Bright Lena	(107,253,255)	0.9575	4246
Cameraman	(103,153,168)	1.0000	1029
Blood	(88,136,210)	0.9996	1659
	Trial 3		
Lena	(22,98,250)	0.9996	2563
Dark Lena	(3,53,42)	0.9955	4423
Bright Lena	(95,254,255)	0.9445	4639
Cameraman	(13,121,254)	0.9999	2773
Blood	(15,165,218)	0.9999	2601

event entropy. In the case of the Cameraman image, RL has found a narrow membership function, but upon inspection of the parameters themselves, we see that it is still better than any MF found by SA. Let us now focus on the comparison of generated membership functions to the image histograms. In both the SA and RL cases, the membership functions shift based on the histogram. This naturally reflects what one may consider "bright" depending on the image.

5.1.3 Segmentation by thresholding

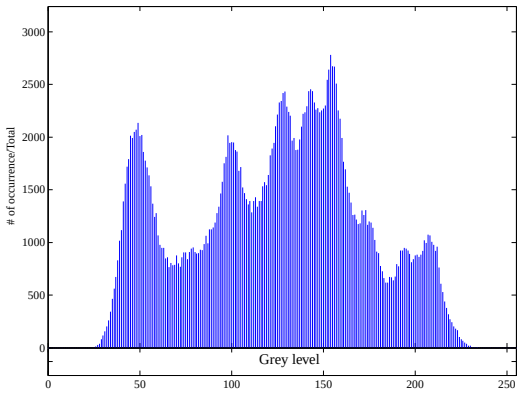
Image segmentation is an important area in the field of image processing. It is a process in which an image is broken up into various classes or sets such that each one is homogeneous with respect to one or more attributes of the image. This is useful for image understanding, texture analysis and classification of structures within the image.

Numerous methods for segmentation have been proposed in the literature, and more recent techniques have employed fuzzy set theory. Pal [53] proposed some of the earliest fuzzy segmentation techniques. Forero-Vargas [24] gives an overview of several techniques that use measures of fuzziness. Basir et al. [9] have developed an intricate fuzzy integral-based region merging algorithm for image segmentation.

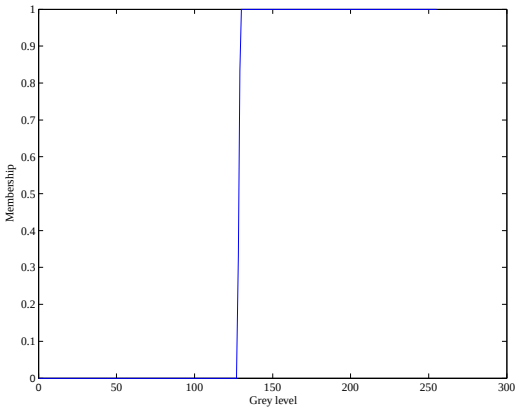
Thresholding is the simplest method of performing image segmentation. We do not claim that it can perform at par with some of the more involved methods above, but it is a useful way to demonstrate how we can apply the brightness membership functions generated by the RL agent. The idea is straightforward. Given the membership function, $\mu_{bright}(r_k)$, we can separate pixels based on the crossover point, $\mu_{bright}(r_c) = 0.5$. Pixels with greylevel less than r_c will be assigned to one class, while those with greylevel greater than or equal to r_c will be assigned to the other class. This is the same as saying that pixels that are bright with membership 0.5 or greater are classified as bright, and the other pixels are classified as dark. In essence, it is a very simple defuzzification operation. The thresholding approach can be extended to greater than two classes by employing local measures of fuzziness [9].

We have chosen to use the Blood image to demonstrate segmentation by thresholding. This image is often used to test segmentation algorithms because of its unique histogram (Figure 5.6(b)). A binary thresholding technique should be able to correctly separate the cells from the plasma. We have segmentated the image using the membership functions determined by both

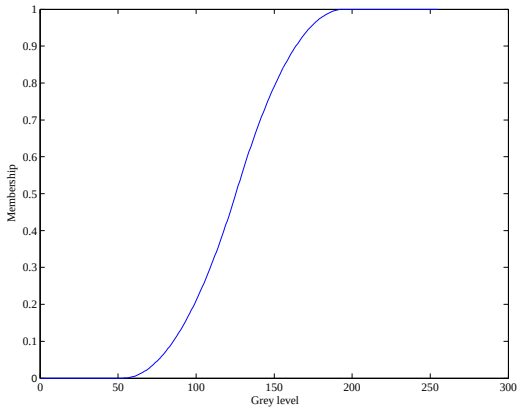
(a) Input image



(b) Histogram



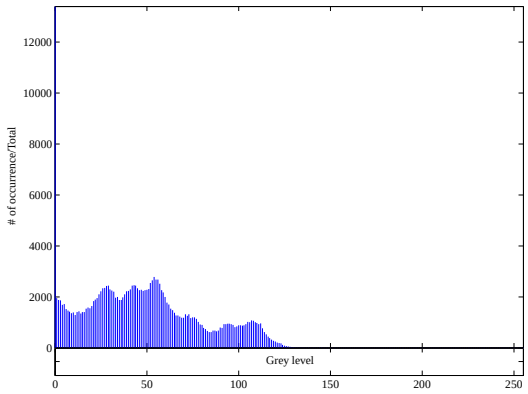
(c) Membership function determined by SA



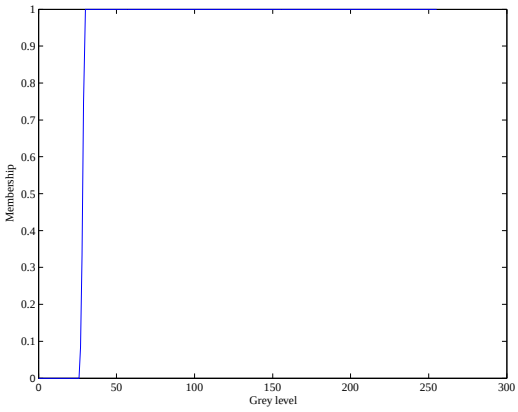
(d) Membership function determined by RL

Figure 5.2: Results for Lena

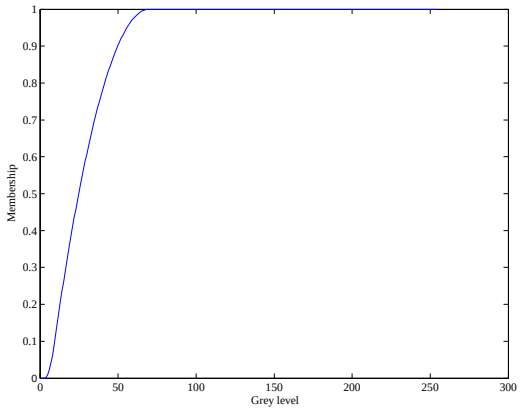
(a) Input image



(b) Histogram



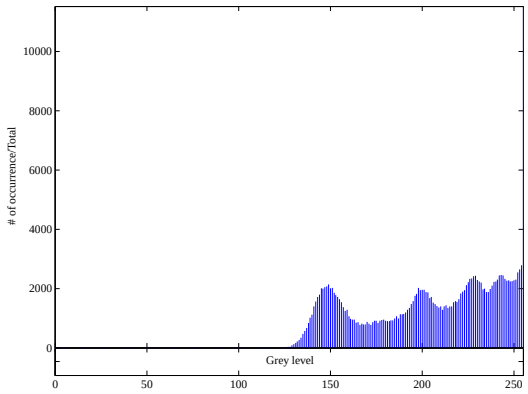
(c) Membership function determined by SA



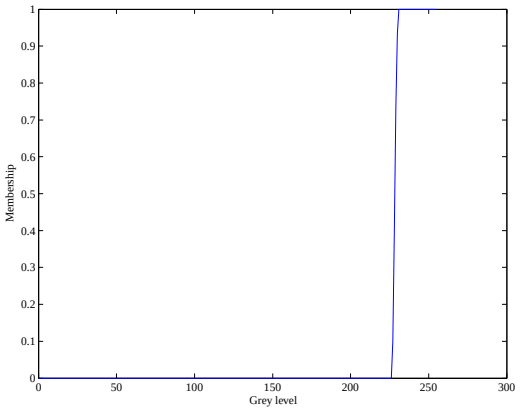
(d) Membership function determined by RL

Figure 5.3: Results for Dark Lena

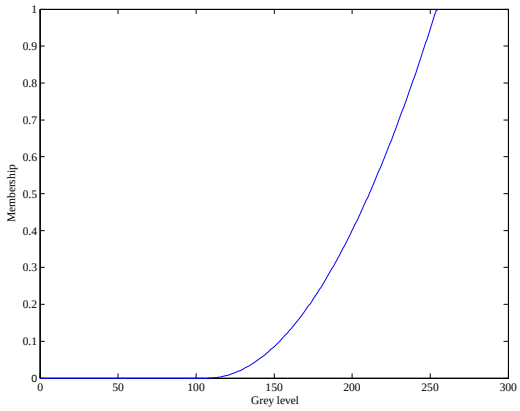
(a) Input image



(b) Histogram



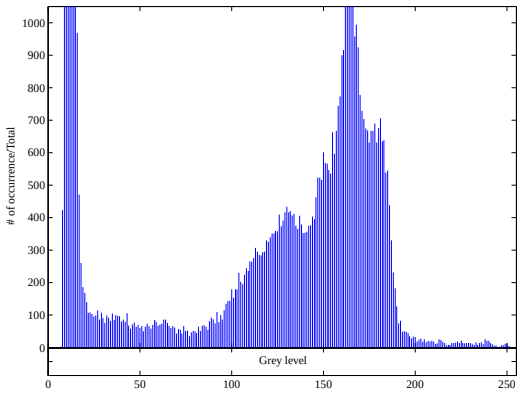
(c) Membership function determined by SA



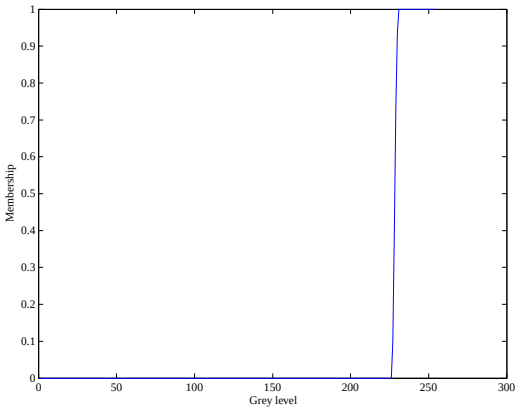
(d) Membership function determined by RL

Figure 5.4: Results for Bright Lena

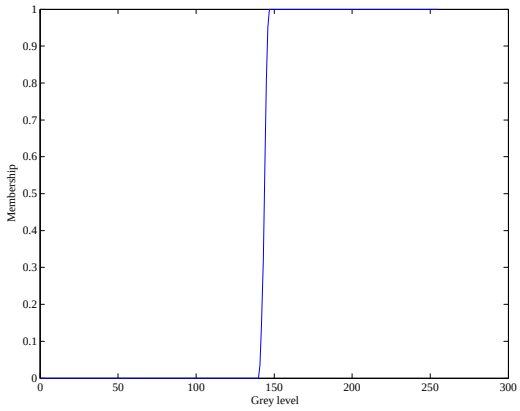
(a) Input image



(b) Histogram

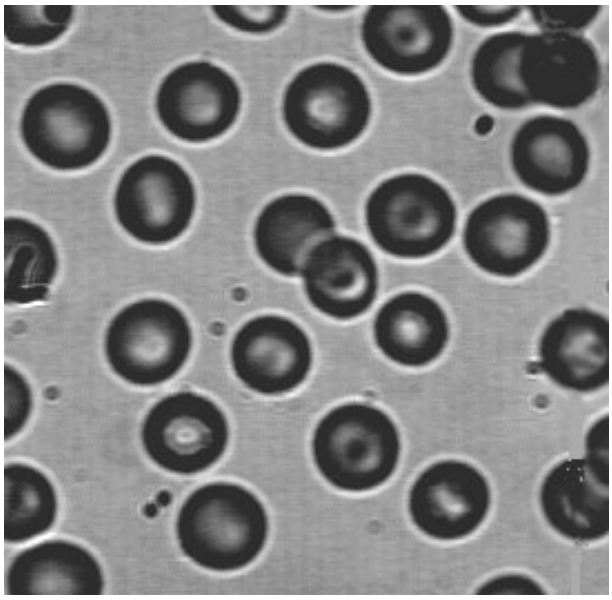


(c) Membership function determined by SA

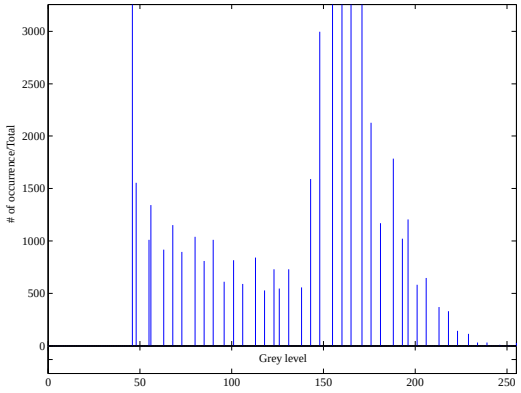


(d) Membership function determined by RL

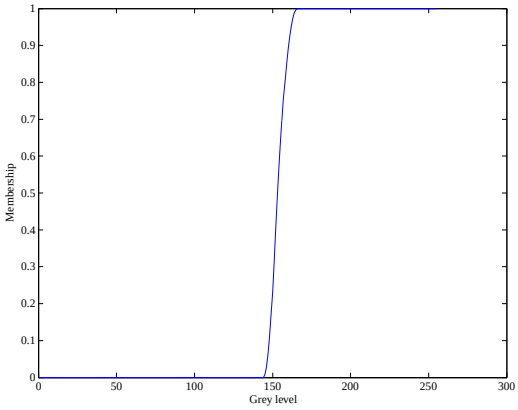
Figure 5.5: Results for Cameraman



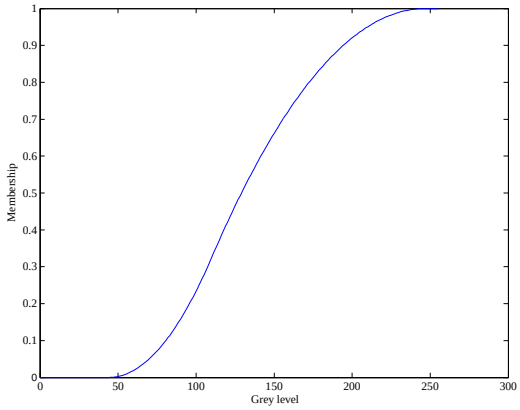
(a) Input image



(b) Histogram



(c) Membership function determined by SA



(d) Membership function determined by RL

Figure 5.6: Results for Blood

SA and Q-learning (Figure 5.6(c) and Figure 5.6(d)). We have also used Otsu's method [51], a well-known non-fuzzy technique for comparison. The thresholds selected by each method are given in Table 5.5. The segmented images are shown in Figure 5.7.

Table 5.5: Thresholds determined for segmentation of Blood image

Method	Threshold
Otsu's method	109
Simulated annealing	155
Q-learning	131

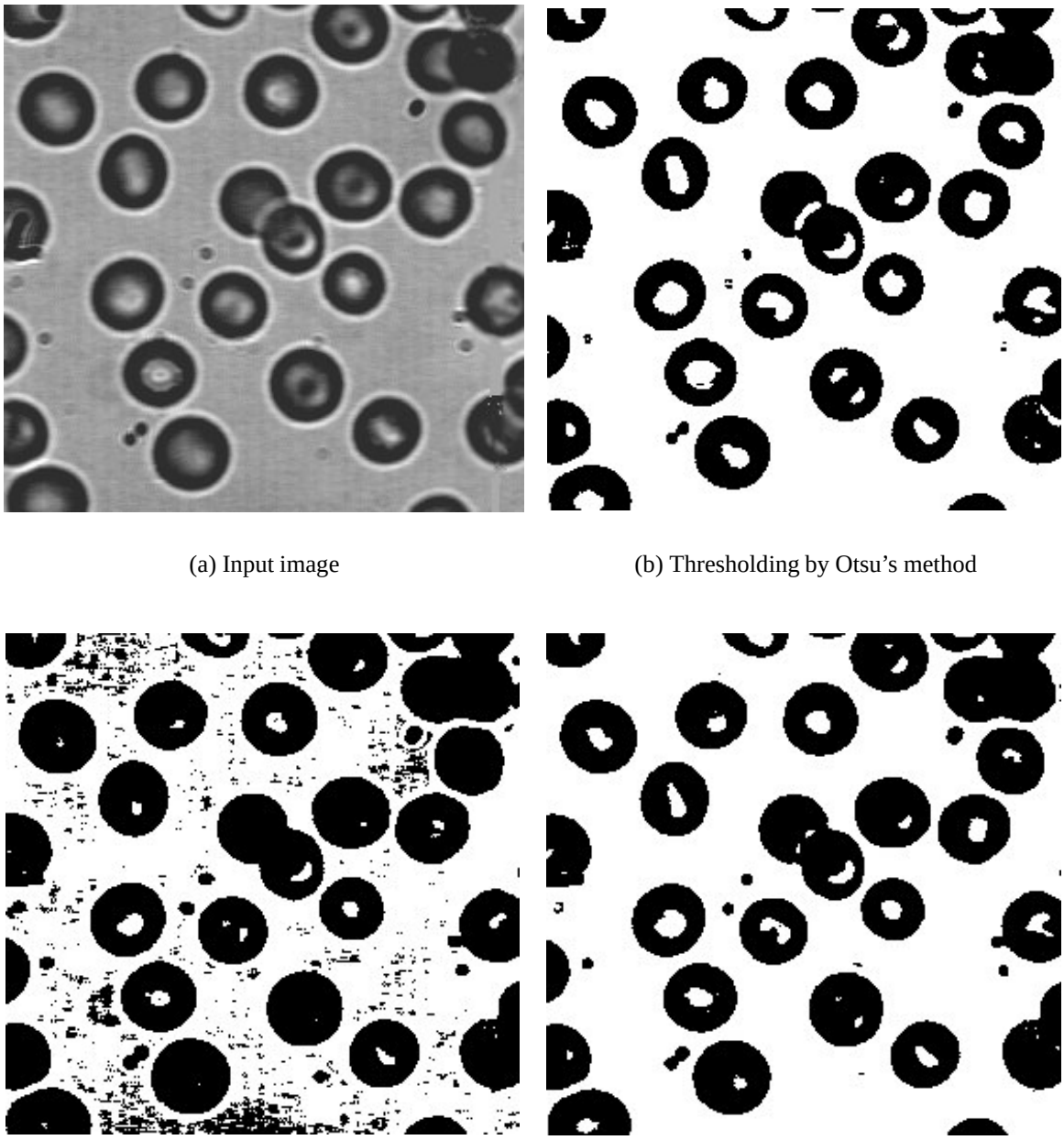
While Otsu's method still slightly outperforms both fuzzy methods, our reinforcement learning-based technique clearly is superior to the simulated annealing approach. Even though the membership function determined by the SA technique has maximum entropy of 1.0, the threshold is too high and as a result, the segmented image is very noisy.

The combination of these RL-generated membership functions with local thresholding methods would yield much more accurate thresholding and thus improved results, but that is beyond the scope of this thesis. Here, we focus on the agent's ability to generate appropriate MF.

5.1.4 Discussion

While the simulated annealing algorithm is always able to find a membership function with maximal entropy of a fuzzy event, these membership functions seem inferior to those produced by the reinforcement learning algorithm both on a subjective visual level and in the segmentation task. However, the Q-learning agent is also trained using entropy as reinforcement. This suggests that high entropy levels are a good indicator of an appropriate membership function, but maximal entropy values (1.0) may reflect strange membership functions that are certainly not desirable. In Section 4.1.1, we pointed out several other methods of measuring the goodness of a membership functions. It would be useful to employ such methods in combination with entropy as a reinforcement for the RL agent, and as a cost function for SA. The two parameter-finding schemes could then be compared once again.

Since the RL agent is rewarded only on entropy, we may question why it does not find the



(a) Input image

(b) Thresholding by Otsu's method

(c) Thresholding using MF determined by SA

(d) Thresholding using MF determined by RL

Figure 5.7: Segmentation of Blood image

membership functions with maximum entropy of 1, yet consistently finds MF with entropy near 1. Is this a flaw, or a merit, considering the majority of membership functions of entropy 1 are so steep? We believe that these results suggest that making decisions sequentially (as with RL) is a better approach than manipulating a vector of all of the parameters. The RL agent first finds a reasonable value of parameter a, before it chooses parameter b. It then chooses b based on its previous experience with this parameter, and so on. This avoids the situation observed in SA, where three particular parameters may provide a high reward when chosen together, but in combination with other parameters do not make sense. It is this sequential decision making that limits the discovery of steep MF with maximum entropy values.

In addition, we concude that both methods were able to stay away from local maxima, which is a clear advantage of non gradient-based optimization techniques.

5.2 Parameter control for a text-detection algorithm

We now discuss the results of controlling the parameters of Wolf et al.'s algorithm for detecting text in still images with a reinforcement learning agent. In this case, we manipulate 10 continuous parameters, and therefore generalization over the state space is necessary. The fuzzy ARTMAP is the tool used for generalization and DIRECT algorithm supports our choice of continuous actions.

The following results are divided into two subsections. The first, concerns the ability of the agent to learn, measured by its received reward over a number of iterations. The second, concerns the evaluation of the parameters chosen by the agent, not only on the training set of images, but on a separate set of images used for testing.

5.2.1 Learning

Learning was evaluated using a test set of 72 ground-truthed images containing artificial text, and 72 images which contained no text. All images were in format CIF (384×288 pixels). We experimented using three popular RL algorithms: Q-learning, Sarsa and Sarsa(λ) (described in Section 2.1.2).

Figure 5.8 shows the performance of each algorithm. Learning convergence was observed within 500 iterations by following the reward history. This is a remarkable improvement over the previous results (Section 5.1.2), where the number of iterations was nearly double for an optimization of far fewer parameters. This is clearly due to the generalization capabilities of the fuzzy ARTMAP. Due to the stochastic behaviour of the agent, these results are averaged over 5 trials, and then smoothed by plotting the mean of the past 100 iterations. We note that Q-learning provides significantly better performance, and therefore it is used in the following experiments. The results are surprising, as the incorporation of eligibility traces by Sarsa(λ) should improve temporal credit assignment. Even if the off-policy method, Q-learning, is more suited to the problem at hand, Sarsa(λ) should outperform Sarsa. A possible reason for this poor performance could be due to the nature of the problem. We are dealing with an episodic task, where the episodes are short and constant-length. The incorporation of eligibility traces could simply be unnecessary complication. The values of the eligibility traces are stored in the fuzzy ARTMAP network, so that we may generalize over them as well as Q-values. As generalization is performed over both inputs and outputs in the fuzzy ARTMAP, the performance degradation could therefore be implementation-related, where Q-values and eligibility traces have interfered with eachother.

Reinforcement learning parameters were determined empirically, and in each case, were optimal (chosen from a discrete set of values). These are shown in Table 5.6. A trace threshold of 0.01 was also used in the case of Sarsa(λ). Figure 5.9 demonstrates the affect of the step size, α, on learning performance. We can see that all values in the range of 0.2-0.9 offer comparable performance, while mid-range values are preferred. Other works [10, 11] have suggested that variable learning rates may be advantageous in terms of both learning speed and superior final Q-values.

For the action selection policy, we used the commonly-used $\epsilon-$ greedy policy (Section 2.1.3) and experimented with both constant values of ϵ, as well as values that decreased according to a simulated annealing-like cooling schedule:

$$\epsilon_t = \epsilon_0 \left(\frac{\epsilon_T}{\epsilon_0} \right)^{\frac{t}{T}}, \tag{5.2}$$

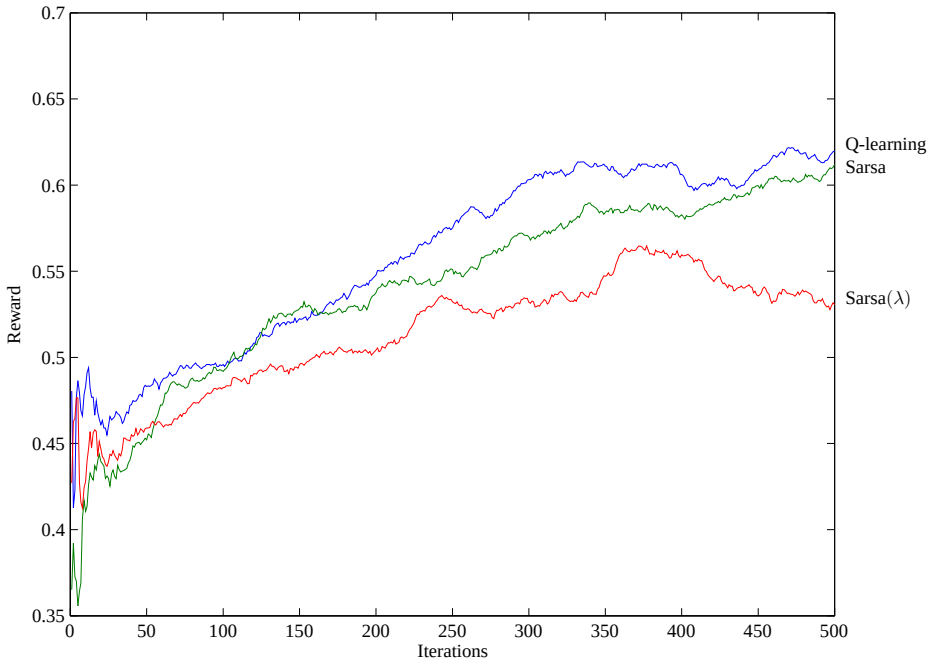


Figure 5.8: Comparing the reinforcement learning algorithms, Application 2

where ϵ_t is the probability of taking a random action at step t, where t increases from 0 to T. Here, we refer to total steps, and not iterations of the image-based algorithm (i.e. T represents the end of learning). The initial and final epsilon values were set to 0.9 and 0.01, respectively. Figure 5.10 compares the results of using a constant-valued ϵ policy versus the decreasing policy. Learning is seen to converge at a slower rate using a decreasing ϵ but converges on a higher average reward. Using a constant value of $\epsilon = 0.01$ is too low, and offers poor performance. The policy with decreasing ϵ has been used in the following experiments.

The fuzzy ARTMAP neural network contains several parameters of its own. We have used the parameter values recommended by Patrascu [56] with the exception of the ART$_b$ vigilance which was lowered to 0.9. We found that this improved learning performance by increasing the generalization over the outputs. This resulted in fewer categories created in the fuzzy ARTMAP to represent the Q-values. The parameters are provided in Table 5.7 and a description of each can be found in [14]. We note that the ART$_a$ vigilance is not set but controlled dynamically in order to ensure an output from the network [56].

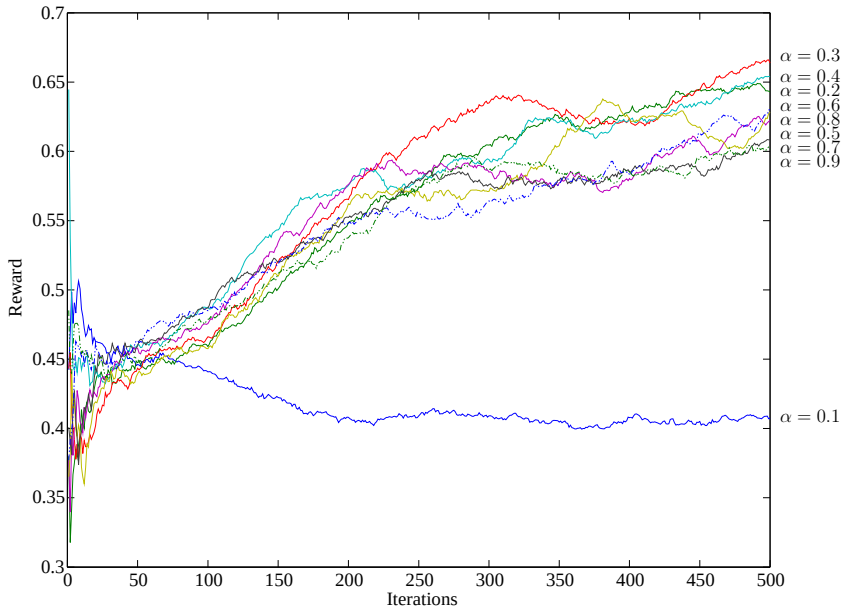


Figure 5.9: Effect of step size parameter, α

Table 5.6: Parameters used in the reinforcement learning algorithms, Application 2

Algorithm	α	γ	λ
Q-learning	0.3	0.99	N/A
Sarsa	0.3	0.99	N/A
Sarsa(λ)	0.3	0.99	0.7

Table 5.7: Parameters used in each fuzzy ARTMAP

Base vigilance	ART$_b$ vigilance	Learning rate	Mismatch
0.95	0.90	0.09	0.00001

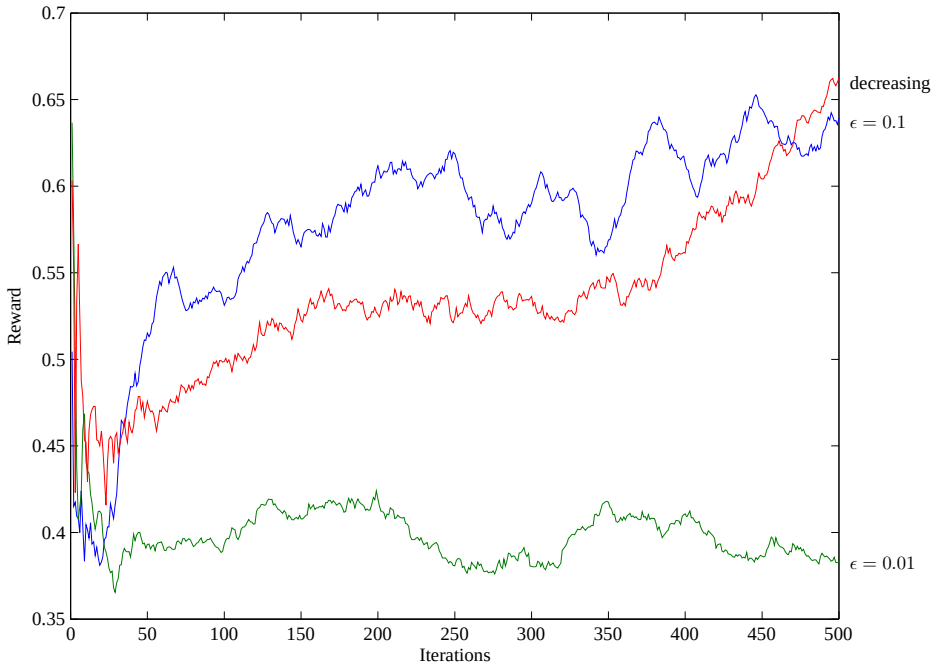


Figure 5.10: Comparison of action selection policies

Table 5.8 provides the optimal parameters recommended by the RL agent after 500 iterations, using the best learning conditions, as determined empirically. We note that these are a sample set of parameters, and as in the case of the membership functions, the stochastic nature of RL results in the possibility of different parameter sets recommended at each trial. These are compared to Wolf et al.'s recommended parameters in [92]. The reader will notice that the two sets of parameters differ greatly. Parameters 1-3 and 8-10 have been chosen close to the midpoint of their ranges. We are certain that these values reflect the starting point of the DIRECT algorithm, though the stochastic policy of the RL agent has assured that values throughout the domain have been tried. The parameters will change to reflect the set of images used in training.

Table 5.8: Chosen parameters

Set	1	2	3	4	5	6	7	8	9	10
Wolf	13	87	2	105	50	1.2	0.3	0.1	0.2	0.7
RL	19.78	100.49	3.05	102.90	101.60	4.72	0.68	0.48	0.51	0.48

5.2.2 Cross validation

We used three-fold cross validation in order to evaluate the system with the learned parameters. Two thirds of the images have been used for training and one third for testing, the mean value over the three runs being the final result. The tests have also been repeated three times for each set, and results averaged. Table 5.9 provides the results of evaluating the parameters chosen by the agent. We also apply Wolf's recommended parameters to each test set for comparison.

We present four measures: recall, precision (using only text-containing images), precision (considering both text and non-text images) and the harmonic mean of the recall and second precision measure. Note that the recall remains the same as we add non-text images, but the precision decreases as we have more and more falsely detected text regions. As in an information retrieval system, this is an effect of the generality of the dataset (see Sect. 4.2.4).

Table 5.9: Parameter performance

Set	Rec	Pr (Text)	Pr (T+NT)	HM
Previously recommended parameters				
Trial 1	0.7703	0.4043	0.1926	0.3081
Trial 2	0.7970	0.4242	0.1899	0.3067
Trial 3	0.8162	0.4081	0.1980	0.3187
Mean	0.7945	0.4122	0.1935	0.3112
RL determined parameters				
Trial 1	0.5081	0.7520	0.6164	0.5570
Trial 2	0.5091	0.9081	0.9081	0.6524
Trial 3	0.3405	0.9692	0.9333	0.4990
Mean	0.4526	0.8764	0.8193	0.5695

While Wolf et al.'s suggested parameters lead to better recall, they reflect the assumption that all images contain text, and thus lead to many more falsely detected text regions. The parameters determined by the RL agent balance the consideration of text-containing and non text-containing images, thus leading to significantly better recall values. The harmonic mean of precision and recall averaged over all three trials for the RL-determined parameters is 83% greater than the same measure using the previously recommended parameters (0.5695 as compared to 0.3112). Each individual trial is also improved using the RL-determined parameters.

5.2.3 Intermediate reward

Section 4.2.4 describes two types of intermediate reward that could be applied to the system with the intent of improving performance. We have repeated the experiments above, maintaining the terminal reward derived from ground truth rectangle matching, while adding the Bhattacharyya distance and classification error rate first separately and then simutaneously.

Figure 5.11 shows the learning performance in all of the test cases. Performance has been measured only in terms of achieving the final goal, that is detection of text regions, and not the intermediate rewards of well-separated text and non-text distributions and good binarization. This not only reflects the true goal of the agent, but also allows the results to be compared at the same level. We have not observed much difference in the learning performance, with the exception of the final average reward values settling slightly higher for the Bhattacharyya distance case on data sets 1 and 2, and the less-regular shape of the curve for set 1 when using both the Bhattacharyya distance and classification error rate. Thus we can conclude that the addition of neither of these particular intermediate rewards has a significant effect on the rate of learning.

The cross validation experiments have also been repeated and the results are summarized in Table 5.10. On average, performance (measured by the harmonic mean of recall and precision) increases when the Bhattacharyya distance is added to the reward signal, while the effect of the Classification error rate is negligible. These results are consistent with the performance when both intermediate reward signals are added. In this case, the performance does not differ greatly from the Bhattacharyya distance-only case. Therefore we conclude that in repeated experiments, incorporating the Bhattacharyya distance as intermediate reward results in better parameter selection.

5.2.4 Inclusion of image features

Since the RL agent is constantly learning, even without image features, its value function will reflect the latest images on which it has learned. Thus it would adapt to a general change in input images or video. Its state space is constructed from the past parameters selected, as well as the current step of the image-based task; in this case, specifically the current step of the text-detection

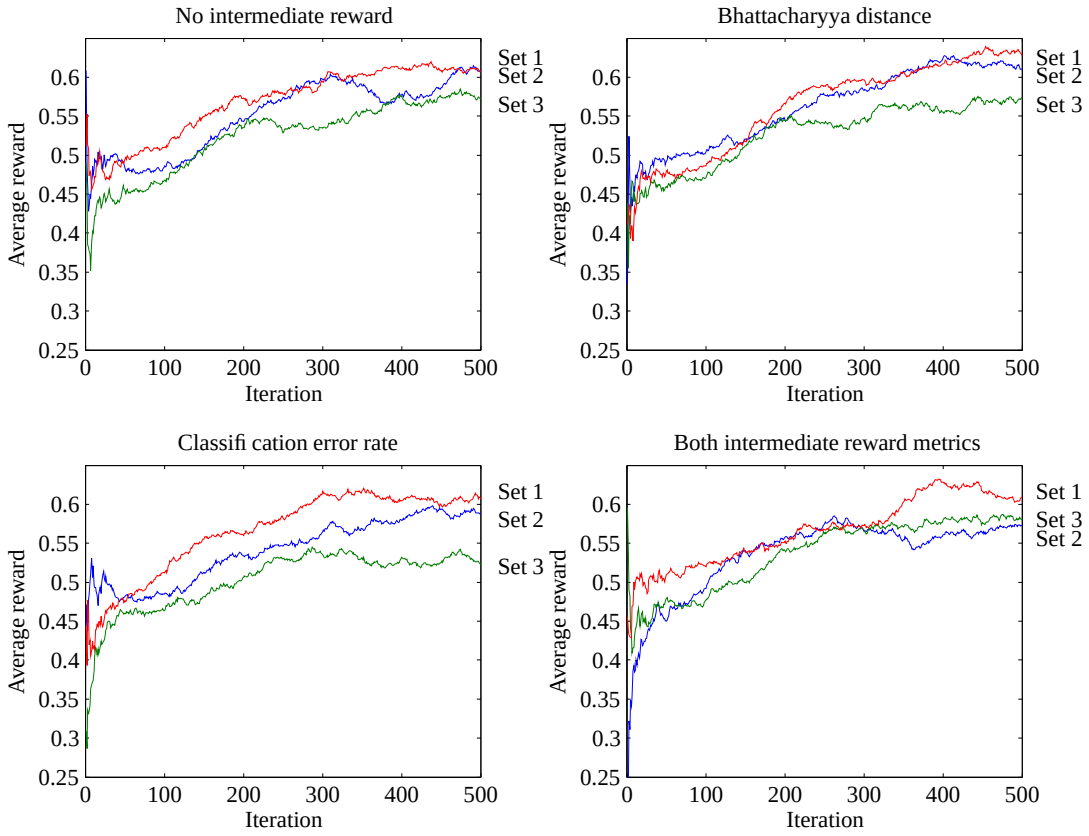


Figure 5.11: Adding intermediate reward signals. Average reward vs. number of iterations.

Table 5.10: Parameter performance with intermediate reward

Set	Rec	Pr (Text)	Pr (T+NT)	HM
No intermediate reward				
Trial 1	0.39459	0.71371	0.62309	0.47649
Trial 2	0.46162	0.85524	0.75732	0.54122
Trial 3	0.37928	0.74076	0.64125	0.44734
Mean	0.41183	0.76990	0.67389	0.48835
Bhattacharyya distance				
Trial 1	0.54324	0.70130	0.52152	0.53029
Trial 2	0.38081	0.88363	0.75292	0.47803
Trial 3	0.44595	0.75297	0.65121	0.52446
Mean	0.45667	0.77930	0.64188	0.51093
Classification error rate				
Trial 1	0.40541	0.84132	0.74584	0.49959
Trial 2	0.41919	0.83409	0.72074	0.50484
Trial 3	0.30721	0.69291	0.66068	0.41199
Mean	0.37727	0.78944	0.70909	0.47214
Both intermediate rewards				
Trial 1	0.37658	0.76571	0.69691	0.48149
Trial 2	0.54141	0.88528	0.70947	0.60034
Trial 3	0.37387	0.81545	0.73970	0.47358
Mean	0.43062	0.82215	0.71536	0.51847

algorithm. Though effective, this behaviour is not ideal, as in many cases, visual information may change faster than the agent can respond. We have demonstrated that the agent can choose a good global set of parameters. A logical next step is a system that adapts its behaviour to visual input.

The cross validation experiments have been repeated to determine the effect on the inclusion of image features in the state space on the quality of parameters selected by the agent. These experiments differ from the previous experiments in that only a single image, selected randomly from the set, is used per iteration, as opposed to the averaging of 3 "text" and 3 "non-text" images per iteration. This is due to the features being image-specific, and therefore the state space must be constructed with respect to the current image. This permits the use of only one image per iteration. We have executed the tests using both the features described in Section 4.2.5 and without features. All other variables have remained constant between the two groups.

Table 5.11 provides the 3-fold cross validation results. In all three datasets, the inclusion of image-based features has a positive effect on the parameter performance, measured by the harmonic mean of precision and recall. We note that these results are comparable to the previous tests where we used six images per iteration. This demonstrates that the averaging has a significant positive effect on the results, as do the features. Unfortunately, the two are incompatible and cannot be used simultaneously.

Table 5.11: Parameter performance with image features

Set	Rec	Pr (Text)	Pr (T+NT)	HM
Not using image features				
Trial 1	0.39910	0.79877	0.72147	0.49668
Trial 2	0.52121	0.87594	0.67331	0.52086
Trial 3	0.35315	0.67991	0.53044	0.36025
Mean	0.42449	0.78487	0.64174	0.45926
Using image features				
Trial 1	0.51802	0.66632	0.57277	0.54359
Trial 2	0.53636	0.89970	0.72013	0.61252
Trial 3	0.36126	0.62808	0.45952	0.40410
Mean	0.47188	0.73137	0.58414	0.52007

5.2.5 Discussion

In the learning phase, the agent has been seen to achieve higher reward by choosing better parameters in consecutive iterations. We observed convergence within 500 iterations, a speed improvement over the case without the fuzzy ARTMAP. The results have been smoothed, as we are continuously using different images as input (at each iteration), and therefore the return at each successive step can differ greatly depending on the input. So we are not as concerned as with the change in reward between steps as we are with an average improvement over time.

Upon inspection, the parameters obtained from the approximated value function after training is complete are very different than the recommended parameters in [92]. However, the proximity of some of these parameters to the midpoints of the ranges suggest that the DIRECT optimization method may bias the selection of parameters. This may be a disadvantage of such a method, which always begins its sampling at the midpoint of parameter space.

In the cross-validation phase, the agent adapts its performance to the inclusion of non-text images, and succeeds in maintaining a good level of recall on text-images. This demonstrates that a method which adapts to the type of input images is advantageous, compared to statically chosen parameters.

Upon consideration of intermediate reward, the two metrics introduced: Bhattacharyya distance and classification error rate did not have an effect on the learning rate. This is contrary to what we had expected. The addition of Bhattacharyya distance, however, resulted in the agent choosing a better parameter set, and thus performance was increased on average by 5%.

The inclusion of image features lead to a performance increase of 13% over identical experiments without features. The nature of the feature extraction process, however, prevented us from using the technique of using 3 “text” and 3 “non-text” images per iteration, and averaging to smooth the response. The best results using features were at par with the best results using averaging without features. A method to combine the benefits the mutually exclusive techniques would be ideal.

We believe that the system could be significantly improved in one major respect. This would not be without a considerable amount of programming as well as theoretical considerations. So far, we have developed a simulated system, that could not have worked without ground truth images. Though it has learned a good general set of parameters, we would prefer a system that

can dynamically update its parameters based on input. An online system does not necessarily mean abandoning the knowledge imparted on the system through explicit training. Reinforcement learning research has recognized that providing examples, or "training runs" can drastically improve performance [40, 74]. In the early stages of learning, RL agents explore their world randomly. In the case of situations where only a few states are relevant, the agent may never experience these, unless an external teacher provides some guidance during training. This is especially true for reward functions that are mostly uniform, most state-action pairs receiving approximately the same reward, except for a rare few. We must wait for a long time, until the agent "stumbles" upon some interesting state, receives the reward, and then that reward is bootstrapped.

In our specific case, the agent could be trained with ground truth images (an off-line mode), to a point where it has acquired some knowledge about the parameter space. Then, it could be placed on-line, adapting the parameters to the current stream of images or video. If the visual information changes in some way, then the rewards received using the current policy of selecting parameters would change. The agent would then adapt its behaviour to receive maximum reward. In this system, there still exists a major question of how to transfer knowledge when the reward function changes. Clearly, the current reward system involving ground truth images could not be applied when these images were not available. Thus, the information contained in the value-function estimation with respect to the ground truth reward function would need to be available with respect to the new reward function. This is an open research question.

Upon considering the on-line reward function, we have two proposals. We first suggest that reward or punishment could be derived from the optical character recognition (OCR) phase following detection. This measure would accurately reflect the text content of the OCR stream. High reward would be given for text strings (regardless of their content), and low reward (or punishment) would be given for characters detected that did not form text strings. One challenge here would be in deriving this reward. Another challenge, possibly greater, would be in rewarding the agent for detecting no text, when the image itself contained no text. In this case, the OCR stream would be empty. We could not generate a high reward for such empty OCR streams, as then the agent would simply learn to choose parameters that caused it to never detect text. Possibly a balance could be reached between high reward (for strings containing text), less reward

(for empty strings) and punishment (for strings containing illegible text).

Another idea for on-line reward is derived from the geometrical properties of the detected rectangles and the pixels within these rectangles. By calculating the reward on the text detection output rather than the OCR output, we would reward the agent for the true goal of text-detection, and not OCR, whose performance is dependent on later steps (binarization, and in the case of video: temporal tracking and multiple frame integration). Such reward could potentially address the chicken-and-egg problem of detection: that is, we would like to use geometrical features for detection, but they can only be calculated after the text has been detected [91]. RL would introduce a feedback loop such that the geometrical features would influence the parameters controlling detection. But whether these features truly represent the goal of text detection is unknown. We would hope the agent would not find some way to exploit the geometrical properties yet not arrive at properly detected text. This would only be seen after experimentation. The challenge here would be in combining the various metrics derived from the geometrical properties. Again, we must arrive at a single value of reward.

Constructing a viable reward signal that doesn't require ground truth is the last step toward an on-line system. Such a system would clearly distinguish the advantages of a reinforcement learning framework over other optimization methods.

Chapter 6

Conclusions and Future Work

Efficiently determining the optimal parameters for an algorithm that processes visual information is a challenging task. Both the dimensionality of the parameters and information to be processed can limit attempts at solving such a problem. This thesis has proposed a reinforcement learning framework which uses connectionist systems to generalize over the state and action space to handle this problem even in the case of a highly-dimensional, continuous parameter space. We view parameter selection as a decision process over time, where past decisions may affect future decisions.

Defining state and reward has been the major design obstacle in developing the framework. Two tools have been introduced to cope with the more general challenges of RL with visual information. The fuzzy ARTMAP artificial neural network has been demonstrated to be an effective way to manage large state and action spaces. The DIRECT algorithm was successfully employed to allow for continuous action spaces while minimizing the computational cost of finding an action with the maximum value-estimate for a given state. Both of these applied may be extended beyond the parameter control task to facilitate image-based and non image-based RL.

The first application considered was the determination of an optimal fuzzy membership function to represent the "brightness" of a particular image. The entropy of a fuzzy event can be used to judge the optimality of such a membership function. The problem then becomes one of combinatorial optimization. We have shown through several experiments generating MF, and through

the application of image segmentation, that our RL framework is superior to the past efforts with simulated annealing. Our work found many irregular MF that could possess maximum entropy of a fuzzy event, but failed to describe the brightness of the respective images. This result was not demonstrated in the previous work, which initially presented the entropy measure as an evaluation metric. This suggests that entropy of a fuzzy event may not be the ideal measurement to provide reinforcement nor judge the quality of MF.

Next, the utility of the framework has been demonstrated by optimizing a set of ten parameters for an algorithm to detect text in images taken from video sequences. The algorithm has been proposed to aid in video and image indexation, such that the semantic knowledge contained in the text can be considered while indexing. The RL agent is able to find a set of global parameters superior to the previously recommended parameters, based on traditional measures taken from the field of information retrieval. Its performance has verified through three-fold cross validation. We have also defined a series of compact image features specific to the text-detection application, and demonstrated that introducing image features into the state space allows for parameter selection on a per-image, rather than per-set basis. These experiments have led us to propose three techniques for improving parameter control by learning:

- By using several images at each step, and then averaging the results of using a single chosen set of parameters, we can reduce the sensitivity to any particular image, and thus learn more with respect to the agent's action rather than the image itself.

- The Bhattacharyya distance used as an intermediate reward has also lead to a better learning of parameters. This measure is specific to the text-detection application, but it supports the idea of carefully chosen intermediate reward as opposed to terminal reward only.

- Image features permit a better choice of parameters, as the agent can choose based on characteristics of the current image rather than only its past experience. The use of features, however, is incompatible with the image averaging technique.

From an application standpoint, we will focus our immediate efforts along one major track. While the use of ground-truthed images has been useful in evaluating our framework, their use may be questioned when using methods capable of on-line learning. Now that the approach has

proven successful, our aim is to replace the ground truth-based reward by some reward received directly from either an optical character recognition (OCR) engine or geometrical properties of the text rectangles and the text pixels within these rectangles. This will permit on-line parameter optimization, which further justifies the application of RL.

Other future areas of interest lie in selecting more image features to form state information. So far, we have limited ourselves to hand-chosen features of a limited number. For example, we have computed the mean value of all of the maximum response Gabor features over all windows to reduce dimensionality. We would like to add more information, but as with many classes of features, it is expected that they will form a space of high dimensionality, that must be reduced even before we apply the FAM for generalization. One can employ such methods as Principal Component Analysis [21] to cope with the problem of excessive dimensions. Efforts could be made on the integration of intelligent techniques for feature selection.

The search for a challenging application on which to test the proposed framework has resulted in the implementation of the text-detection algorithm. This problem has not only required the bulk of development, it already poses many questions for future research and thus dominates the immediate horizon. Despite the encouraging results and proposed improvements from and to this particular application, the main contribution of this thesis is still a reinforcement learning framework that employs both the fuzzy ARTMAP and DIRECT optimization so that it is extendable to a wide class of image-based problems. We have proposed this system in the hopes that it will be the basis for many more parameter-control applications. In any such future endeavours, the reward signal and state-defining features will need to be defined as they are task-specific. Beyond parameter control, we will continue to see applications of agents learning from visual information by reinforcement. Further developments in theoretical RL as well as related fields, such as neural network-based generalization methods, will permit stronger results and more diverse applications.

Appendix A

Wolf et al.'s Text Detection Algorithm

This appendix provides details on the text detection algorithm for which we have chosen to control the parameters using reinforcement learning. Details on the function of the parameters themselves are also provided. The material presented here forms the basis for the task-specific reward (Section 4.2.4) and image features (Section 4.2.5).

Grey level constraints

As we have decided to focus on horizontal, artificial text, we can assume high contrast of the characters against the background image. We also can assume that text can be extracted using only luminance information. Therefore, we convert all frames into grey scale images before processing. To further justify this decision, compare watching a series of commercials on a colour television to watching the same series on a black and white television. Text is still visible on the black and white display, with the exception of very rare circumstances.

The detection method is based on the regular texture of text characters which form horizontally aligned vertical strokes. We apply the horizontal version of the Sobel operator as a gradient measure of the input, I and then use a slightly modified version of the LeBourgeois algorithm

[39], which detects the text with a measure of accumulated gradients:

$$\boldsymbol{A}(x,y) = \left[\sum_{i=-\lfloor S/2 \rfloor}^{\lfloor S/2 \rfloor} \left(\frac{\partial \boldsymbol{I}}{\partial x}(x+i,y) \right)^2 \right]^{\frac{1}{2}}, \tag{A.1}$$

where S is the size of the accumulation window. The response of this filter is a measure of the probability of each pixel to be part of text.

Next, we apply a two-threshold version of Otsu's global thresholding algorithm [51]. We assume two distributions in the image ("non-text" and "text"), and an optimal threshold is calculated from the grey value histogram by maximizing a criterion used in discriminant analysis, the inter-class variance:

$$t_{opt} = \arg\max_t \left(\omega_0 \omega_1 \left(\mu_1 - \mu_0 \right)^2 \right), \tag{A.2}$$

where $\omega_0 = \sum_{i=1}^{t} \frac{\boldsymbol{h}_i}{N}$ is the normalized mass of the first class, $\omega_1 = \sum_{i=t+1}^{L} \frac{\boldsymbol{h}_i}{N}$ is the normalized mass of the second class, μ_0 and μ_1 are the mean grey levels of the respective classes, $\boldsymbol{h}$ is the grey level histogram, N the number of pixels and L the number of histogram bins, equalling the number of grey levels.

In order to make the binarization decision more robust, Wolf et al. suggest a second threshold and change the decision for each pixel as follows:

$$\begin{array}{lll} \boldsymbol{I}_{x,y} < k_l & \Rightarrow & \boldsymbol{B}_{x,y} = 0, \\ \boldsymbol{I}_{x,y} > k_h & \Rightarrow & \boldsymbol{B}_{x,y} = 255, \\ \boldsymbol{I}_{x,y} \leq k_h & \Rightarrow & \boldsymbol{B}_{x,y} = \begin{cases} 255 & \text{if there is a path to a pixel,} \quad \boldsymbol{I}_{u,v} > k_h \\ 0 & \text{otherwise,} \end{cases} \end{array} \tag{A.3}$$

where k_h is the optimal threshold calculated by Otsu's method and k_l is calculated as:

$$k_l = \mu_0 + \alpha \left(k_h - \mu_1 \right), \tag{A.4}$$

where α is a parameter. Note the path is only composed of pixels (x,y) such that $\boldsymbol{I}_{x,y} > k_l$. The

output of this step is a binary image distinguishing text pixels from background pixels.

Morphological constraints

Next, we perform the following series of steps composing of morphological operations:

Step 1: Morphological closing (1 iteration).

This first step serves to close small holes in the components as well as connect components separated by small gaps. The structuring element used is:

$$\boldsymbol{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \textcircled{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{A.5}$$

Step 2: Removal of small horizontal bridges between connected components.

This step aims to disconnect text and non-text components connected by small horizontal bridges. To accomplish this, we remove pixels of columns whose local connected component's height is under some threshold. First, a matrix, $\boldsymbol{A}$ is built which is the same size as the input image. Each element of this matrix corresponds to a pixel of the input image and it stores the local height of the connected component to which it belongs. Then a thresholding operation removes pixels (x, y) such that $\boldsymbol{A}_{x,y} < t_1$, where t_1 is a fixed threshold parameter.

Step 3: Conditional dilation (16 iterations).

The next two steps serve to correct a shortcoming of the previous step. That is, the possible separation of noisy text regions into two or more connected components. We wish to connect any loose characters that are part of words, and thus we want to merge all connected components which are horizontally aligned and who have similar heights. The structuring element is as follows:

$$\boldsymbol{B}_C = \begin{bmatrix} \textcircled{1} & 1 \end{bmatrix}, \tag{A.6}$$

but the implementation is non-standard. First we must perform a connected components analysis for 4-connected neighbourhoods on the binary input image. We assume that each connected

component corresponds to a character or a word. Similar to the previous step, we build a matrix, $\boldsymbol{H}$ with size identical to the input image, and with every element of the matrix corresponding to one pixel of the input image. This time, however, each element contains the height of the respective connected component. We then construct a second matrix, $\boldsymbol{P}$, which holds for each pixel the y co-ordinate of the respective connected component. Next, we replace each zero value with its smallest non-zero neighbour to the right of the same row, as follows:

$$\boldsymbol{H}_{x,y} = \begin{cases} \boldsymbol{H}_{x,y} & \text{if} \quad \boldsymbol{H}_{x,y} \neq 0, \\ \boldsymbol{H}_{v,y} : v = \min\limits_{u>x} (\boldsymbol{H}_{u,y} > 0) & \text{otherwise.} \end{cases} \tag{A.7}$$

The same "smearing" of zero values is also applied to the $\boldsymbol{P}$ matrix. The two resulting matrices are then used to perform the conditional dilation. The algorithm traverses the image line-by-line from left to right. Each non-text pixel (x, y) preceded by a text pixel is set to a pseudo colour (which is neither 0 nor 255) under certain conditions, based on the relative heights stored in the $\boldsymbol{H}$ matrix. The general difference function for any two values, a and b, used for the conditions is defined as follows:

$$\triangle (a, b) = \frac{|a - b|}{\min (a, b)}. \tag{A.8}$$

If both of the following conditions are met:

- The relative difference of the heights of the connected component which includes a given pixel and its neighbouring component to the right does not exceed a defined threshold,t_2:

$$\triangle (\boldsymbol{H}_{x-1,y}, \boldsymbol{H}_{x,y}) < t_2. \tag{A.9}$$

- The relative difference of the y positions of these two connected components does not exceed a defined threshold, t_3:

$$\triangle (\boldsymbol{P}_{x-1,y}, \boldsymbol{P}_{x,y}) < t_3, \tag{A.10}$$

and the maximum number of iterations has not been exceeded, then the current pixel is set to the

pseudo colour. Otherwise, it does not change.

Step 4: Conditional erosion (16 iterations).

Again, the structuring element is $\boldsymbol{B}_C$ (Eq. A.6). In this step, we place conditions on the grey levels of the pixels rather than the shape. The image is eroded using the structuring element $\boldsymbol{B}_C$ with the additional condition that only pixels marked with the pseudo colour are eroded. As a final step, the remaining pixels marked with the pseudo colour are set to text.

Step 5: Horizontal erosion (12 iterations).

This erosion step serves to eliminate components that do not respect the hypothesis that text has a certain minimum length. The hypothesis, already imposed in the greylevel constraints (by the accumulating gradients step), distinguishes between small components containing high contrast vertical strokes and text with a minimum length. The following structuring element is used:

$$\boldsymbol{B}_H = \begin{bmatrix} 1 & \textcircled{1} & 1 \end{bmatrix}. \tag{A.11}$$

Step 6: Horizontal dilation (6 iterations).

After we have eroded the components, the size of the remaining components is reduced. This step aims to restore these components to nearly their original size, by dilating with the same structuring element, $\boldsymbol{B}_H$. To avoid reconnecting text components with non-text components, only half of the previous number of iterations is used.

Following the dilation operation, a connected components analysis is performed to extract the remaining components. Their bounding boxes are calculated and grown by 3 pixels to the left and 3 pixels to the right in order to account for the difference in number of iterations between horizontal erosion and dilation. These bounding boxes (which we also will call rectangles) are evaluated in the following series of geometrical constraints.

Geometrical constraints

After the morphological phase, we still expect to be left with a certain number of false rectangles which represent non-text regions. We impose a series of geometrical constraints in order to

eliminate as many of these as possible. The first two general constraints are imposed:

$$\frac{\text{width}}{\text{height}} > t_4, \text{and} \tag{A.12}$$

$$\frac{\text{number of text pixels of the component}}{\text{area of the bounding box}} > t_5, \tag{A.13}$$

where t_4 and t_5 are fixed thresholds. Then, some special cases are considered, where the sizes of the bounding boxes are increased to accommodate the heads and tails of characters, which have been eliminated during the morphological operations.

The final task is to combine two partly overlapping rectangles if they meet *one* of the following conditions:

$$\begin{array}{rcl} \frac{\text{Area}(R_s) - \text{Area}(R_b \bigcap R_s)}{\text{Area}(R_s)} & < & t_6 \quad \text{, or} \\ \frac{\text{Area}(R_s) - \text{Area}(R_b \bigcap R_s)}{\text{Area}(R_s)} & < & t_8 \quad \wedge \quad \frac{\text{Area}(R_s)}{\text{Area}(R_b)} \quad < \quad t_7, \end{array} \tag{A.14}$$

where R_b and R_s are the bigger and smaller rectangle, respectively, and t_6,t_7, and t_8 are fixed thresholds, $t_6 < t_8$. This guarantees that rectangles will be joined either if the non-overlapping part of the smaller rectangle is below threshold t_6, or if it is below some larger threshold t_8 but the difference in size of the two rectangles remains below another threshold, t_7.

Bibliography

[1] M.J. Aitkenhead and A.J.S. McDonald. A neural network based obstacle-navigation animat in a virtual environment. *Engineering Applications of Artificial Intelligence*, 15(3-4):229–239, 2002.

[2] J. S. Albus. *Brains, behavior, and robotics*. BYTE Books, Peterborough, N.H., 1981.

[3] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the 18th National Conference on Artificial Intelligence*, Edmonton, Alberta, July 2002.

[4] Ahmet Arslan and Mehmet Kaya. Determination of fuzzy logic membership functions using genetic algorithms. *Fuzzy Sets and Systems*, 118(2):297–306, 2001.

[5] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23(2/3):279–303, 1996.

[6] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1):11–73, 1997.

[7] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1):75–113, 1997.

[8] L. Baird and H. Klopf. Reinforcement learning with high-dimensional continuous actions. Technical Report WL-TR-93-1147, Wright Laboratory,Wright-Patterson Air Force Base, 1993.

[9] O. Basir, H. Zhu, and F. Karray. Fuzzy based image segmentation. In M Nachtegael, D. Van der Weken, D. Van De Ville, and E. E Kerre, editors, *Fuzzy Filters in Image Processing*, Studies in Fuzziness and Soft Computing, pages 101–128. Springer, 2003.

[10] D.F. Beal and M.C. Smith. Temporal difference learning for heuristic search and game playing. *Information Sciences*, 122(1):3–21, 2000.

[11] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[12] Gilles Brassard and Paul Bratley. *Fundamentals of algorithmics*. Prentice-Hall, Inc., 1996.

[13] Richard P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[14] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, 1992.

[15] H. D. Cheng and Jim Rong Chen. Automatically determine the membership function based on the maximum entropy principle. *Information Sciences*, 96(3-4):163–182, 1997.

[16] H. D. Cheng and H. Xu. A novel fuzzy logic approach to mammogram contrast enhancement. *Information Sciences*, 148(1-4):167–184, 2002.

[17] R. Coulom. Feedforward neural networks in reinforcement learning applied to high-dimensional motor control. In *13th International Conference on Algorithmic Learning Theory*, pages 402–413. Springer, 2002.

[18] Robert H. Crites and Andrew G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2-3):235–262, 1998.

[19] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[20] T. D'Orazio, G. Cicirelli, and A. Distante. Development of a vision-based behavior by reinforcement learning. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics: 'Human Communication and Cybernetics'*, volume 5, pages 453–457, Tokyo, Japan, 1999.

[21] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. Wiley, New York, NY, 2nd edition, November 2001.

[22] D. Ernst, M. Glavic, and L. Wehenkel. Power systems stability control: reinforcement learning framework. *Power Systems, IEEE Transactions on*, 19(1):427–435, 2004.

[23] D.E. Finkel. *DIRECT Optimization Algorithm User Guide*. Center for Research in Scientific Computation North Carolina State University, Raleigh, NC 27695-8205, March 2003.

[24] Manuel Guillermo Forero-Vargas. Fuzzy thresholding and histogram analysis. In M Nachtegael, D. Van der Weken, D. Van De Ville, and E. E Kerre, editors, *Fuzzy Filters in Image Processing*, Studies in Fuzziness and Soft Computing, pages 238–270. Springer, 2003.

[25] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[26] K. Fukunaga and R. R. Hayes. Effects of sample size in classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):873–885, 1989.

[27] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 189–192, Manchester, August 1988.

[28] I. Hossain, J. Liu, and J. You. Tropical cyclone pattern recognition for intensity and forecasting. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 6, pages 851–856, 1999.

[29] D.P. Huijsmans and N. Sebe. Extended performance graphs for cluster retrieval. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 26–31, 2001.

[30] Masaru Iida, Masanori Sugisaka, and Katsunari Shibata. Application of direct-vision-based reinforcement learning to a real mobile robot. In *Proceedings of the International Conference of Neural Information Processing Systems*, volume 5, pages 2556–2560, 2002.

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

[32] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[33] I. Kalaykov and G. Tolt. Real-time image noise cancellation based on fuzzy similarity. In M Nachtegael, D. Van der Weken, D. Van De Ville, and E. E Kerre, editors, *Fuzzy Filters in Image Processing*, Studies in Fuzziness and Soft Computing, pages 54–71. Springer, 2003.

[34] Willett Kempton. Interview methods for eliciting fuzzy categories. *Fuzzy Sets and Systems*, 14(1):43–64, 1984.

[35] J.W. Kim, B.M. Kim, and J.Y. Kim. Genetic algorithm simulation approach to determine membership functions of fuzzy traffic controller. *Electronics Letters*, 34(20):1982–1983, 1998.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[37] Y. Kobayashi, J. Ota, K. Inoue, and T. Arai. State and action space construction using vision information. In *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, volume 5, pages 447–452, 1999.

[38] Raghu Krishnapuram. Generation of membership functions via possibilistic clustering. In *Proccedings of the Third IEEE Conference on Fuzzy Systems*, volume 2, pages 902–908, Orlando, U.S.A., 1994.

[39] F. LeBourgeois. Robust multifont OCR system from gray level images. In *4th International Conference on Document Analysis and Recognition*, pages 1–5, 1997.

[40] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 08(3):293–321, 1992.

[41] Michael Littman, Richard Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

[42] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Series in Statistics. Springer, 2001.

[43] P. Liu. Representation of digital image by fuzzy neural network. *Fuzzy Sets and Systems*, 130(1):109–123, 2002.

[44] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1/2/3):159–195, 1996.

[45] B. Mak and E. Barnard. Phone clustering using the Bhattacharyya distance. In *Proc. ICSLP '96*, volume 4, pages 2005–2008, Philadelphia, PA, 1996.

[46] B.S. Manjunath and W.Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, 1996.

[47] R.A. McCallum. Hidden state and reinforcement learning with instance-based state. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 26(3):464–473, 1996.

[48] K. R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12:181–201, March 2001.

[49] M. Negnevitsky. *Artificial intelligence : a guide to intelligent systems*. Addison-Wesley, Harlow, England ; New York, 2002.

[50] John O'Doherty, Peter Dayan, Johannes Schultz, Ralf Deichmann, Karl Friston, and Raymond J. Dolan. Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science*, 304(5669):452–454, 2004.

[51] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66, 1979.

[52] M. Ouslim and K.M. Curtis. Machine parts classification based on a digital neural network. In *Proceedings of the 3rd IEEE International Conference on Electronics, Circuits, and Systems*, pages 643–646, 1996.

[53] S. K. Pal. Fuzziness, image information and scene analysis. In Ronald R. Yager and Lotfi Asker Zadeh, editors, *An Introduction to fuzzy logic applications in intelligent systems*, The Kluwer International series in engineering and computer science. Knowledge representation, learning and expert systems ; SECS 165, pages 147–184. Kluwer Academic, Boston, 1992.

[54] S. K. Pal and N. K. Pal. Entropy: a new definition and its applications. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5), 1991.

[55] S. K. Pal and N. K. Pal. Higher order fuzzy entropy and hybrid entropy of a set. *Information Sciences*, 61(3):211–231, 1992.

[56] R. Patrascu and D. Stacey. Adaptive exploration in reinforcement learning. In *International Joint Conference on Neural Networks*, volume 4, pages 2276–2281, 1999.

[57] Relu-Eugen Patrascu. Adaptive exploration in reinforcement learning. Master's thesis, University of Guelph, 1998.

[58] Witold Pedrycz and George Vukovich. On elicitation of membership functions. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 32(6):761–767, 2002.

[59] S. Peng and L. Lucke. Multi-level adaptive fuzzy filter for mixed noise removal. In *Proceedings of the 1995 IEEE International Symposium on Circuits and Systems-ISCAS 95. Part 2 (of 3), Apr 30-May 3 1995*, volume 2, pages 1524–1527, Seattle, WA, USA, 1995. IEEE, Piscataway, NJ, USA.

[60] C. D. Perttunen, D. R. Jones, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, October 1993.

[61] C. Ribeiro. Reinforcement learning agents. *Artificial Intelligence Review*, 17(3):223–250, 2002.

[62] M. Ricordeau. Q-concept-learning: generalization with concept lattice representation in reinforcement learning. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 316–323, 2003.

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

[64] Stuart J. Russell and Andrew L. Zimdars. Q-decomposition for reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning*, Washington, D.C., August 2003.

[65] B. Sallans and G. E. Hinton. Using free energies to represent Q-values in a multiagent reinforcement learning task. In *Proceedings of Neural Information Processing Systems*, pages 1075–1081, 2000.

[66] Giampiero Salvi. Accent clustering in swedish using the bhattcharyya distance. In *15th International Congress of Phonetic Science*, pages 1149–1152, Barcelona, Spain, 2003.

[67] A. Sancho-Royo and J. L. Verdegay. Methods for the Construction of Membership Functions. *International Journal of Intelligent Systems*, 14(12):1213–1230, 1999.

[68] Katsunari Shibata and Masaru Iida. Acquisition of box pushing by direct-vision-based reinforcement learning. In *Proceedings of the Society of Instrument and Control Engineers Annual Conference*, 2003.

[69] Katsunari Shibata, Masanori Sugisaka, and Koji Ito. Fast and stable learning in direct-vision-based reinforcement learning. In *Proceedings of the 6th International Symposium on Artificial Life and Robotics*, volume 1, pages 200–2003, 2001.

[70] M. Shokri and H. R. Tizhoosh. Using reinforcement learning for image thresholding. In *CCECE 2003-CCGEI 2003*, Montréal, 2003.

[71] B. W. Silverman. Using kernel density estimates to investigate multimodality. *Journal of the Royal Statistical Society. Series B (Methodological)*, 43(1):97–99, 1981.

[72] Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, Banff, Alberta, July 2004. To appear.

[73] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *International Conference on Machine Learning*, pages 284–292, 1994.

[74] William D. Smart and Leslie Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA-2002)*, volume 4, pages 3404–3410, 2002.

[75] Kenneth O. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002.

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

[77] R. 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.

[78] R. S. Sutton. University of Alberta, personal communication, 2004.

[79] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass., 1998.

[80] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[81] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of NIPS-12*, pages 1057–1063. MIT Press, 2000.

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

[83] H. R. Tizhoosh. *Fuzzy Image Processing*. Springer-Verlag, 1997.

[84] H. R. Tizhoosh and G. W. Taylor. Reinforced contrast adaptation. *International Journal of Image and Graphics*, 2003. Submitted.

[85] Hamid R. Tizhoosh. Observer-dependent image enhancement. In M Nachtegael, D. Van der Weken, D. Van De Ville, and E. E Kerre, editors, *Fuzzy Filters in Image Processing*, Studies in Fuzziness and Soft Computing, pages 238–270. Springer, 2003.

[86] L. H. Tsoukalas and R. E. Uhrig. *Fuzzy and neural approaches in engineering*. Adaptive and learning systems for signal processing, communications, and control. Wiley, New York ; Chichester, England, 1997.

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

[88] Christopher J.C.H. Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 08(3):279–292, 1992.

[89] Steven D. Whitehead and Long-Ji Lin. Reinforcement learning of non-Markov decision processes. *Artificial Intelligence*, 73(1-2):271–306, 1995.

[90] C. Wolf. Content based image retrieval using interest points and texture features. Master's thesis, Vienna Technical University, 2000.

[91] C. Wolf. *Text Detection in Images taken from Video Sequences for Semantic Indexing*. PhD thesis, INSA de Lyon, 2003.

[92] C. Wolf and J. M. Jolion. Extraction and recognition of artificial text in multimedia documents. *Pattern Analysis and Applications*, 6(4):309–326, February 2004.

[93] C. Wolf and J. M. Jolion. Precision-recall graphs for the evaluation of object detection algorithms. 2004. Work in progress.

[94] P. Y. Yin. Maximum entropy-based optimal threshold selection using deterministic reinforcement learning with controlled randomization. *Signal Processing*, 82(7):993–1006, 2002.

[95] Lotfi A. Zadeh. Probability measures of fuzzy events. *Journal of Mathematical Analysis and Applications*, 23:421–427, 1968.