

To the Graduate Council:

I am submitting herewith a thesis written by Christopher Allen Niedzwiedz entitled "Vision-Based Reinforcement Learning Using A Consolidated Actor-Critic Model" I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Engineering.

Itamar Arel, Major Professor

We have read this thesis and  
recommend its acceptance:

Gregory Peterson

---

Hairong Qi

---

Acceptance for the Council:

Carolyn R. Hodges

---

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

# Vision-Based Reinforcement Learning Using A Consolidated Actor-Critic Model

A Thesis  
Presented for the  
Master of Science  
Degree  
The University of Tennessee, Knoxville

Christopher Allen Niedzwiedz  
December 2009

Copyright © 2009 by Christopher Allen Niedzwiedz  
All rights reserved.

# Dedication

This thesis is dedicated to my parents, Frank and Teddi Niedzwiedz. For your unwavering encouragement, support, and emphasis on education, I thank you.

# Acknowledgments

Several people exist whom I would like to acknowledge for their help and support.

First and foremost, I would like to acknowledge Dr. Itamar Arel for his constant support and patience. If it weren't for his guidance and tutelage, I would not be where I am today.

Additionally I would like to thank Dr. Gregory Peterson and Dr. Hairong Qi. If not for their guidance in both my undergraduate and graduate level work, my education would have not been the fulfilling experience it has been.

Also, I would like to thanks the Machine Intelligence Laboratory, Bobby Coop, Scott Livingston, and Everett Stiles. Their assistance with my work has been a great help over the years.

Finally, I would like to thank my family and friends for their love, support, and understanding.

# Abstract

Vision-based machine learning agents are tasked with making decisions based on high-dimensional, noisy input, placing a heavy load on available resources. Moreover, observations typically provide only partial information with respect to the environment state, necessitating robust state inference by the agent. Reinforcement learning provides a framework for decision making with the goal of maximizing long-term reward. This thesis introduces a novel approach to vision-based reinforcement learning through the use of a consolidated actor-critic model (CACM). The approach takes advantage of artificial neural networks as non-linear function approximators and the reduced computational requirements of the CACM scheme to yield a scalable vision-based control system. In this thesis, a comparison between the actor-critic and CACM is made. Additionally, the affect of observation prediction and correlated exploration has on the agent's performance is investigated.

# Contents

|          |                                                          |           |
|----------|----------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction . . . . .</b>                            | <b>1</b>  |
| 1.1      | Vision-Based Machine Learning . . . . .                  | 1         |
| 1.2      | Reinforcement Learning Agents . . . . .                  | 2         |
| 1.3      | Motivation . . . . .                                     | 3         |
| 1.4      | Thesis Outline . . . . .                                 | 4         |
| <b>2</b> | <b>Literature Review . . . . .</b>                       | <b>6</b>  |
| 2.1      | Partially Observable Markov Decision Processes . . . . . | 6         |
| 2.2      | Artificial Neural Networks . . . . .                     | 7         |
| 2.2.1    | Feed-forward ANNs . . . . .                              | 9         |
| 2.2.2    | Recurrent ANNs . . . . .                                 | 9         |
| 2.3      | Reinforcement Learning . . . . .                         | 10        |
| 2.3.1    | Watkins Q-Learning . . . . .                             | 10        |
| 2.3.2    | The Actor-Critic Model . . . . .                         | 11        |
| 2.3.3    | The Consolidated Actor-Critic Model . . . . .            | 15        |
| 2.4      | Stochastic Meta-Descent . . . . .                        | 18        |
| <b>3</b> | <b>Design Approach . . . . .</b>                         | <b>20</b> |
| 3.1      | Vision-Based Maze Navigation . . . . .                   | 20        |
| 3.2      | Design of the Machine Learning Agent . . . . .           | 22        |
| 3.2.1    | Feature Extraction . . . . .                             | 22        |
| 3.2.2    | Correlated Exploration . . . . .                         | 22        |
| 3.3      | Simulation Descriptions . . . . .                        | 25        |
| 3.4      | Simulation Results . . . . .                             | 27        |
| 3.4.1    | Vision-Based Navigation with the CACM . . . . .          | 27        |

|          |                                                |           |
|----------|------------------------------------------------|-----------|
| 3.4.2    | Performance Comparison . . . . .               | 27        |
| 3.4.3    | Impact of Action Set . . . . .                 | 29        |
| 3.4.4    | Correlated Exploration . . . . .               | 31        |
| 3.4.5    | The Impact of Observation Prediction . . . . . | 33        |
| <b>4</b> | <b>Conclusions . . . . .</b>                   | <b>38</b> |
| 4.1      | Thesis Summary . . . . .                       | 38        |
| 4.2      | Future Work . . . . .                          | 39        |
| 4.3      | Relevant Publications . . . . .                | 39        |
|          | <b>Bibliography . . . . .</b>                  | <b>41</b> |
|          | <b>Vita . . . . .</b>                          | <b>45</b> |

# List of Figures

|      |                                                                        |    |
|------|------------------------------------------------------------------------|----|
| 2-1  | A Simple Markov Chain . . . . .                                        | 6  |
| 2-2  | A Simple Artificial Neuron . . . . .                                   | 8  |
| 2-3  | Example Feed-forward Neural Network . . . . .                          | 9  |
| 2-4  | Example Elman Neural Network . . . . .                                 | 11 |
| 2-5  | An Actor-Critic Model for Reinforcement Learning . . . . .             | 12 |
| 2-6  | A Consolidated Actor-Critic Model . . . . .                            | 16 |
| 3-1  | Sony AIBO in a Maze Environment . . . . .                              | 21 |
| 3-2  | Block Diagram of the RL Agent and its Environment . . . . .            | 23 |
| 3-3  | Example observation converted to its feature array . . . . .           | 23 |
| 3-4  | Two-state model for bursty exploration . . . . .                       | 24 |
| 3-5  | Maze A . . . . .                                                       | 26 |
| 3-6  | Maze B . . . . .                                                       | 26 |
| 3-7  | CACM Q MSE for maze A . . . . .                                        | 28 |
| 3-8  | CACM Duration vs. Time Step for maze A . . . . .                       | 28 |
| 3-9  | CACM vs. Actor-Critic Bellman MSE . . . . .                            | 30 |
| 3-10 | CACM vs. Actor-Critic Action MSE . . . . .                             | 30 |
| 3-11 | CACM vs. Actor-Critic Episode Duration . . . . .                       | 31 |
| 3-12 | Bellman Error for Differing Action Sets . . . . .                      | 32 |
| 3-13 | Duration per Episode for Differing Action Sets . . . . .               | 32 |
| 3-14 | Distribution for Correlated Exploration . . . . .                      | 34 |
| 3-15 | Distribution for the Random Walk . . . . .                             | 35 |
| 3-16 | Bellman Error for Correlated Exploration . . . . .                     | 36 |
| 3-17 | Episode Durations for Correlated and Random Walk Exploration . . . . . | 36 |
| 3-18 | Bellman Error for Observation Prediction . . . . .                     | 37 |

|                                                                |    |
|----------------------------------------------------------------|----|
| 3-19 Duration per Episode for Observation Prediction . . . . . | 37 |
|----------------------------------------------------------------|----|

# Chapter 1

## Introduction

### 1.1 Vision-Based Machine Learning

Real-world decision making is often based on the state of the surrounding environment. One of the most efficient ways for humans to gather data on their surroundings is through vision. Vision conveys large amounts of information very quickly (at the speed of light). It is useful, therefore, for machine-learning agents to make use of the large amount of information available in the visible spectrum.

As the power and speed of computing has grown, it has become possible for machine-learning agents to perform real-time vision-based tasks. These tasks include classifying objects, navigating large terrain, tracking items of interest and face recognition. Previously, such computation was not feasible due to memory and computational constraints. Extracting information from the high-dimensional input places a heavy load on the agent.

It is the high-dimensionality and uncertainty of visual input that makes this task so difficult. To keep the input small, image resolution can be kept small. This lower resolution, in addition to poor lighting, glare and other factors, introduces noise to the visual input. Further, an agent may need to base judgements on objects that are rotated, translated, or occluded in its field of view. The agent must then make decisions based on this noisy, partial data.

Applying real-time artificial agents to robotic systems introduces new challenges. These systems can be automated vehicles, arms on an assembly line, or nearly any other way robotics is used in the modern world. Uncertainty is introduced by the sensors and actuators. Additionally, it is

difficult to track the absolute positioning of the agent due to cumulative sensor error [1]. Tasks in real-world situations often require continuous, instead of discrete, inputs since many problems cannot be logically divided in this way. A robot must deal with the uncertainty of its motion and sensing in real-time.

To be effective, an agent must be robust enough to overcome environmental uncertainty to accomplish its task. Such agents require a tolerance of noisy inputs, imprecise actuation, and adaptability to a changing environment.

## 1.2 Reinforcement Learning Agents

Reinforcement learning (RL) differs from other machine learning disciplines, such as supervised and unsupervised learning, in that it attempts to solve the credit assignment problem with a non-specific reward signal produced by the environment. This is in contrast to supervised learning where the agent is provided with the exact error between its current output and the expected output. Unsupervised learning methods provide no signal and the agent must organize the data itself. The goal of an RL agent is to maximize the total long-term reward,  $r$ , received from the environment. This can be expressed as

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \end{aligned} \tag{1.1}$$

where  $0 < \gamma < 1$  is a discount factor.

It is the non-specific reward signal that lends itself to the flexibility of RL agents. This signal is set up by the experimenter to provide proper reinforcement to the agent. A positive reward is often provided for achieving the goal of the trial and negative reward if the agent chooses actions that are not conducive to the task. Once the reward function is crafted, the agent is to determine the actions necessary to maximize the function.

The reward helps the agent craft a *value function*. This value function is a measure of the long-term expected reward of a particular state. Using this value function, the agent will form a policy for a given task. A policy is a mapping of states to actions, ideally intended to provide the maximum return.

Further, RL agents do not require an external entity to guide them as does a supervised learning agent. In classification problems, supervised learning agents are presented with a set of samples and a set of labels with which to classify this data. When the agent misclassifies a sample, it is presented with the correct label. This is not so in reinforcement learning. The only feedback an agent would receive is a non-specific “correct” or “incorrect” signal. This robustness inherent in reinforcement learning makes it an attractive field of study for difficult problems.

The actor-critic model is a prominent paradigm for neural network-based RL agents. The model is comprised of two networks: one actor to approximate the policy of the agent, and one critic to learn the value function. The actions decided upon from the actor network are fed into the critic. Once a reward is received from the environment, the signal is propagated through the critic and actor networks respectively.

The actor-critic model, however, requires duplicating computation in that both networks need to converge on a model of the environment before the optimal policy is reached. This introduces a redundancy in the computational modeling of these two networks as both must form similar models of the same environment. The consolidated actor-critic model (CACM) combines the two networks into one, eliminating the computational redundancy while improving overall performance [2].

### 1.3 Motivation

Previous work on vision-based reinforcement learning has involved stochastic models for state transitions to account for inaccuracy in both the sensors and actuators [1] [3]. The state sets for these problems are often fixed and aim to model the real-world likelihood of transition from state to state. Reinforcement learning shifts the burden from forming explicit models of the environment and allows the agent to form its own. The burden is then placed on crafting the agent’s value function, which is a non-trivial task in many cases.

Tabular methods of reinforcement learning, including forms of temporal difference (TD) learning, are ill suited for the high-dimensionality of these problems. These methods suffer from the *curse of dimensionality*, where the memory and computational requirements for these methods grow exponentially with each added input. Further, traditional methods of reinforcement learning impose a finite state set posed as a Markov Decision Process. This model does not lend itself well to the continuous nature of real-world problems.

Neural networks provide a method of approximating the tabular methods of reinforcement learning. Not only are they capable of approximating high-dimensional, non-linear functions, but they are tolerant of noisy inputs. They can be used to model an agent’s value function, policy, or both. Neural networks have been used to approximate and expand upon existing RL methods [4]. While it has been shown that application of function approximation to reinforcement learning can lead to divergence [5], the method has still enjoyed success in the field, especially with the game of backgammon [6]. Another exciting success was the training of a helicopter to fly upside down [7] using dynamic programming methods. In this case, heavy simulation was required before this was taken to the real-world trials.

The consolidated actor-critic (CACM) is a computationally efficient approach to reinforcement learning with neural networks. It provides the flexibility of neural networks and the power of reinforcement learning methods like the actor-critic model while lessening the computational requirements.

This thesis takes a novel approach to vision-based reinforcement learning through the use of the consolidated actor-critic model. The end goal of this work is the continuous operation of a robotic agent in real-world problems. The CACM takes advantage of the approximation abilities of neural networks and the power of reinforcement learning techniques. It also provides the added benefit of lower computational requirements.

## 1.4 Thesis Outline

Chapter 2 covers the background literature upon which this work is based. It begins with an overview of reinforcement learning as a machine learning discipline. This is followed by descriptions of the actor-critic model, CACM, and modifications to the CACM as used in this thesis. Additionally, a brief summary of machine vision techniques is provided.

Chapter 3 describes the experimental setup for the vision-based learning task. It details the different trials, beginning with simple bit vectors to the processing of images to simulate an actual robotic system. The constraints and assumptions of the simulations are enumerated and explained. Results of each simulation is provided. The performance of the CACM simulations are also contrasted with that of the actor-critic model.

Chapter 4 explains future avenues of research on this topic. This includes a discussion on

extending this work to a live robotic system and the challenges involved. In addition, publications resulting from this work are provided.

## Chapter 2

# Literature Review

### 2.1 Partially Observable Markov Decision Processes

Markov Decision Processes (MDPs) are mathematical models that allow for the analysis of problems where state transitions are partially random and partially controlled by the agent [8]. MDPs play an important part in dynamic programming, one of the disciplines on which reinforcement learning is based. A simple Markov chain is depicted in Figure 2-1. Each transition from state  $i$  to state  $j$  occurs with a probability  $0 < \lambda_{ij} < 1$ .

An MDP can be stated as a 4-tuple  $(S, A, P, R)$  where  $S$  is the set of states,  $A$  the set of actions,  $P$  the set of transition probabilities, and  $R$  the reward received after transition to a state. At each time step, an agent takes an action based on a policy  $\pi$ . The policy is a mapping the set of states to the set of actions  $\pi : S \rightarrow A$ .

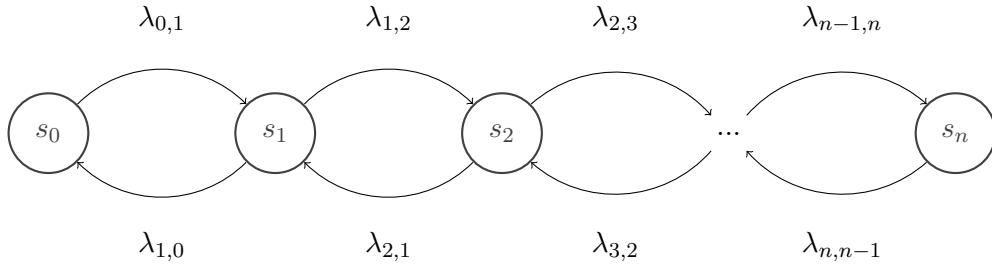


Figure 2-1: A Simple Markov Chain

It is common for reinforcement learning problems to phrase the task at hand as an MDP in that the transition to the next state and the reward received for this transition depend only on the current state and action. It is for this reason that MDPs are considered to be memoryless. Expressed mathematically, this is:

$$\begin{aligned} Pr\{s_t = s', r_{t+1} = r | s_t, a_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\} \\ = Pr_{s_{t+1} = s', r_{t+1} = r | s_t, a_t} \end{aligned} \quad (2.1)$$

where  $a_t$  is the action taken at time  $t$ ,  $r_t$  the reward received, and  $s_t$  the state at time  $t$ .

In fully observable MDPs, complete state information is available to the agent. In real world problems, however, such information is not always available. Agents rely on partial knowledge of the environment, provided through observations. Partially Observable Markov Decision Processes (POMDPs) are a generalization of the MDP. In POMDPs, the underlying system is assumed to be an MDP, but the agent is only able to make observations of a state. The agent must then impose a probability distribution over potential states and use this as input to the original problem [9]. POMDPs have found application in robot navigation [10], visual tracking [11] and medical applications [12].

POMDPs can be expressed as a 5-tuple  $(S, A, O, P, R)$  where  $S$  is the set of states,  $A$  the set of actions,  $O$  the set of observations,  $P$  the set of probabilities, and  $R$  is the reward function, mapping the state-action pair to a specific reward value. The policy,  $\pi$  of the POMDP is a mapping of the observation set to the action set  $\pi : O \rightarrow A$ . The reward for taking action  $a_t$  given observations  $o_t$  is expressed as

$$r_{t+1}(o_t, a_t) = \sum_s Pr\{s | o_t\} r_{t+1}(s, a_t). \quad (2.2)$$

where  $r_{t+1}$  is the reward at time  $t + 1$ , and  $s$  a state in state set  $S$ .

## 2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are biologically-inspired mathematical tools consisting of a set of artificial neurons. Each neuron performs a simple computation on its inputs and the result is combined with that of the other neurons to produce the output of the network as a whole.

Intended to reflect the structure and organization of its biological networks of neurons, the first

artificial neuron was proposed by McCulloch and Pitts in 1943 [13]. While simple in comparison to biological neurons, artificial neurons have proven to be powerful computation devices. Figure 2-2 depicts a simple artificial neuron. The output,  $y$ , of such a neuron is given to be

$$y = f\left(\sum_{i \in I} w_i * i_i\right) \quad (2.3)$$

where  $f(x)$  is the neuron's activation function,  $i_i$  is an input in  $i \in I$  and  $w_i$  is the weight of the input,  $i$ .

From the simple perceptron in 1958 [14], ANNs have carved a prominent position as function approximators. A modern neural network is comprised of interconnected discrete units, or neurons, organized in multiple layers. Each neuron receives input from the previous layer and outputs to the next layer.

The output of a multilayer network can be expressed as

$$y = W_{oh} * f(W_{hi} * i) \quad (2.4)$$

where  $y$  is the output,  $W_{oh}$  is the matrix of weights between the output and hidden layer,  $f$  is a nonlinear activation function,  $W_{hi}$  is the matrix of weights between the hidden and input layer, and  $i$  is the vector of input values. One of the most common activation functions is the sigmoid, given as

$$f(x) = \frac{1}{1 + e^{-x}}. \quad (2.5)$$

ANNs come in two general types: feed-forward and recurrent. Feed-forward networks keep no internal state, other than updated weight values. Recurrent networks have feedback neurons acting

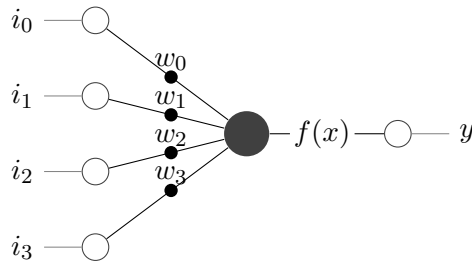


Figure 2-2: A Simple Artificial Neuron

as a delay slot to provide context to the next set of inputs. The next two sections elaborate on these architectures.

### 2.2.1 Feed-forward ANNs

The feed-forward network is one of the simplest designs of an ANN. Depicted in Figure 2-3, the feed-forward network consists of an input layer, a hidden layer, and an output layer. Each layer feeds its output as the input to the next. The connections between the neurons are weighted. As a network is trained, the weights between neurons are updated by an error signal that is then propagated through the network in reverse. This process is known as *backpropagation*. Training neural networks will be discussed with respect to the actor-critic model of reinforcement learning in section 2.3.

### 2.2.2 Recurrent ANNs

Recurrent neural networks (RNNs) are feed-forward networks where the output of the hidden nodes feed back as inputs during the next time step. This gives the ANN the ability to maintain state, providing memory. This memory is required when approximating time-dependent and periodic functions. A common sinusoid is an example of a function that memoryless approaches are unable to approximate. Since the values of the sinusoid repeat for different input values, context must be maintained as to the previous output so as to output correctly.

### Elman Networks

Elman networks are the simplest multilayer RNNs. The output of the hidden layer neurons are not only fed to the output neurons, but also to context neurons that act as a unit delay between time

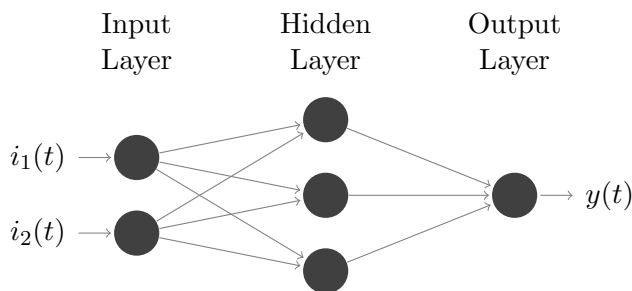


Figure 2-3: Example Feed-forward Neural Network

steps. These context neurons then feed as inputs to the hidden layer as inputs during the next time step [15]. Depicted in Figure 2-4, these networks are capable of learning sequential data due to the recurrent connections in the network.

## 2.3 Reinforcement Learning

Reinforcement learning (RL), as a machine learning discipline [9], has received significant attention from both academia and industry in recent years. What differentiates RL from other machine learning methods is that it aims to solve the *credit assignment problem*, in which an agent is charged with evaluating the long-term impact of each action taken. In doing so, an agent which interacts with an environment attempts to maximize a value function, based only on inputs representing the environment's state and nonspecific reward signal. The agent constructs an estimated value function that expresses the expected return from taking a specific action at a given state.

Temporal difference (TD) learning methods in reinforcement learning, such as Q-Learning [16] and SARSA [4] [17], which employ tables to represent the state or state-action values are practical for low-dimensional problems. They prove ineffective as new state variables are introduced, however, as each variable increases the state space exponentially, increasing the amount of system memory and processing power required. Function approximators, such as ANNs, have been employed to overcome this limitation.

### 2.3.1 Watkins Q-Learning

Watkins Q-learning is an off-policy temporal difference method for learning from delayed reinforcement [16]. Off-policy algorithms permit an agent to explore while also finding the deterministic optimal policy. The policy the agent uses is more randomized to permit exploration of the state space while disallowing exploration to affect the final policy. In contrast, on-policy algorithms are those in which the agent always explores and attempts to find the optimal policy that permits it to continue to explore. The exploration must be considered as part of the policy.

In Q-learning, an agent learns the action-value function that yields the maximum expected

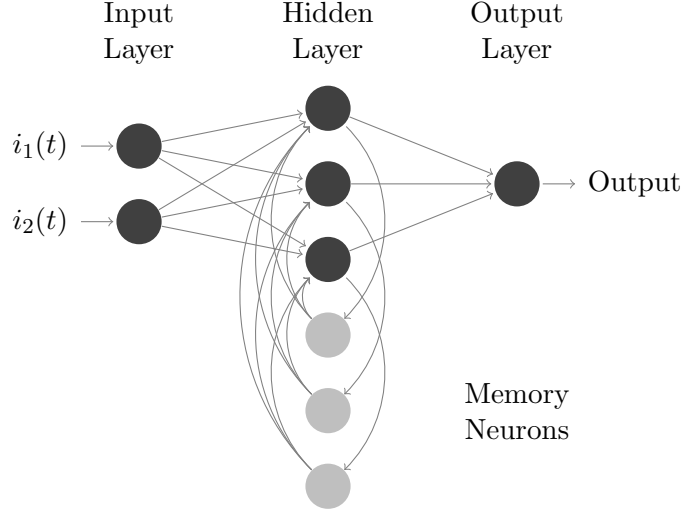


Figure 2-4: Example Elman Neural Network

return. The one step Q-learning update rule is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]. \quad (2.6)$$

where  $Q(s_t, a_t)$  is the value of a particular state,  $s$ , and action,  $a$ ,  $\alpha$  is the learning rate,  $r_{t+1}$  the reward and  $\gamma$  the discount factor. In this case, the learned action-value function,  $Q$ , directly approximates the optimal action-value function, independent of the policy being followed. Q-learning has been proven to converge faster than SARSA. [9]

### 2.3.2 The Actor-Critic Model

The actor-critic model, depicted in Figure 2-5, is comprised of two feed-forward networks. In the general case, the agent is assumed to have no a-priori knowledge of the environment. Both the actor and critic networks must form their own internal representation of the environment, based on interactions with it and the reward received at each step [18]. As in other reinforcement learning methods, the actor-critic model attempts to maximize the discounted expected return,  $R(t)$ , restated from Chapter 1 as

$$\begin{aligned} R(t) &= r(t+1) + \gamma r(t+2) + \dots \\ &= \sum_{k=1}^{\infty} \gamma^{k-1} r(t+k), \end{aligned} \quad (2.7)$$

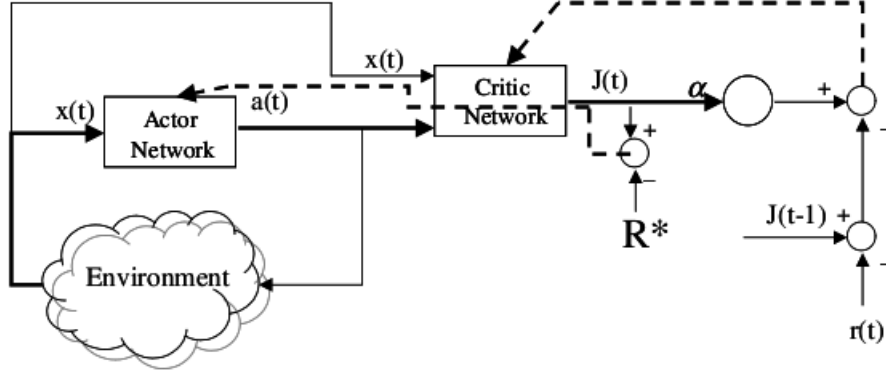


Figure 2-5: An Actor-Critic Model for Reinforcement Learning

where  $r(t)$  denotes the reward received from the environment at time  $t$  and  $\gamma$  is the discount rate. The critic network is responsible for approximating this value, represented as  $J(t)$ . The critic network aims to minimize the overall error defined as

$$E_c(t) = \frac{1}{2} e_c^2(t), \quad (2.8)$$

where  $e_c(t)$  is the standard Bellman error [18],

$$e_c(t) = [r(t) + \alpha J(t)] - J(t-1). \quad (2.9)$$

The weight update rule for the critic network is gradient based. Let  $w_c$  be the set of weights in the critic network, the value of  $w_c$  at time  $t+1$  is

$$w_c(t+1) = w_c(t) + \Delta w_c(t). \quad (2.10)$$

The weights are updated as

$$\Delta w_c(t) = l_c(t) \left[ \frac{-\partial E_c(t)}{\partial w_c(t)} \right], \quad (2.11)$$

$$\frac{\partial E_c(t)}{\partial w_c(t)} = \frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_c(t)}. \quad (2.12)$$

Similarly, the goal of the actor network is to minimize the term

$$\begin{aligned} E_a(t) &= \frac{1}{2} e_a^2(t), \\ e_a(t) &= J(t) - R^*. \end{aligned} \tag{2.13}$$

where  $R^*$  denotes the optimal return. Once again, weight updates are based on gradient-descent techniques and, thus, we have

$$\begin{aligned} w_a(t+1) &= w_a(t) + \Delta w_a(t), \\ \Delta w_a(t) &= l_a(t) \left[ \frac{-\partial E_a(t)}{\partial w_a(t)} \right], \\ \frac{\partial E_a(t)}{\partial w_a(t)} &= \frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_a(t)}, \end{aligned} \tag{2.14}$$

where  $l_a(t)$  is the learning parameter or step size of the actor network update rule. An online learning algorithm can now be derived from the previous equations. Starting with the critic network output, we have

$$J(t) = \sum_{i=1}^{N_{hc}} w_{ci}^{(2)}(t) p_i(t) \tag{2.15}$$

where  $N_{hc}$  is the number of hidden nodes for the critic network, and  $p_i(t)$  is the output of node  $i$  given as

$$\begin{aligned} p_i(t) &= \frac{1 - e^{-q_i(t)}}{1 + e^{-q_i(t)}}, i = 1, \dots, N_{hc}, \\ q_i(t) &= \sum_{j=1}^n w_{cij}^{(1)}(t) x_j(t), i = 1, \dots, N_{hc} \end{aligned} \tag{2.16}$$

where  $q_i(t)$  is the input to hidden node  $i$  at time  $t$ . Applying the chain rule to (2.12) and substituting into (2.11) yields

$$\Delta w_{ci}^{(2)} = l_c(t) [e_c(t) p_i(t)] \tag{2.17}$$

for the output layer to the hidden layer nodes. Another expansion of (2.12) gives us

$$\begin{aligned}
\frac{\partial E_c(t)}{\partial w_c(t)} &= \frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_c(t)}, \\
&= \frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial p_i(t)} \frac{\partial p_i(t)}{\partial q_i(t)} \frac{\partial q_i(t)}{\partial w_{cij}^{(1)}(t)} \partial w_{cij}^{(1)}(t) \\
&= e_c(t) w_{ci}^{(2)}(t) \left[ \frac{1}{2} (1 - p_i^2(t)) \right] x_j(t).
\end{aligned} \tag{2.18}$$

The actor network update rule is calculated similarly, as follows

$$\begin{aligned}
a_i(t) &= \frac{1 - e^{-v_i(t)}}{1 + e^{-v_i(t)}}, i = 1, \dots, N_{ha}, \\
v_i(t) &= \sum_{j=1}^n w_{aij}^{(1)}(t) g_j(t), i = 1, \dots, N_{ha}, \\
g_i(t) &= \frac{1 - e^{-h_i(t)}}{1 + e^{-h_i(t)}}, i = 1, \dots, N_{ha}, \\
h_i(t) &= \sum_{j=1}^n w_{aij}^{(1)}(t) x_j(t), i = 1, \dots, N_{ha},
\end{aligned} \tag{2.19}$$

where  $v$  is the input to the actor node,  $g_i$  and  $h_i$  are the output and input of the hidden nodes of the actor network respectively, and  $a_t(t)$  is the action output. Back-propagating from the output to the hidden layer yields

$$\begin{aligned}
\Delta w_a^{(2)}(t) &= l_a(t) \left[ \frac{-\partial E_a(t)}{\partial w_{a_i}^{(2)}(t)} \right], \\
\frac{\partial E_a(t)}{\partial w_{a_i}^{(2)}(t)} &= \frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_{a_i}^{(2)}(t)}, \\
&= \frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial a_i(t)} \frac{\partial a_i(t)}{\partial v_i(t)} \frac{\partial v_i(t)}{\partial w_{a_i}^{(2)}(t)} \\
&= e_a(t) \sum_{i=1}^{N_{hc}} \left[ w_{ci}^{(2)}(t) \frac{1}{2} (1 - p_i^2(t)) w_{ci,n+1}^{(1)}(t) \right] \cdot \left[ \frac{1}{2} (1 - u^2(t)) \right] g_i(t).
\end{aligned} \tag{2.20}$$

From the hidden layer to the input,

$$\begin{aligned}
\Delta w_{a_{ij}}^{(1)}(t) &= l_a(t) \left[ \frac{-\partial E_a(t)}{\partial w_{a_{ij}}^{(1)}(t)} \right], \\
\frac{\partial E_a(t)}{\partial w_{a_{ij}}^{(1)}(t)} &= \frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial a_i(t)} \frac{\partial a_i(t)}{\partial v_i(t)} \frac{\partial v_i(t)}{\partial g_i(t)} \frac{\partial g_i(t)}{\partial h_i(t)} \frac{\partial h_i(t)}{\partial w_{a_{ij}}^{(1)}(t)}
\end{aligned}$$

$$\begin{aligned}
&= e_a(t) \sum_{i=1}^{N_{hc}} \left[ w_{ci}^{(2)}(t) \frac{1}{2} (1 - p_i^2(t)) w_{ci,n+1}^{(1)}(t) \right] \cdot \\
&\quad \left[ \frac{1}{2} (1 - u^2(t)) \right] w_{ai}^{(2)}(t) \cdot \\
&\quad \left[ \frac{1}{2} (1 - g_i^2(t)) \right] x_j(t).
\end{aligned} \tag{2.21}$$

Actor-critic architectures have been studied as early as 1977 with classic problems such as the  $n$ -armed bandit problem [19]. The drawback of such an architecture is that it requires two systems to form models of the environment independent of one another. In the next section, the consolidated actor-critic is discussed to overcome this problem.

### 2.3.3 The Consolidated Actor-Critic Model

The training of both networks in the traditional actor-critic model results in duplicated effort between the actor and critic, since both have to form internal models of the environment, independently. Combining the networks into a single network would offer the potential to remove such redundancy. The consolidated actor-critic network (CACM) produces both the state-action value estimates of the critic as well as the policy of the actor using a single neural network. Moreover, the architecture offers improved convergence properties and more efficient utilization of resources [2]. Since this model is so critical to this thesis, a brief description is provided here.

Figure 2-6 illustrates the CACM architecture. The network takes a state  $s_t$  and an action  $a_t$  as inputs at time  $t$  and produces a state-action value estimate  $J_t$  and action  $a_{t+1}$  to be taken at the next time step. The latter is applied to the environment and fed back to the network at the subsequent time step. The temporal difference error signal is defined by the standard Bellman error, in an identical way to that followed by the regular actor-critic model (2.9). Additionally, the action error is identical to that given in (2.13) for the actor network. The weight update algorithm for the CACM is gradient-based given by

$$\begin{aligned}
E(t) &= E_c(t) + \frac{\partial E_a(t)}{\partial a(t)} \\
w(t+1) &= w(t) + \Delta w(t) \\
\Delta w(t) &= l(t) \left[ -\frac{\partial E(t)}{\partial w(t)} \right]
\end{aligned} \tag{2.22}$$

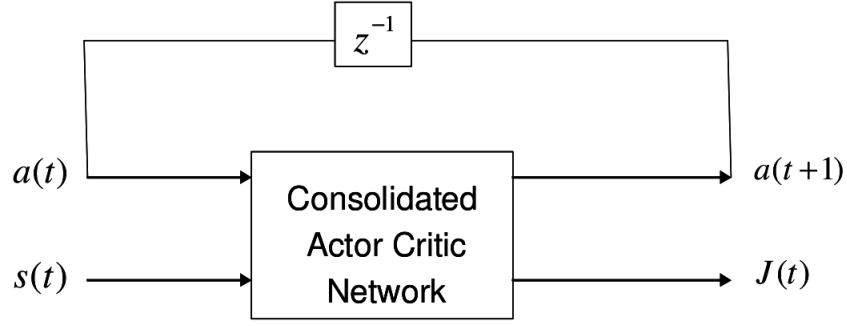


Figure 2-6: A Consolidated Actor-Critic Model

where  $l(t) > 0$  is the learning rate of the network at time  $t$ . The output  $J(t)$  of the CACM is given by (2.15). The action output,  $a(t+1)$  is of the form

$$a(t+1) = \sum_{j=1}^n w_{ai}^{(2)}(t) y_i(t) \quad (2.23)$$

where  $w_{ai}$  represents the weights between the  $i^{th}$  node in the hidden layer and the actor output node. Finally, we obtain  $y_i(t)$  in a similar way to that expressed in (2.16).

To derive the back-propagation through the network expression, we first focus on the action error, which is a linear combination of the hidden layer outputs. Applying the chain rule here yields

$$\frac{\partial E_a(t)}{\partial a(t)} = \sum_{i=1}^{N_h} \frac{\partial E_a(t)}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial a(t)} \quad (2.24)$$

The weights in the network are updated according to

$$\Delta w_{ia}^{(2)}(t) = l(t) \left[ \frac{-\partial E_a(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)} \right] \quad (2.25)$$

for the hidden layer to the actor nodes, where

$$\begin{aligned}
\frac{\partial E_a(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)} &= \sum_{i=1}^{N_h} \frac{\partial E_a(t)}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial x_i(t)} \frac{\partial x_i(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)} \\
&= \sum_{i=1}^{N_h} \frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial x_i(t)} \times \frac{\partial x_i(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ia}^{(2)}(t)} \\
&= e_a(t) \left( \sum_{i=1}^{N_h} w_{ci}^{(2)}(t) \left[ \frac{1}{2} (1 - y_i^2(t)) \right] \times w^{(1)}(t) y_i(t) \right) \tag{2.26}
\end{aligned}$$

and  $w_{ia}^{(1)}(t)$  denoting the weight between the action node and the  $i^{th}$  hidden node. Moreover, from the hidden layer to the critic node, we have

$$\Delta w_c^{(2)}(t) = l(t) \left[ \frac{-\partial E_c(t)}{\partial w_{ic}^{(2)}(t)} \right], \tag{2.27}$$

where

$$\begin{aligned}
\frac{\partial E_c(t)}{\partial w_{ic}^{(2)}(t)} &= \frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial w_{ic}^{(2)}(t)} \\
&= e_c(t) y_i(t). \tag{2.28}
\end{aligned}$$

Finally, for the inputs to the hidden layer, we express the weight update as

$$\Delta w_{ij}^{(1)}(t) = l(t) \left[ \frac{-\partial E(t)}{\partial w_{ij}^{(1)}(t)} \right] \tag{2.29}$$

where

$$\begin{aligned}
\frac{\partial E(t)}{\partial w_{ij}^{(1)}(t)} &= \frac{\partial E_c(t)}{\partial w_{ij}^{(1)}(t)} + \frac{\partial E_a(t)}{\partial a(t)} \frac{\partial a(t)}{\partial w_{ij}^{(1)}(t)} \\
&= [e_c(t) w_{ic}^{(2)}(t) + e_a(t) \left( \sum_{i=1}^{N_h} w_{ci}^{(2)}(t) \left[ \frac{1}{2} (1 - y_i^2(t)) \right] w_{ia}^{(1)}(t) \right) \\
&\quad \times w_{ia}^{(2)}(t) \left[ \frac{1}{2} (1 - y_i^2(t)) \right] u_j(t)] \tag{2.30}
\end{aligned}$$

It is noted that the temporal difference nature of the action correction formulation resembles the one employed in the value estimation portion of TD learning. That is, information obtained at

time  $t + 1$  is used to define the error correcting signals pertaining to time  $t$ .

## 2.4 Stochastic Meta-Descent

Stochastic meta-descent (SMD) was first presented in [20] as a modification to existing gradient descent techniques. Instead of using identical constant learning rates for all weight updates, SMD employs an independent learning rate for each. The weight update rule is now

$$w_{ij}(t+1) = w_{ij}(t) + \lambda_{ij}(t)\delta_{ij}(t) \quad (2.31)$$

where  $\lambda_{ij}(t)$  is the learning rate for  $w_{ij}$  at time  $t$ . It is updated as

$$\ln \lambda_{ij}(t) = \ln \lambda_{ij}(t-1) - \mu \frac{\partial J(t)}{\partial \ln \lambda_{ij}} \quad (2.32)$$

where  $\mu$  is the learning rate of the learning rate, or global meta-learning rate. Further, this equation can be rewritten as

$$\begin{aligned} \ln \lambda_{ij}(t) &= \ln \lambda_{ij}(t-1) - \mu \frac{\partial J(t)}{\partial w_{ij}(t)} \frac{\partial w_{ij}(t)}{\partial \ln \lambda_{ij}} \\ &= \ln \lambda_{ij}(t-1) - \mu \delta_{ij}(t) v_{ij}(t) \end{aligned} \quad (2.33)$$

where

$$v_{ij}(t) = \frac{\partial w_{ij}(t)}{\partial \ln \lambda_{ij}}. \quad (2.34)$$

Equation 2.32 can be further simplified under the assumption that, for small  $\mu$ ,  $e^\mu = 1 + \mu$  leaving

$$\ln \lambda_{ij}(t) = \ln \lambda_{ij}(t-1) \max(\rho, 1 + \mu \delta_{ij}(t) v_{ij}(t)) \quad (2.35)$$

where  $\rho$  protects against unreasonably small or negative values.  $v_{ij}$  measures the long-term impact of the change an individual learning rate has on its corresponding weight. The SMD algorithm defines  $v_{ij}$  as an exponential average of the effect of all past learning rates on the new weight and is of the form

$$v_{ij} = \sum_{k=0}^{\infty} \beta^k \frac{\partial w_{ij}(t+1)}{\partial \ln \lambda_{ij}(t-k)} \quad (2.36)$$

where  $\beta$  is between 0 and 1 and determines the time scale over which long-term dependencies take effect.

SMD is an improvement on gradient descent methods in that it reduces the amount of oscillation that takes place with a constant learning rate. It is an  $O(n)$  algorithm that permits adaptation of learning rates based on performance. Stochastic sampling helps avoid local minima in the optimization process. SMD has been applied to several different fields such as vision-based tracking [21] and scalable recurrent neural networks [22].

## Chapter 3

# Design Approach

### 3.1 Vision-Based Maze Navigation

This thesis poses the problem of vision-based maze navigation as a POMDP as discussed in section 2.1. The set of observations is comprised of images of the environment taken with the agent's on-board camera. The agent's task is to locate a pink ball in a maze comprised of green and black panels using its visual input and reinforcement learning methods.

Vision-based navigation is a complex, real world problem that poses a difficult challenge for machine learning agents. Input from a camera or other sensor is both high-dimensional and noisy. The high dimensionality of the raw image is a function of the number of pixels provided. Even small images are hundreds by hundreds of pixels making it impractical to apply each pixel as an input to the system. Noise is introduced both from the environment and by the agent. Shadows and different lighting in the environment can alter images significantly. The agent's own sensors and actuators can cause the same image to be distorted since the agent will never be in the exact same position twice. This means that the visual input received by the agent will never be exactly the same.

In this problem, the agent is intended to resemble the Sony AIBO robotic dog, depicted in figure 3-1. The robot has four legs that move to drive it forwards, backwards and to turn. The AIBO has an on-board camera located where a normal dog's mouth would be and is capable of turning its head to change the field of view. Consistent motion with the robot's legs is a non-trivial task on its own. The legs are likely to become snagged on cracks and velcro on the maze floor. This form of



Figure 3-1: Sony AIBO in a Maze Environment

motion will also introduce noise in the images since the camera will not be in the same orientation with respect to the horizon.

In this task, the agent must find its pink ball, which is hidden in a maze consisting of green and black tiles. This ball marks the goal state of the maze. The agent's action set consists of four movements: forward, backward, left and right. Each action taken will orient the agent appropriately so that it is facing the direction in which it just moved. For example, if the agent is facing north and moves left, it is located in the adjacent cell west of its current location and oriented to the west. Any action in the direction of an adjacent wall results in no movement. At the start of each episode, the agent is randomly located within the maze and will take an action at each time step. When the goal state is reached, the agent once again relocated randomly.

The choice of discrete actions in this task is to simplify the overall movements. There is still work to be done for continuous movement of the agent, and this is discussed in the concluding chapter. The coming sections describe not only the results of the CACM as applied to vision-based navigation, but also to these actions. A more realistic action set consists of forward motion and turning left, right, or backward. This set has been shown to increase the time required for the agent to converge to the optimal policy and is discussed in this chapter.

## 3.2 Design of the Machine Learning Agent

The agent is comprised of two primary modules: feature extraction and a consolidated actor-critic model. This architecture can be seen in figure 3-2. The image is fed to the agent as input  $o'(t)$  at time  $t$ . This image passes through a simple feature extraction routine to produce observations,  $o(t)$ , to be fed to the CACM. The agent then makes an action in the environment based on its policy to yield the next image as  $o'(t + 1)$ .

The consolidated actor-critic makes use of a neural network for policy and value function estimation. As discussed in section 2.2, neural networks are noise tolerant universal function approximators. This scheme will permit efficient real-time vision based maze navigation.

### 3.2.1 Feature Extraction

The feature extraction performed is a simple averaging and thresholding operation. The image is split into a 7x7 grid. For each cell, the pixels within are averaged to yield a single three-element vector. These are passed through a heuristic thresholding routine that produces a vector  $X$ . Each  $x \in X$  is of the set  $E = \{0, 1, 2\}$  to represent the three primary colors found in the maze. Here, 0 is used to represent the black squares found on the outer walls and the floor panels, 1 is for the green interior panels, and 2 is used to represent the pink ball for which the agent is looking. Figure 3-3 illustrates this process on an actual image from the simulation.

### 3.2.2 Correlated Exploration

Random walk exploration is a common scheme in machine learning agents. At every time step, there is some probability that the agent will either explore or not explore. It will either follow its policy for its next action, or it will select randomly from its action set. This probability is independent of whether or not it explored previously. One of the problems with the random walk is that it only produces localized exploration, which can slow the convergence of the agent.

Action selection in the agent is  $\epsilon$ -greedy, where the agent will explore with a probability  $\epsilon$ , but choose an action based on its policy the rest of the time. In this work, epsilon is a function of the number of time steps per episode, represented as  $t$ , and is given as

$$\epsilon(t) = 0.0413 + \frac{\log(t + 1)}{22.9361}. \quad (3.1)$$

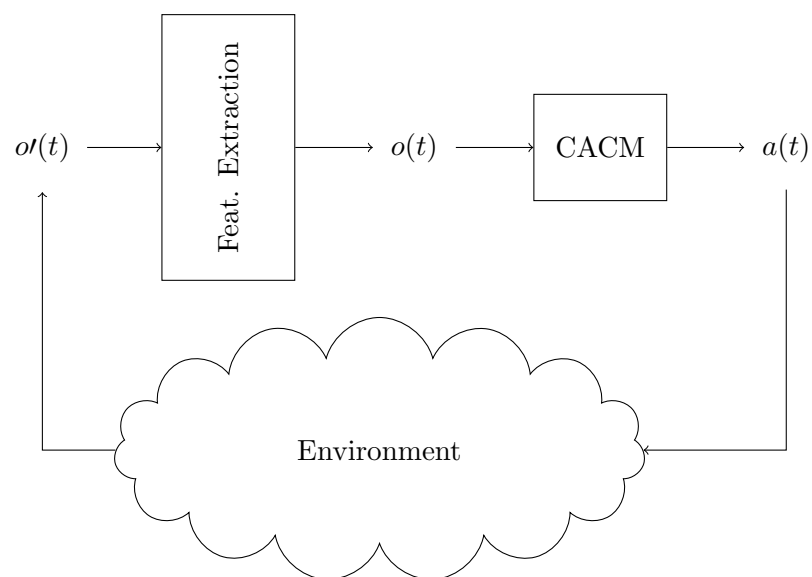


Figure 3-2: Block Diagram of the RL Agent and its Environment



|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3-3: Example observation converted to its feature array

Therefore, the agent will explore less as its policy improves.

While the decision to explore is an  $\epsilon$ -greedy process, the decision of which action to take is often uniformly selected from the action set. This results in a *random walk* through the state set. An alternative paradigm is explored in this thesis. Since the maze in question, as well as many other environments, is comprised of long corridors instead of open rooms, the sequence of states to reach the goal is correlated. Therefore, a random walk will result in very little motion in the direction to the goal state. In order to take advantage of the structure of the environment, a correlated exploration scheme is applied.

The correlated exploration scheme is implemented using a two-state Markov chain depicted in figure 3-4. The two states, A and B, represent choosing a new action or continue moving in the same direction respectively. This model is also often used in modeling bursty network traffic, where the two states represent the flow of data being either ON or OFF.

The average number of times the agent will attempt the same action is given as

$$B = \frac{1}{1 - \alpha} \quad (3.2)$$

and the mean time spent on correlated exploration when exploring is given as

$$\lambda = \frac{1 - \beta}{2 - \beta - \alpha} \quad (3.3)$$

Where  $\alpha$  is the probability of taking the same action once in that state and  $\beta$  the probability of remaining in the random exploration state. As the agent learns the policy, the odds of exploration decrease as in the case of the random walk. In order to sufficiently differentiate correlated ex-

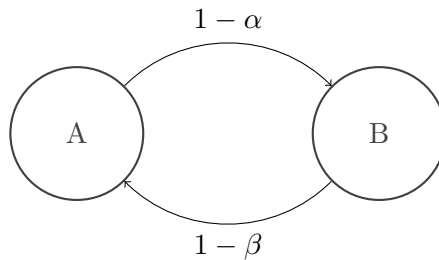


Figure 3-4: Two-state model for bursty exploration

ploration from selecting actions uniformly,  $\alpha$  must be high to keep the agent following a bursty scheme.

### 3.3 Simulation Descriptions

The simulations were carried out in two flavors. The first does not involve images from a real-world environment and is used for comparing modifications to the agent itself. The second uses a pair of mazes built in the Machine Intelligence Lab (MIL). For the second maze, the Sony AIBO was used to take pictures in each direction from each cell. The goal state is marked by the pink ball that comes with each robot. Two variations of the maze are shown in figures 3-5 and 3-6.

The agent is implemented with an Elman network. The recurrent connections provide the context necessary to form state estimations based on a series of observations. Since the agent is randomly relocated on each successful maze completion, invalidating any context, the memory neurons are reset. The agent must move 6 steps after relocation before learning can begin. This is so that the agent is able to make a valid state inference upon which to base its actions. Any action before proper context is formed cannot be used to train the agent, as the internal weight set will be updated with garbage on its inputs. Acting on the reward signal from these actions would introduce noise to the learning process. Actions are represented as two-element vectors with each element  $a \in -1, 1$ .

The simulations are organized into *trials* consisting of discrete time steps representing the transition between states. An *episode* is the set of state transitions from the start to the goal state. Each trial is comprised of many episodes, starting with an untrained agent and ending with an agent who has converged on an acceptable policy. For each episode, the mean squared error for the value function and the duration is recorded. These values are recorded for an entire trial with the duration kept as a rolling average over all episodes.

Each episode of the simulation is limited to 1000 steps. If the agent has not reached the goal by step 1000, it is randomly relocated in the maze and has to start over. On each successful episode, the agent receives a positive reward as discussed previously and relocated randomly as before. The agent is provided a reward signal from the environment based on its actions. For these simulations, the agent receives +5 for successful maze completion and 0 for all other states. There is no additional penalty for relocation as a result of the step limit.

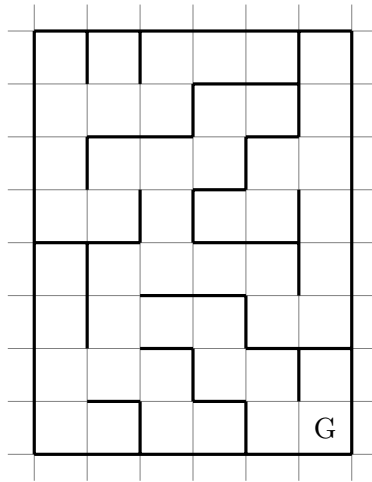


Figure 3-5: Maze A

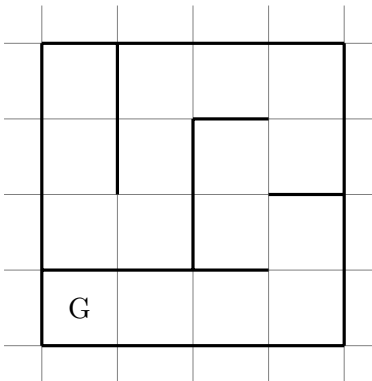


Figure 3-6: Maze B

## 3.4 Simulation Results

All errors provided in the following figures are calculated as mean squared error (MSE) where

$$MSE = \frac{1}{n} \sum_{i=1}^n e_i^2. \quad (3.4)$$

MSE makes a good metric of measurement because it is able to account for both the variance and bias of the error function.

### 3.4.1 Vision-Based Navigation with the CACM

As has been described in section 3.3, the agent consists of a feature extraction engine that feeds observations to a consolidated actor-critic. The CACM chooses an action to take according to its policy and affects the environment accordingly. Figures 3-7 and 3-8 show the Bellman MSE as well as the duration versus time step. It took over 20 million time steps, over 11 hours, before the CACM reaches a reasonable policy to find the goal state. This was one of the quicker simulations as the average run time is around 15 hours for maze A. As will be discussed in Chapter 4, this poses a problem for a live robotic agent that cannot move nearly as rapidly as the computer simulation.

### 3.4.2 Performance Comparison

The consolidated actor-critic is more computationally efficient than the traditional actor-critic model. Some of the work in this thesis has been focused on demonstrating this fact displayed in simple problems in [2]. Starting with simple grid-world navigation tasks, where maze walls are represented with 1's and openings with 0's, this work has progressed to a vision-based learning task. For time considerations, the results were gathered using the smaller of the two mazes considered.

For this comparison, the CACM was configured with 50 hidden neurons and the actor-critic with 100 (50 per network). During experimentation it was discovered that hidden neuron counts below 50 caused trials to run well over three times as long and below 40 ran for weeks before being manually terminated. More concisely, this means the traditional actor-critic is unable to effectively model the system given the same computational power as the CACM. If the actor-critic were to be run with the same number of hidden neurons, i.e. 25 per network, it would be unable to sufficiently model the environment.

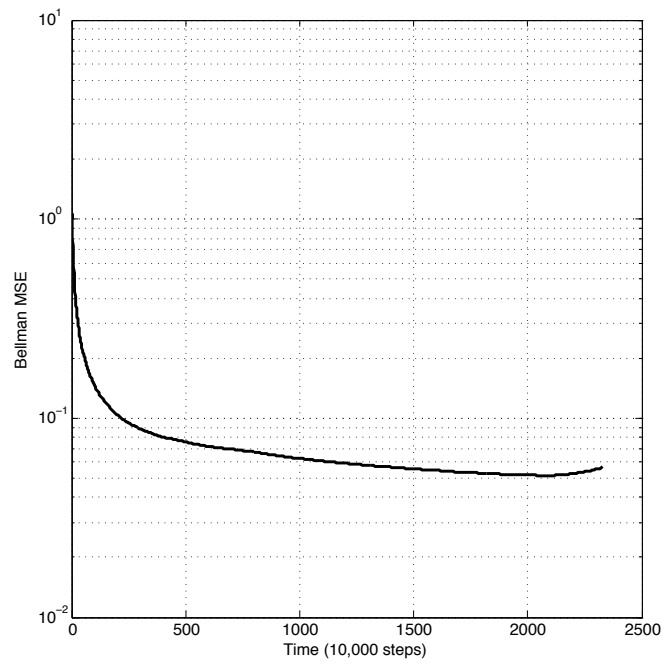


Figure 3-7: CACM Q MSE for maze A

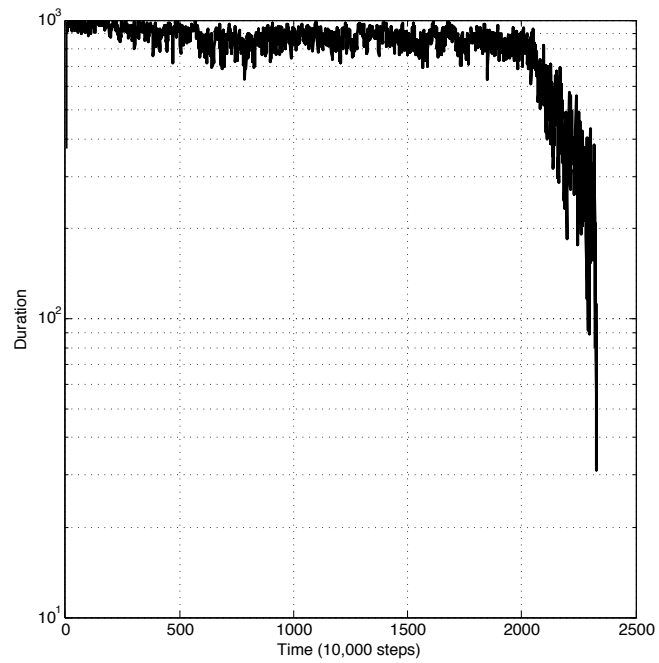


Figure 3-8: CACM Duration vs. Time Step for maze A

Figures 3-9 and 3-10 show the Bellman MSE and action MSE respectively. It can be observed that the Q error of the actor-critic is slightly lower than that of the CACM; however, in terms of the action error, the CACM is slightly improved. The on-processor time of the actor-critic scheme was over 2.8 hours. The on-processor time of the CACM was less than two-thirds the time at 1.7 hours.

Figure 3-11 shows the durations of both the CACM and actor-critic model. This plot shows that the average episode duration for the CACM was better than that of the actor-critic. In the actor-critic, there is no sharing of knowledge between the actor and critic, and therefore a dependence on the critic is built into the system. It must learn the value function before the actor is able to accurately form a model and perform action updates. This is a result of the dependence on the value function to determine the best action to take at a given time. The inconsistent errors introduce noise into learning the policy.

The Bellman error for the CACM was slightly worse than for the actor-critic. This can be explained by the perturbation of the action errors on the hidden layer neurons. Since the output of the value function and the action outputs are dependent on the same set of neurons, this error will introduce noise into the value function approximation. However, while the action error introduces noise to the value function, the learning of the value function will reduce the noise in the action approximation.

### 3.4.3 Impact of Action Set

Many problems have certain assumptions and constraints about the set of actions an agent is permitted to make. Many example problems of maze navigation present a basic set of actions: move North, move South, move East, move West. This action set is non-trivial to implement on the Sony AIBO robot, which already has difficulty maintaining straight paths between maze cells using its four legs for motion. Since this thesis focuses on a discrete time environment, obstacle avoidance and maintaining particular paths through the environment are not considered.

In the simulations in this thesis, a set of relative actions are used as described previously. These will move the agent relative to its current position. This eliminates built in knowledge about how the agent is positioned in its environment. Providing the agent a notion of absolute direction with the actions mentioned in the above paragraph is providing implicit extra information in the design.

When implemented in a robot, the action set would most likely consist of rotating in either of

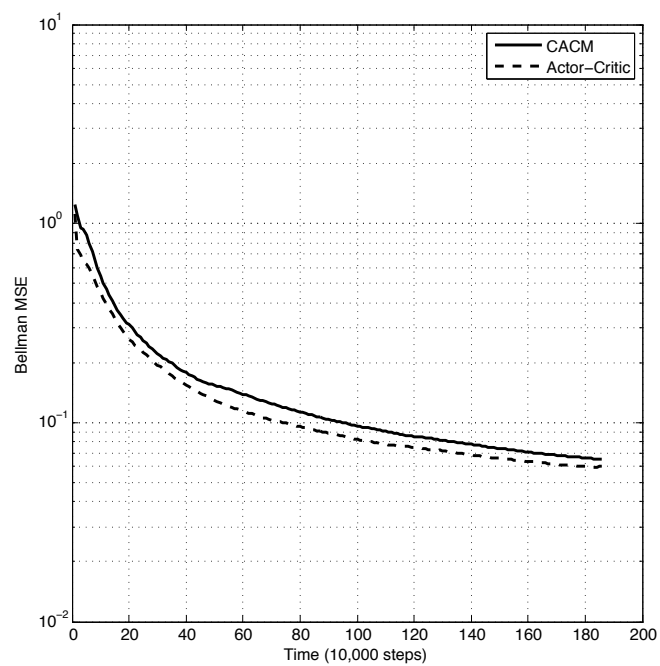


Figure 3-9: CACM vs. Actor-Critic Bellman MSE

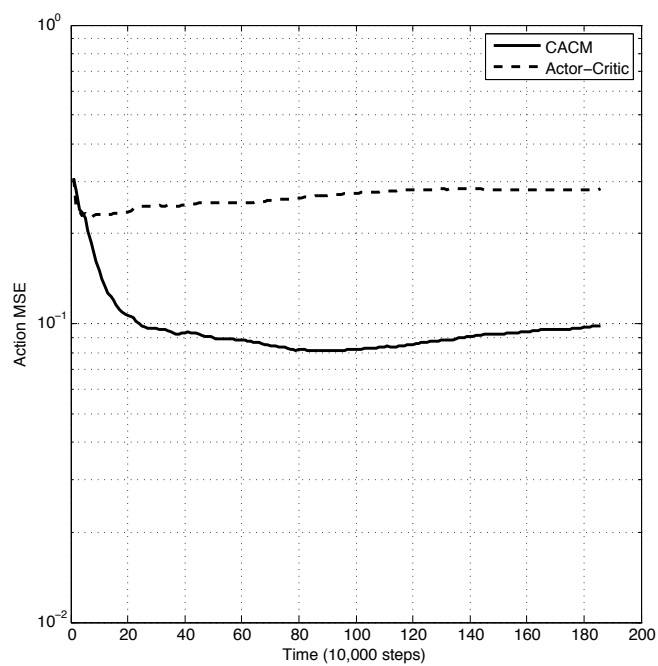


Figure 3-10: CACM vs. Actor-Critic Action MSE

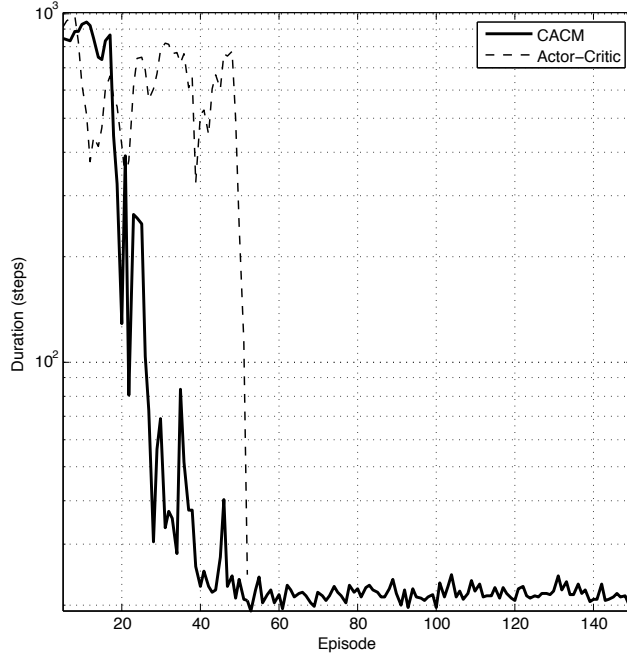


Figure 3-11: CACM vs. Actor-Critic Episode Duration

two directions or moving forward. The robot could also be permitted to rotate a full  $180^\circ$ . With this in mind, trials were run with the action set, consisting of: move forward, rotate left, rotate right, and turn around. So as to provide additional information, two new inputs are added to the CACM to act as a compass. These inputs hold the last action made by the agent.

Figures 3-12 and 3-13 show the resulting Bellman error and episode durations respectively. From these figures, it can be ascertained that the extra moves slow convergence with respect to the number of time steps required. This is understandable as the agent would spend much of its time rotating in place without advancing through the maze. This delays the reward received from the goal state and delays the entire learning process.

#### 3.4.4 Correlated Exploration

As discussed in Section 3.2.2, exploration is often performed as a random walk through the state set. This leaves an agent randomly exploring the states within a few actions away from its current state. Since the state set of many real-world problems exhibit order and structure, it is advantageous for artificial agents to take this into account.

This thesis introduces work on Markov-modulated, correlated exploration. In this scheme, the

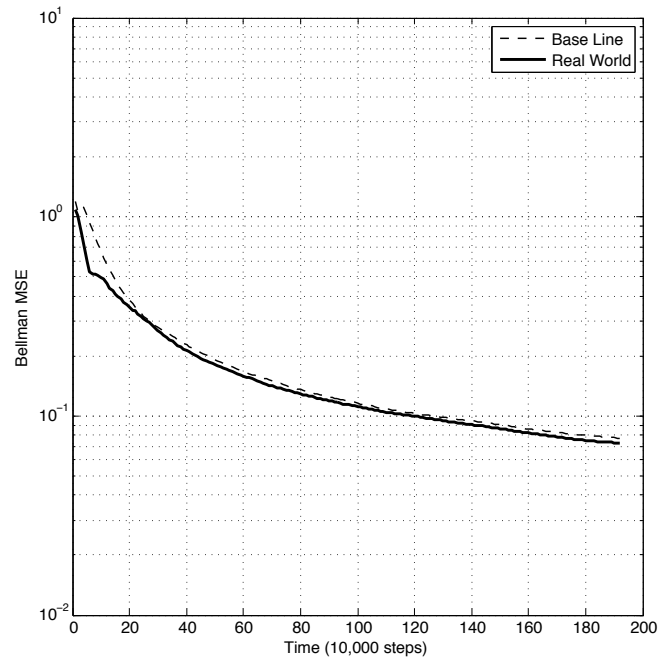


Figure 3-12: Bellman Error for Differing Action Sets

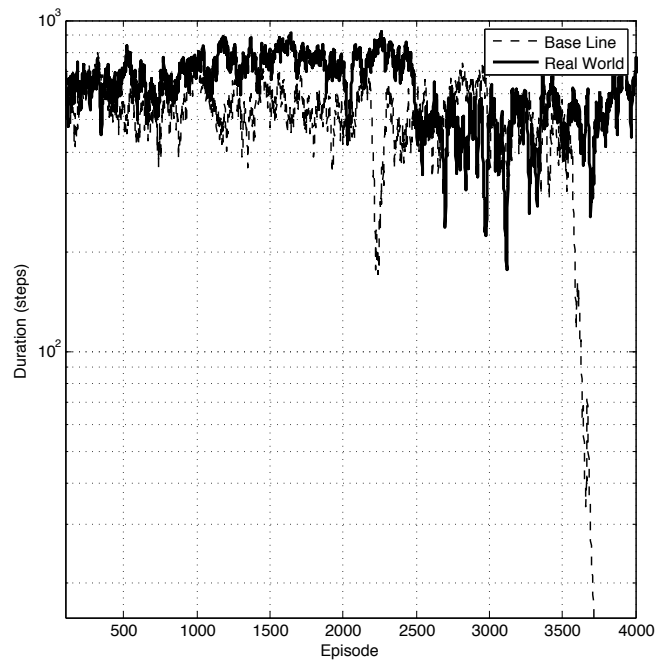


Figure 3-13: Duration per Episode for Differing Action Sets

agent’s exploratory actions are related to the actions previously taken. For the example of the maze, this means that the agent is able to take advantage of the structure of the maze. The correlated exploration should permit the agent to move in bursts through the maze instead of exploring its immediate surroundings.

Through trial and error,  $\alpha$  has been selected to be 0.8 and  $\beta$  to be 0.2. Table 3.1 shows the mean number of steps required before the agent reached the desired episode duration. From this table, it can be seen that Markov-modulated exploration requires the average duration of correlated exploration to be at least 2.5 to be equivalent with the random walk. In this case, the agent has a mean burst size of 5.00 according to 3.2.

Figures 3-14 and 3-15 show the distributions for correlated exploration and random walk exploration respectively. It is apparent from these figures that correlated exploration, when configured correctly, offers an impressive performance gain over the random walk. It is also interesting to point out that the distribution for the random walk is more spread out over the domain. The correlated exploration exhibits more of a normal, localized distribution. Figures 3-16 and 3-17 show the Bellman MSE and duration per episode respectively. Here again, the correlated exploration exhibits improved performance over the random walk. The durations for the random walk eventually dip lower than that of the correlated exploration, but correlated exploration is able to achieve a workable policy more quickly.

### 3.4.5 The Impact of Observation Prediction

The purpose of introducing observation prediction to the machine learning agent occurred initially as a test bed to see if the network had the approximation power to learn the underlying structure of its surroundings. Early simulations involved guiding the agent down pre-selected paths to see if it was capable of modelling its environment. This was later introduced to the final agent to study its affect on learning.

The assumption here is that since the functions being approximated by the neural network are based on the same underlying model, the environment, their mathematical representations share structure. Since the output neurons of the network are fully connected to the hidden-layer neurons, knowledge sharing between the outputs is possible. The observations received by the agent on each step are approximated so that the error signal from this output will drive the model to be more accurate. The more accurate the model, the more accurate the state inferences and therefore the

| Average Duration | Mean Time Steps (10,000 steps) |
|------------------|--------------------------------|
| 1.67             | 261.3                          |
| 2.00             | 88.7                           |
| 2.50             | 51.0                           |
| 3.33             | 35.8                           |
| 5.00             | 32.6                           |
| 10.00            | 31.4                           |

Table 3.1: Mean Duration for different configurations of Correlated exploration

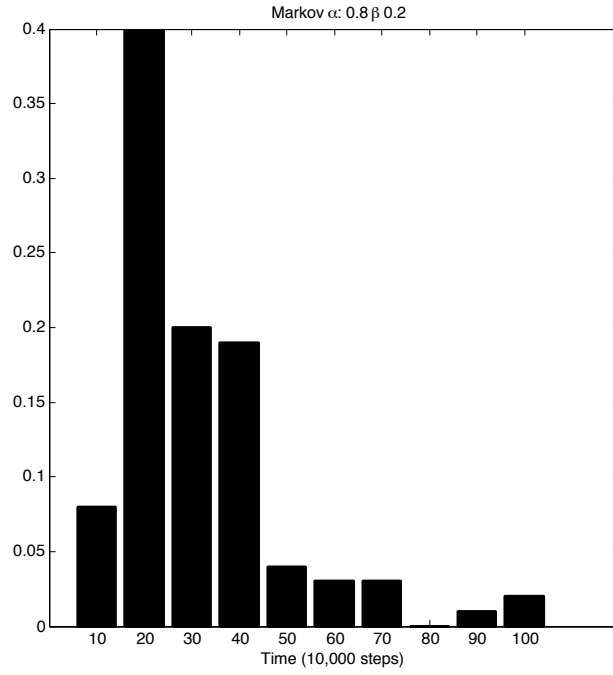


Figure 3-14: Distribution for Correlated Exploration

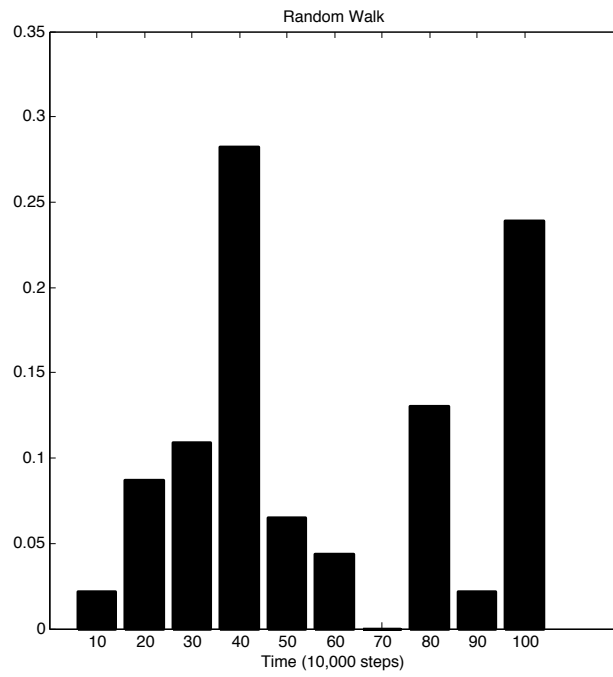


Figure 3-15: Distribution for the Random Walk

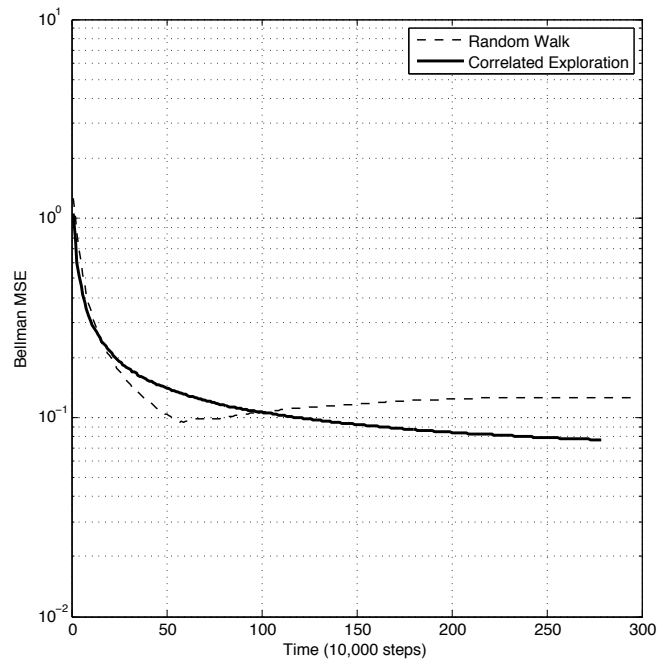


Figure 3-16: Bellman Error for Correlated Exploration

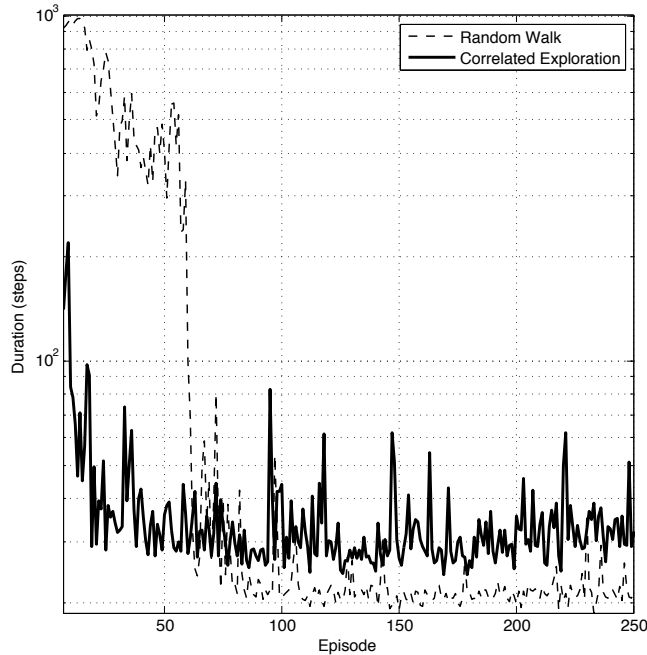


Figure 3-17: Episode Durations for Correlated and Random Walk Exploration

more accurate the value and policy functions.

Figures 3-18 and 3-19 show the value function for the baseline and CACM with observation prediction. Observation prediction, in the case of the simple grid-world maze, lowered the error in the value function. This results from the additional feedback from the environment in the form of the observation error.

As in the case of the CACM versus the actor-critic, introducing additional information to form the model of the environment will help with the convergence of the network. There are limits to this however, as the observation prediction slowed the initial convergence. The model without this feature could reach an acceptable level sooner, but the observation prediction will more accurately model the environment in the long run.

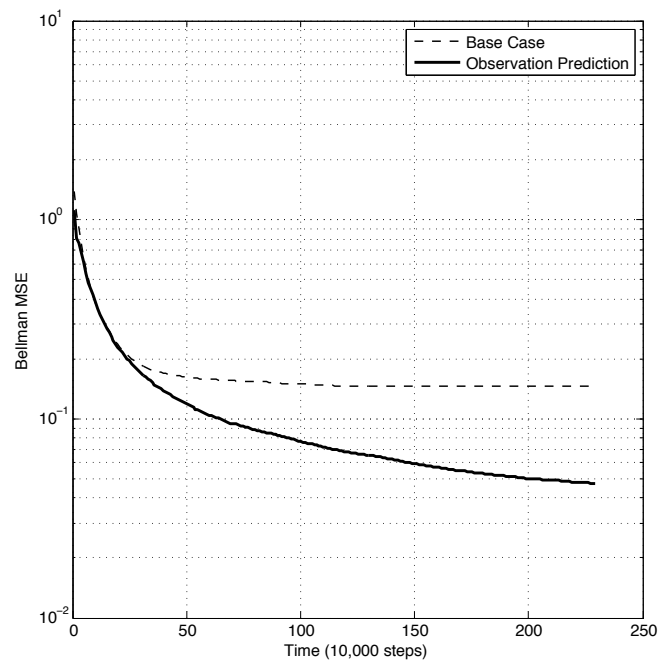


Figure 3-18: Bellman Error for Observation Prediction

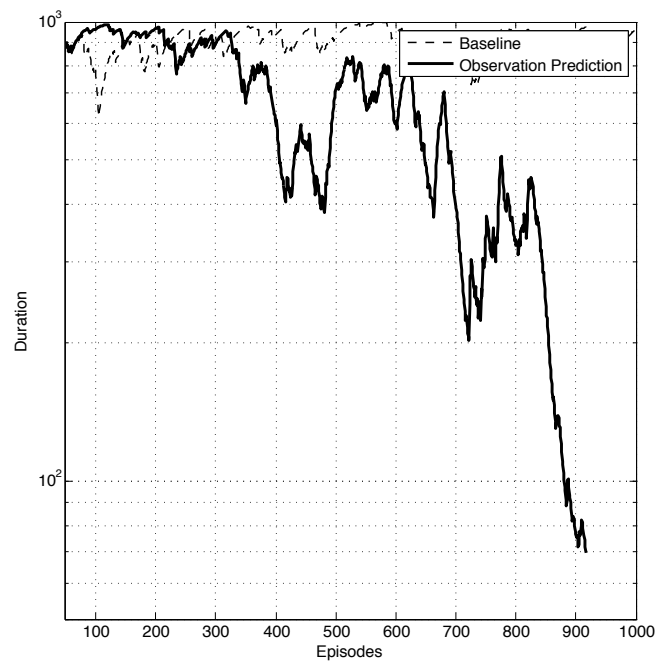


Figure 3-19: Duration per Episode for Observation Prediction

# Chapter 4

## Conclusions

### 4.1 Thesis Summary

This thesis describes the application of a consolidated actor-critic model to vision-based problem solving. The problems are poised as partially observable Markov decision processes, where inference about the agent’s state must be made based on observations from the environment. Several parameters of the agent were explored also in an attempt to reflect real-world action sets. Markov-modulated correlated exploration was also proposed and compared with random walk exploration.

It has been demonstrated that while the CACM is quite capable of learning the value function and policy for an environment, like many applications of reinforcement learning, it takes many iterations to get right. It will require simulation on a computer to bring the agent to a point where it is able enough to perform the real-world task so that it will complete its objective within acceptable time constraints. This is common among reinforcement learning agents. The agent will be first trained in simulation until the error has dropped sufficiently and then tried out in the real-world. There, the agent will adapt to the extra noise and inconsistency introduced by the sensors and actuators. This has had success as mentioned in the introduction to this paper.

Correlated exploration takes advantage of correlated state transitions in the environment. It has been shown here that this can improve the convergence of an RL agent when the proper mean burst size has been obtained. This improves the performance of the agent since the environment in which it is operating has implicit structure. State transitions to reach the goal are highly correlated.

The selection of a realistic action set increased the time to converge significantly. The new action

set negatively impacts the exploration method being used as the agent will spend significantly less time exploring the state space as the probability of transitioning between states is significantly reduced. This is another reason to begin training in simulation before the agent is released on its task, as real world trials would not complete soon enough.

The introduction of observation prediction in an agent also showed an improvement in overall agent performance. As in the comparison of CACM and actor-critic model, the introduction of new error to guide the development of the model within the agent improves convergence time. This occurs as a result of knowledge sharing between the two functions being approximated as all outputs of the network depend on the same set of hidden neurons.

## 4.2 Future Work

The next logical steps would be to take a Sony AIBO robot and run the CACM in real-time with the robot navigating a maze. This would take considerable time, so simulations would have to be run until the agents error is to sufficient levels. Once trained, the agent would be permitted to run through the maze environment. It would need to continue to learn as more noise is introduced through the real-world lighting conditions and imprecise nature of its four legged motion. A controller for the state transitions would have to be developed to maintain discrete set and action spaces.

The next extension from here would be to take the agent back into simulation to generate a set of actions and observations to better model a continuous environment. Once successful in simulation, this can be taken to the real-world maze and have the agent adapt to the robot's imperfection in navigation. This would then take on two different vision based tasks: maze navigation and maneuvering through an environment with imprecise actuation. Therefore, no external controller would be needed to guide the agent between states as was required previously, and any imperfections in the maze would have to be dealt with by the agent.

## 4.3 Relevant Publications

1. I. Elhanany, C. Niedzwiedz, Z. Liu, S. Livingston, "A Consolidated Actor-Critic Model for Partially-Observable Markov Decision Processes," IET Electronics Letters, October, 2008

2. C. Niedzwiedz, I. Elhanany, Z. Liu, S. Livingston, "A Consolidated Actor-Critic Model with Function Approximation for High-Dimensional POMDPs," AAAI 2008 workshop for Advancement in POMDP Solvers (part of the AAAI 2008 Conference), Chicago, July, 2008
3. C. Niedzwiedz, Z. Liu, I. Elhanany, S. Livingston, "A Consolidated Actor-Critic Model for Temporal Difference Learning with Function Approximation and Application to POMDPs," 3rd Northeast Student Colloquium on Artificial Intelligence, May, 2008

# Bibliography

# Bibliography

- [1] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, “Purposive behavior acquisition for a real robot by vision-based reinforcement learning,” in *Machine Learning*, 1994, pp. 279–303.
- [2] Z. Liu, “Hardware-efficient scalable reinforcement learning systems,” Ph.D. dissertation, The University of Tennessee, 2007.
- [3] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda, “Coordination of multiple behaviors acquired by a vision-based reinforcement learning,” in *In Proc. of IEEE International Conference on Intelligent Robots and Systems 1994 (IROS 1994)*, 1994, pp. 917–924.
- [4] G. A. Rummery and M. Niranjan, “On-line Q-learning using connectionist systems,” Cambridge University Engineering Department, Tech. Rep. CUED/F-INFENG/TR 166, 1994.
- [5] J. N. Tsitsiklis and B. V. Roy, “An analysis of temporal-difference learning with function approximation,” *IEEE Transactions on Automatic Control*, Tech. Rep., 1997.
- [6] G. Tesauro, “Temporal difference learning and td-gammon,” *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995.
- [7] R. Enns and J. Si, “Helicopter Trimming and Tracking Control Using Direct Neural Dynamic Programming,” *Neural Networks, IEEE Transactions on*, vol. 14, no. 4, pp. 929–939, 2003.
- [8] R. Bellman, “A markovian decision process,” *Journal of Mathematics and Mechanics*, no. 6, pp. 679–684, 1957.
- [9] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction*. The MIT Press, 1998.

- [10] R. Simmons and S. Koenig, “Probabilistic robot navigation in partially observable environments,” *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 55–56, 2000.
- [11] T. Darrell and A. Pentland, “Active gesture recognition using partially observable markov decision processes,” in *Proceedings of the 13th International Conference on Pattern Recognition, 1996*, vol. 3, 1996, pp. 984–988.
- [12] M. Hauskrecht and H. Fraser, “Planning treatment of ischemic heart disease with partially observable markov decision processes,” *Artificial Intelligence in Medicine*, vol. 18, pp. 221–244, 2000.
- [13] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.
- [14] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.
- [15] J. L. Elman, “Finding structure in time,” *Cognitive Science*, no. 14, pp. 179–211, 1990.
- [16] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, The University of Tennessee, 1989.
- [17] S. Singh, R. S. Sutton, and P. Kaelbling, “Reinforcement learning with replacing eligibility traces,” in *Machine Learning*, 1996, pp. 123–158.
- [18] J. Si, L. Yang, and D. Liu, “Direct Neural Dynamic Programming,” in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. G. Barto, W. B. Powell, and D. W. II, Eds. Institute of Electrical and Electronics Engineers, 2004, pp. 125–151.
- [19] I. H. Witten, “An adaptive optimal controller for discrete-time markov environments,” in *Information and Control*, 1977, pp. 386–295.
- [20] N. N. Schraudolph, “Local gain adaptation in stochastic gradient descent,” in *In Proc. Intl. Conf. Artificial Neural Networks*, 1999, pp. 569–574.
- [21] M. Bray, E. Koller-Meier, N. N. Schraudolph, and L. V. Gool, “Stochastic meta-descent for tracking articulated structures,” in *CVPRW ’04: Proceedings of the 2004 Conference on Com-*

*puter Vision and Pattern Recognition Workshop (CVPRW'04) Volume 1.* Washington, DC, USA: IEEE Computer Society, 2004, p. 7.

- [22] Z. Liu and I. Elhanany, “A fast and scalable recurrent neural network based on stochastic meta descent,” *IEEE Transactions on Neural Networks*, vol. 19, no. 9, pp. 1652–1658, 2008.

# Vita

Christopher Allen Niedzwiedz resides in Knoxville, TN. He is currently employed as a software engineer for Cisco Systems, Inc. A graduate of the University of Tennessee, he returned shortly after graduation to pursue a masters degree in computer engineering.